

PostGIS 2.4.0 Handbuch

SVN Revision ()

Contents

1	Einführung	1
1.1	Projektleitung	1
1.2	Aktuelle Kernentwickler	1
1.3	Frühere Hauptteilnehmer	2
1.4	Weitere Mitwirkende	2
1.5	Weitere Information	3
2	PostGIS Installation	4
2.1	Kurzfassung	4
2.2	Systemvoraussetzungen	6
2.3	Nutzung des Quellcodes	7
2.4	Kompilation und Installation mittels Quellcode: Detaillierte Beschreibung	7
2.4.1	Konfiguration	8
2.4.2	Build-Prozess	9
2.4.3	Build-Prozess für PostGIS Extensions und Softwareverteilung	9
2.4.4	Softwaretest	12
2.4.5	Installation	21
2.5	Eine Geodatenbank über EXTENSIONS anlegen	21
2.6	Ersellung einer Geodatenbank ohne Extensions	22
2.7	Installation und Verwendung des Adressennormierers	23
2.7.1	Installation von Regex::Assemble	23
2.8	Installation, Aktualisierung des Tiger Geokodierers und Daten laden	23
2.8.1	Aktivierung des Tiger Geokodierer in Ihrer PostGIS Datenbank: Verwendung von Extension	24
2.8.1.1	Umwandlung einer normalen Installation des Tiger-Geokodierers in das Extension Modell	26
2.8.2	Den Tiger Geokodierer in Ihrer PostGIS Datenbank aktivieren: Ohne die Verwendung von Extension	26
2.8.3	Die Adressennormierer-Extension zusammen mit dem Tiger Geokodierer verwenden	27
2.8.4	Tiger-Daten laden	27
2.8.5	Upgrade Ihrer Tiger Geokodierer Installation	27
2.9	Erzeugung einer Geodatenbank mit einem Template	28
2.10	Upgrading	28

2.10.1	Soft upgrade	29
2.10.1.1	Soft Upgrade Pre 9.1+ oder ohne Extensions/Erweiterungen	29
2.10.1.2	Soft Upgrade von PostGIS 9.1+ mittels EXTENSIONS	29
2.10.2	Hard upgrade	30
2.11	Übliche Probleme bei der Installation	31
2.12	Loader/Dumper	31
3	Häufige Fragen zu PostGIS	32
4	PostGIS Anwendung: Datenmanagement und Abfragen	37
4.1	GIS Objekte	37
4.1.1	OpenGIS WKB und WKT	37
4.1.2	PostGIS EWKB, EWKT und Normalformen/kanonische Formen	38
4.1.3	SQL-MM Part 3	39
4.2	PostGIS geographischer Datentyp	40
4.2.1	Geographie Grundlagen	40
4.2.2	Wann sollte man den GEOGRAPHY Datentyp dem GEOMETRY Datentyp vorziehen	42
4.2.3	Fortgeschrittene FAQ's zum GEOGRAPHY Datentyp	42
4.3	Verwendung von OGC-Standards	43
4.3.1	Die SPATIAL_REF_SYS Tabelle und Koordinatenreferenzsysteme	43
4.3.2	Die gespeicherte Abfrage GEOMETRY_COLUMNS	44
4.3.3	Erstellung einer räumlichen Tabelle	45
4.3.4	Geometrische Spalten in "geometry_columns" händisch registrieren	46
4.3.5	Wahrung der OGC-Konformität von Geometrien	49
4.3.6	DE-9IM-Matrix (DE-9IM)	53
4.3.6.1	Theorie	54
4.4	GIS (Vector) Daten laden	57
4.4.1	Daten via SQL laden	57
4.4.2	shp2pgsql: Using the ESRI Shapefile Loader	57
4.5	Abfrage von GIS-Daten	59
4.5.1	Daten via SQL abfragen	59
4.5.2	Verwendung des Dumper	60
4.6	Indizes aufbauen	61
4.6.1	GiST-Indizes	61
4.6.2	BRIN Indizes	62
4.6.3	Verwendung von Indizes	63
4.7	Komplexe Abfragen	63
4.7.1	Indizes nutzen	64
4.7.2	Beispiele für Spatial SQL	64

5	Rasterdaten: Verwaltung, Abfrage und Anwendung	67
5.1	Laden und Erstellen von Rastertabellen	67
5.1.1	Verwendung von raster2pgsql zum Laden von Rastern	67
5.1.2	Erzeugung von Rastern mit den PostGIS Rasterfunktionen	71
5.2	Raster Katalog	71
5.2.1	Rasterspalten Katalog	72
5.2.2	Raster Übersicht/Raster Overviews	73
5.3	Eigene Anwendungen mit PostGIS Raster erstellen	74
5.3.1	PHP Beispiel: Ausgabe mittels ST_AsPNG in Verbindung mit anderen Rasterfunktionen	74
5.3.2	ASP.NET C# Beispiel: Ausgabe mittels ST_AsPNG in Verbindung mit anderen Rasterfunktionen	74
5.3.3	Applikation für die Java-Konsole, welche eine Rasterabfrage als Bilddatei ausgibt	76
5.3.4	Verwenden Sie PLPython um Bilder via SQL herauszuschreiben	77
5.3.5	Faster mit PSQL ausgeben	78
6	Anwendung der PostGIS Geometrie: Applikationsentwicklung	79
6.1	Verwendung von MapServer	79
6.1.1	Grundlegende Handhabung	79
6.1.2	Häufig gestellte Fragen	80
6.1.3	Erweiterte Verwendung	81
6.1.4	Beispiele	82
6.2	Java Clients (JDBC)	83
6.3	C Clients (libpq)	85
6.3.1	Text Cursor	85
6.3.2	Binäre Cursor	85
7	Performance Tipps	86
7.1	Kleine Tabellen mit großen Geometrien	86
7.1.1	Problembeschreibung	86
7.1.2	Umgehungslösung	86
7.2	CLUSTER auf die geometrischen Indizes	87
7.3	Vermeidung von Dimensionsumrechnungen	87
7.4	Tunen der Konfiguration	88
7.4.1	Startup/Inbetriebnahme	88
7.4.2	Runtime/Laufzeit	88

8	PostGIS Referenz	90
8.1	PostgreSQL PostGIS Geometry/Geography/Box Types	90
8.1.1	box2d	90
8.1.2	box3d	90
8.1.3	geometry	92
8.1.4	geometry_dump	92
8.1.5	geography	92
8.2	PostGIS Grand Unified Custom Variables (GUCs)	93
8.2.1	postgis.backend	93
8.2.2	postgis.gdal_datapath	93
8.2.3	postgis.gdal_enabled_drivers	94
8.2.4	postgis.enable_outdb_rasters	96
8.3	Funktionen zur Verwaltung von Geometrien	96
8.3.1	AddGeometryColumn	96
8.3.2	DropGeometryColumn	98
8.3.3	DropGeometryTable	99
8.3.4	PostGIS_Full_Version	100
8.3.5	PostGIS_GEOS_Version	100
8.3.6	PostGIS_Liblwgeom_Version	101
8.3.7	PostGIS_LibXML_Version	101
8.3.8	PostGIS_Lib_Build_Date	102
8.3.9	PostGIS_Lib_Version	102
8.3.10	PostGIS_PROJ_Version	103
8.3.11	PostGIS_Scripts_Build_Date	103
8.3.12	PostGIS_Scripts_Installed	104
8.3.13	PostGIS_Scripts_Released	104
8.3.14	PostGIS_Version	105
8.3.15	Populate_Geometry_Columns	106
8.3.16	UpdateGeometrySRID	107
8.4	Geometrische Konstruktoren	108
8.4.1	ST_BdPolyFromText	108
8.4.2	ST_BdMPolyFromText	109
8.4.3	ST_Box2dFromGeoHash	110
8.4.4	ST_GeogFromText	110
8.4.5	ST_GeographyFromText	111
8.4.6	ST_GeogFromWKB	111
8.4.7	ST_GeomFromTWKB	112
8.4.8	ST_GeomCollFromText	113
8.4.9	ST_GeomFromEWKB	113

8.4.10	ST_GeomFromEWKT	115
8.4.11	ST_GeometryFromText	116
8.4.12	ST_GeomFromGeoHash	117
8.4.13	ST_GeomFromGML	118
8.4.14	ST_GeomFromGeoJSON	120
8.4.15	ST_GeomFromKML	121
8.4.16	ST_GMLToSQL	122
8.4.17	ST_GeomFromText	122
8.4.18	ST_GeomFromWKB	124
8.4.19	ST_LineFromEncodedPolyline	125
8.4.20	ST_LineFromMultiPoint	125
8.4.21	ST_LineFromText	126
8.4.22	ST_LineFromWKB	127
8.4.23	ST_LinestringFromWKB	128
8.4.24	ST_MakeBox2D	129
8.4.25	ST_3DMakeBox	129
8.4.26	ST_MakeLine	130
8.4.27	ST_MakeEnvelope	131
8.4.28	ST_MakePolygon	132
8.4.29	ST_MakePoint	134
8.4.30	ST_MakePointM	135
8.4.31	ST_MLineFromText	136
8.4.32	ST_MPointFromText	137
8.4.33	ST_MPolyFromText	137
8.4.34	ST_Point	138
8.4.35	ST_PointFromGeoHash	139
8.4.36	ST_PointFromText	140
8.4.37	ST_PointFromWKB	141
8.4.38	ST_Polygon	142
8.4.39	ST_PolygonFromText	143
8.4.40	ST_WKBToSQL	144
8.4.41	ST_WKTToSQL	144
8.5	Zugriffsfunktionen auf Geometrien	144
8.5.1	GeometryType	144
8.5.2	ST_Boundary	146
8.5.3	ST_CoordDim	148
8.5.4	ST_Dimension	149
8.5.5	ST_EndPoint	149
8.5.6	ST_Envelope	150

8.5.7	ST_BoundingDiagonal	151
8.5.8	ST_ExteriorRing	152
8.5.9	ST_GeometryN	153
8.5.10	ST_GeometryType	155
8.5.11	ST_InteriorRingN	157
8.5.12	ST_IsPolygonCCW	158
8.5.13	ST_IsPolygonCW	158
8.5.14	ST_IsClosed	159
8.5.15	ST_IsCollection	161
8.5.16	ST_IsEmpty	162
8.5.17	ST_IsRing	163
8.5.18	ST_IsSimple	164
8.5.19	ST_IsValid	165
8.5.20	ST_IsValidReason	166
8.5.21	ST_IsValidDetail	167
8.5.22	ST_M	168
8.5.23	ST_NDims	169
8.5.24	ST_NPoints	170
8.5.25	ST_NRings	170
8.5.26	ST_NumGeometries	171
8.5.27	ST_NumInteriorRings	172
8.5.28	ST_NumInteriorRing	173
8.5.29	ST_NumPatches	173
8.5.30	ST_NumPoints	174
8.5.31	ST_PatchN	174
8.5.32	ST_PointN	175
8.5.33	ST_Points	177
8.5.34	ST_SRID	177
8.5.35	ST_StartPoint	178
8.5.36	ST_Summary	179
8.5.37	ST_X	180
8.5.38	ST_XMax	181
8.5.39	ST_XMin	182
8.5.40	ST_Y	183
8.5.41	ST_YMax	184
8.5.42	ST_YMin	185
8.5.43	ST_Z	186
8.5.44	ST_ZMax	187
8.5.45	ST_Zmflag	188

8.5.46	ST_ZMin	189
8.6	Geometrische Editoren	190
8.6.1	ST_AddPoint	190
8.6.2	ST_Affine	190
8.6.3	ST_Force2D	192
8.6.4	ST_Force3D	193
8.6.5	ST_Force3DZ	194
8.6.6	ST_Force3DM	195
8.6.7	ST_Force4D	195
8.6.8	ST_ForcePolygonCCW	196
8.6.9	ST_ForceCollection	197
8.6.10	ST_ForcePolygonCW	198
8.6.11	ST_ForceSFS	198
8.6.12	ST_ForceRHR	199
8.6.13	ST_ForceCurve	199
8.6.14	ST_LineMerge	200
8.6.15	ST_CollectionExtract	201
8.6.16	ST_CollectionHomogenize	202
8.6.17	ST_Multi	203
8.6.18	ST_Normalize	203
8.6.19	ST_RemovePoint	204
8.6.20	ST_Reverse	205
8.6.21	ST_Rotate	205
8.6.22	ST_RotateX	206
8.6.23	ST_RotateY	207
8.6.24	ST_RotateZ	208
8.6.25	ST_Scale	209
8.6.26	ST_Segmentize	211
8.6.27	ST_SetPoint	212
8.6.28	ST_SetSRID	212
8.6.29	ST_SnapToGrid	213
8.6.30	ST_Snap	215
8.6.31	ST_Transform	217
8.6.32	ST_Translate	220
8.6.33	ST_TransScale	221
8.7	Geometrie Ausgabe	222
8.7.1	ST_AsBinary	222
8.7.2	ST_AsEncodedPolyline	224
8.7.3	ST_AsEWKB	225

8.7.4	ST_AsEWKT	226
8.7.5	ST_AsGeoJSON	227
8.7.6	ST_AsGML	229
8.7.7	ST_AsHEXEWKB	232
8.7.8	ST_AsKML	232
8.7.9	ST_AsLatLonText	234
8.7.10	ST_AsSVG	235
8.7.11	ST_AsText	236
8.7.12	ST_AsTWKB	237
8.7.13	ST_AsX3D	238
8.7.14	ST_GeoHash	241
8.7.15	ST_AsGeobuf	242
8.7.16	ST_AsMVTGeom	243
8.7.17	ST_AsMVT	243
8.8	Operatoren	245
8.8.1	&&	245
8.8.2	&&(geometry,box2df)	245
8.8.3	&&(box2df,geometry)	246
8.8.4	&&(box2df,box2df)	247
8.8.5	&&&	248
8.8.6	&&&(geometry,gidx)	249
8.8.7	&&&(gidx,geometry)	250
8.8.8	&&&(gidx,gidx)	251
8.8.9	&<	252
8.8.10	&< 	252
8.8.11	&>	253
8.8.12	<<	254
8.8.13	<< 	255
8.8.14	=	255
8.8.15	>>	257
8.8.16	@	258
8.8.17	@(geometry,box2df)	258
8.8.18	@(box2df,geometry)	259
8.8.19	@(box2df,box2df)	260
8.8.20	&>	261
8.8.21	>>	262
8.8.22	~	262
8.8.23	~(geometry,box2df)	263
8.8.24	~(box2df,geometry)	264

8.8.25	~(box2df,box2df)	265
8.8.26	~=	266
8.8.27	<->	266
8.8.28	!=	268
8.8.29	<#>	269
8.8.30	<<->>	271
8.8.31	<<#>>	271
8.9	Räumliche Beziehungen und Messungen	272
8.9.1	ST_3DClosestPoint	272
8.9.2	ST_3DDistance	273
8.9.3	ST_3DDWithin	274
8.9.4	ST_3DDFullyWithin	275
8.9.5	ST_3DIntersects	276
8.9.6	ST_3DLongestLine	277
8.9.7	ST_3DMaxDistance	278
8.9.8	ST_3DShortestLine	279
8.9.9	ST_Area	281
8.9.10	ST_Azimuth	282
8.9.11	ST_Centroid	284
8.9.12	ST_ClosestPoint	285
8.9.13	ST_ClusterDBSCAN	287
8.9.14	ST_ClusterIntersecting	289
8.9.15	ST_ClusterKMeans	289
8.9.16	ST_ClusterWithin	290
8.9.17	ST_Contains	291
8.9.18	ST_ContainsProperly	294
8.9.19	ST_Covers	295
8.9.20	ST_CoveredBy	297
8.9.21	ST_Crosses	298
8.9.22	ST_LineCrossingDirection	300
8.9.23	ST_Disjoint	302
8.9.24	ST_Distance	303
8.9.25	ST_MinimumClearance	305
8.9.26	ST_MinimumClearanceLine	306
8.9.27	ST_HausdorffDistance	306
8.9.28	ST_FrechetDistance	307
8.9.29	ST_MaxDistance	308
8.9.30	ST_DistanceSphere	309
8.9.31	ST_DistanceSpheroid	310

8.9.32	ST_DFullyWithin	311
8.9.33	ST_DWithin	312
8.9.34	ST_Equals	313
8.9.35	ST_GeometricMedian	314
8.9.36	ST_HasArc	315
8.9.37	ST_Intersects	316
8.9.38	ST_Length	317
8.9.39	ST_Length2D	319
8.9.40	ST_3DLength	319
8.9.41	ST_LengthSpheroid	320
8.9.42	ST_Length2D_Spheroid	321
8.9.43	ST_LongestLine	322
8.9.44	ST_OrderingEquals	324
8.9.45	ST_Overlaps	325
8.9.46	ST_Perimeter	327
8.9.47	ST_Perimeter2D	329
8.9.48	ST_3DPerimeter	329
8.9.49	ST_PointOnSurface	330
8.9.50	ST_Project	331
8.9.51	ST_Relate	331
8.9.52	ST_RelateMatch	333
8.9.53	ST_ShortestLine	334
8.9.54	ST_Touches	335
8.9.55	ST_Within	337
8.10	SFCGAL Funktionen	339
8.10.1	postgis_sfcgal_version	339
8.10.2	ST_Extrude	339
8.10.3	ST_StraightSkeleton	341
8.10.4	ST_ApproximateMedialAxis	342
8.10.5	ST_IsPlanar	343
8.10.6	ST_Orientation	343
8.10.7	ST_ForceLHR	343
8.10.8	ST_MinkowskiSum	344
8.10.9	ST_3DIntersection	346
8.10.10	ST_3DDifference	348
8.10.11	ST_3DUnion	349
8.10.12	ST_3DArea	350
8.10.13	ST_Tessellate	351
8.10.14	ST_Volume	353

8.10.15 ST_MakeSolid	354
8.10.16 ST_IsSolid	354
8.11 Geometrieverarbeitung	355
8.11.1 ST_Buffer	355
8.11.2 ST_BuildArea	359
8.11.3 ST_ClipByBox2D	360
8.11.4 ST_Collect	361
8.11.5 ST_ConcaveHull	363
8.11.6 ST_ConvexHull	367
8.11.7 ST_CurveToLine	368
8.11.8 ST_DelaunayTriangles	371
8.11.9 ST_Difference	376
8.11.10 ST_Dump	377
8.11.11 ST_DumpPoints	379
8.11.12 ST_DumpRings	383
8.11.13 ST_FlipCoordinates	384
8.11.14 ST_GeneratePoints	385
8.11.15 ST_Intersection	386
8.11.16 ST_LineToCurve	389
8.11.17 ST_MakeValid	390
8.11.18 ST_MemUnion	391
8.11.19 ST_MinimumBoundingCircle	391
8.11.20 ST_MinimumBoundingRadius	393
8.11.21 ST_Polygonize	394
8.11.22 ST_Node	395
8.11.23 ST_OffsetCurve	396
8.11.24 ST_RemoveRepeatedPoints	399
8.11.25 ST_SharedPaths	400
8.11.26 ST_ShiftLongitude	402
8.11.27 ST_WrapX	403
8.11.28 ST_Simplify	403
8.11.29 ST_SimplifyPreserveTopology	404
8.11.30 ST_SimplifyVW	405
8.11.31 ST_SetEffectiveArea	406
8.11.32 ST_Split	408
8.11.33 ST_SymDifference	410
8.11.34 ST_Subdivide	412
8.11.35 ST_SwapOrdinates	414
8.11.36 ST_Union	415

8.11.37 ST_UnaryUnion	418
8.11.38 ST_VoronoiLines	418
8.11.39 ST_VoronoiPolygons	419
8.12 Lineare Referenzierung	422
8.12.1 ST_LineInterpolatePoint	422
8.12.2 ST_LineLocatePoint	424
8.12.3 ST_LineSubstring	425
8.12.4 ST_LocateAlong	427
8.12.5 ST_LocateBetween	428
8.12.6 ST_LocateBetweenElevations	429
8.12.7 ST_InterpolatePoint	429
8.12.8 ST_AddMeasure	430
8.13 Unterstützung zeitbezogener Daten	431
8.13.1 ST_IsValidTrajectory	431
8.13.2 ST_ClosestPointOfApproach	432
8.13.3 ST_DistanceCPA	433
8.13.4 ST_CPAWithin	433
8.14 Unterstützung für lange andauernde Transaktionen/Long Transactions	434
8.14.1 AddAuth	434
8.14.2 CheckAuth	435
8.14.3 DisableLongTransactions	436
8.14.4 EnableLongTransactions	436
8.14.5 LockRow	437
8.14.6 UnlockRows	438
8.15 Sonstige Funktionen	438
8.15.1 ST_Accum	438
8.15.2 Box2D	439
8.15.3 Box3D	440
8.15.4 ST_EstimatedExtent	441
8.15.5 ST_Expand	442
8.15.6 ST_Extent	443
8.15.7 ST_3DExtent	444
8.15.8 Find_SRID	446
8.15.9 ST_MemSize	446
8.15.10 ST_PointInsideCircle	447
8.16 Außergewöhnliche Funktionen	448
8.16.1 PostGIS_AddBBox	448
8.16.2 PostGIS_DropBBox	449
8.16.3 PostGIS_HasBBox	450

9	Raster Referenz	451
9.1	Datentypen zur Unterstützung von Rastern.	452
9.1.1	geomval	452
9.1.2	addbandarg	452
9.1.3	rastbandarg	452
9.1.4	raster	453
9.1.5	reclassarg	453
9.1.6	summarystats	454
9.1.7	unionarg	454
9.2	Rastermanagement	455
9.2.1	AddRasterConstraints	455
9.2.2	DropRasterConstraints	457
9.2.3	AddOverviewConstraints	458
9.2.4	DropOverviewConstraints	459
9.2.5	PostGIS_GDAL_Version	459
9.2.6	PostGIS_Raster_Lib_Build_Date	459
9.2.7	PostGIS_Raster_Lib_Version	460
9.2.8	ST_GDALDrivers	460
9.2.9	UpdateRasterSRID	465
9.2.10	ST_CreateOverview	465
9.3	Raster Constructors	466
9.3.1	ST_AddBand	466
9.3.2	ST_AsRaster	468
9.3.3	ST_Band	471
9.3.4	ST_MakeEmptyCoverage	472
9.3.5	ST_MakeEmptyRaster	473
9.3.6	ST_Tile	474
9.3.7	ST_Retile	477
9.3.8	ST_FromGDALRaster	477
9.4	Zugriffsfunktionen auf Raster	478
9.4.1	ST_GeoReference	478
9.4.2	ST_Height	479
9.4.3	ST_IsEmpty	480
9.4.4	ST_MemSize	480
9.4.5	ST_MetaData	481
9.4.6	ST_NumBands	482
9.4.7	ST_PixelHeight	482
9.4.8	ST_PixelWidth	483
9.4.9	ST_ScaleX	484

9.4.10	ST_ScaleY	485
9.4.11	ST_RasterToWorldCoord	485
9.4.12	ST_RasterToWorldCoordX	486
9.4.13	ST_RasterToWorldCoordY	487
9.4.14	ST_Rotation	488
9.4.15	ST_SkewX	489
9.4.16	ST_SkewY	489
9.4.17	ST_SRID	490
9.4.18	ST_Summary	491
9.4.19	ST_UpperLeftX	491
9.4.20	ST_UpperLeftY	492
9.4.21	ST_Width	492
9.4.22	ST_WorldToRasterCoord	493
9.4.23	ST_WorldToRasterCoordX	494
9.4.24	ST_WorldToRasterCoordY	494
9.5	Zugriffsfunktionen auf Rasterbänder	495
9.5.1	ST_BandMetaData	495
9.5.2	ST_BandNoDataValue	496
9.5.3	ST_BandIsNoData	496
9.5.4	ST_BandPath	498
9.5.5	ST_BandPixelType	498
9.5.6	ST_HasNoBand	499
9.6	Zugriffsfunktionen und Änderungsmethoden für Rasterpixel	500
9.6.1	ST_PixelAsPolygon	500
9.6.2	ST_PixelAsPolygons	500
9.6.3	ST_PixelAsPoint	501
9.6.4	ST_PixelAsPoints	502
9.6.5	ST_PixelAsCentroid	503
9.6.6	ST_PixelAsCentroids	503
9.6.7	ST_Value	505
9.6.8	ST_NearestValue	507
9.6.9	ST_Neighborhood	509
9.6.10	ST_SetValue	511
9.6.11	ST_SetValues	512
9.6.12	ST_DumpValues	521
9.6.13	ST_PixelOfValue	522
9.7	Raster Editoren	523
9.7.1	ST_SetGeoReference	523
9.7.2	ST_SetRotation	524

9.7.3	ST_SetScale	525
9.7.4	ST_SetSkew	526
9.7.5	ST_SetSRID	527
9.7.6	ST_SetUpperLeft	527
9.7.7	ST_Resample	528
9.7.8	ST_Rescale	529
9.7.9	ST_Reskew	530
9.7.10	ST_SnapToGrid	531
9.7.11	ST_Resize	532
9.7.12	ST_Transform	534
9.8	Editoren für Rasterbänder	536
9.8.1	ST_SetBandNoDataValue	536
9.8.2	ST_SetBandIsNoData	537
9.9	Rasterband Statistik und Analytik	539
9.9.1	ST_Count	539
9.9.2	ST_CountAgg	539
9.9.3	ST_Histogram	541
9.9.4	ST_Quantile	542
9.9.5	ST_SummaryStats	544
9.9.6	ST_SummaryStatsAgg	546
9.9.7	ST_ValueCount	548
9.10	Ausgabe von Rastern	550
9.10.1	ST_AsBinary	550
9.10.2	ST_AsGDALRaster	551
9.10.3	ST_AsJPEG	552
9.10.4	ST_AsPNG	553
9.10.5	ST_AsTIFF	554
9.11	Rasterdatenverarbeitung	555
9.11.1	Map Algebra	555
9.11.1.1	ST_Clip	555
9.11.1.2	ST_ColorMap	557
9.11.1.3	ST_Intersection	560
9.11.1.4	ST_MapAlgebra	562
9.11.1.5	ST_MapAlgebra	569
9.11.1.6	ST_MapAlgebraExpr	571
9.11.1.7	ST_MapAlgebraExpr	573
9.11.1.8	ST_MapAlgebraFct	578
9.11.1.9	ST_MapAlgebraFct	582
9.11.1.10	ST_MapAlgebraFctNgb	586

9.11.1.11	ST_Reclass	588
9.11.1.12	ST_Union	589
9.11.2	Integrierte Map Algebra Callback Funktionen	591
9.11.2.1	ST_Distinct4ma	591
9.11.2.2	ST_InvDistWeight4ma	592
9.11.2.3	ST_Max4ma	592
9.11.2.4	ST_Mean4ma	593
9.11.2.5	ST_Min4ma	595
9.11.2.6	ST_MinDist4ma	596
9.11.2.7	ST_Range4ma	596
9.11.2.8	ST_StdDev4ma	597
9.11.2.9	ST_Sum4ma	598
9.11.3	DHM (Digitales Höhenmodell)	599
9.11.3.1	ST_Aspect	599
9.11.3.2	ST_HillShade	601
9.11.3.3	ST_Roughness	603
9.11.3.4	ST_Slope	603
9.11.3.5	ST_TPI	605
9.11.3.6	ST_TRI	606
9.11.4	Raster to Geometry	606
9.11.4.1	Box3D	606
9.11.4.2	ST_ConvexHull	607
9.11.4.3	ST_DumpAsPolygons	608
9.11.4.4	ST_Envelope	609
9.11.4.5	ST_MinConvexHull	609
9.11.4.6	ST_Polygon	611
9.12	Rasteroperatoren	612
9.12.1	&&	612
9.12.2	&<	613
9.12.3	&>	613
9.12.4	=	614
9.12.5	@	614
9.12.6	~=	615
9.12.7	~	615
9.13	Räumliche Beziehungen von Rastern und Rasterbändern	616
9.13.1	ST_Contains	616
9.13.2	ST_ContainsProperly	617
9.13.3	ST_Covers	618
9.13.4	ST_CoveredBy	619

9.13.5	ST_Disjoint	620
9.13.6	ST_Intersects	621
9.13.7	ST_Overlaps	621
9.13.8	ST_Touches	622
9.13.9	ST_SameAlignment	623
9.13.10	ST_NotSameAlignmentReason	624
9.13.11	ST_Within	625
9.13.12	ST_DWithin	626
9.13.13	ST_DFullyWithin	627
10	Häufige Fragen zu PostGIS Raster	629
11	Topologie	633
11.1	Topologische Datentypen	633
11.1.1	getfaceedges_returntype	633
11.1.2	TopoGeometry	634
11.1.3	validatetopology_returntype	634
11.2	Topologische Domänen	635
11.2.1	TopoElement	635
11.2.2	TopoElementArray	635
11.3	Verwaltung von Topologie und TopoGeometry	636
11.3.1	AddTopoGeometryColumn	636
11.3.2	DropTopology	637
11.3.3	DropTopoGeometryColumn	638
11.3.4	Populate_Topology_Layer	638
11.3.5	TopologySummary	639
11.3.6	ValidateTopology	640
11.4	Topologie Konstruktoren	641
11.4.1	CreateTopology	641
11.4.2	CopyTopology	642
11.4.3	ST_InitTopoGeo	642
11.4.4	ST_CreateTopoGeo	643
11.4.5	TopoGeo_AddPoint	644
11.4.6	TopoGeo_AddLineString	644
11.4.7	TopoGeo_AddPolygon	644
11.5	Topologie Editoren	645
11.5.1	ST_AddIsoNode	645
11.5.2	ST_AddIsoEdge	645
11.5.3	ST_AddEdgeNewFaces	646

11.5.4	ST_AddEdgeModFace	647
11.5.5	ST_RemEdgeNewFace	647
11.5.6	ST_RemEdgeModFace	648
11.5.7	ST_ChangeEdgeGeom	649
11.5.8	ST_ModEdgeSplit	649
11.5.9	ST_ModEdgeHeal	650
11.5.10	ST_NewEdgeHeal	651
11.5.11	ST_MoveIsoNode	651
11.5.12	ST_NewEdgesSplit	652
11.5.13	ST_RemoveIsoNode	653
11.5.14	ST_RemoveIsoEdge	653
11.6	Zugriffsfunktionen zur Topologie	654
11.6.1	GetEdgeByPoint	654
11.6.2	GetFaceByPoint	655
11.6.3	GetNodeByPoint	656
11.6.4	GetTopologyID	657
11.6.5	GetTopologySRID	657
11.6.6	GetTopologyName	658
11.6.7	ST_GetFaceEdges	658
11.6.8	ST_GetFaceGeometry	659
11.6.9	GetRingEdges	660
11.6.10	GetNodeEdges	660
11.7	Topologie Verarbeitung	661
11.7.1	Polygonize	661
11.7.2	AddNode	661
11.7.3	AddEdge	662
11.7.4	AddFace	663
11.7.5	ST_Simplify	665
11.8	TopoGeometry Konstruktoren	666
11.8.1	CreateTopoGeom	666
11.8.2	toTopoGeom	667
11.8.3	TopoElementArray_Agg	669
11.9	TopoGeometry Editoren	669
11.9.1	clearTopoGeom	669
11.9.2	TopoGeom_addElement	670
11.9.3	TopoGeom_remElement	670
11.9.4	toTopoGeom	670
11.10	TopoGeometry Accessors	671
11.10.1	GetTopoGeomElementArray	671

11.10.2 GetTopoGeomElements	671
11.11 TopoGeometry Ausgabe	672
11.11.1 AsGML	672
11.11.2 AsTopoJSON	674
11.12 Räumliche Beziehungen einer Topologie	676
11.12.1 Equals	676
11.12.2 Intersects	676
12 Adressennormierer	678
12.1 Funktionsweise des Parsers	678
12.2 Adressennormierer Datentypen	678
12.2.1 stdaddr	678
12.3 Adressennormierer Tabellen	679
12.3.1 rules Tabelle	679
12.3.2 lex Tabelle	682
12.3.3 gaz Tabelle	682
12.4 Adressennormierer Funktionen	683
12.4.1 parse_address	683
12.4.2 standardize_address	684
13 PostGIS Extras	687
13.1 Tiger Geokoder	687
13.1.1 Drop_Indexes_Generate_Script	687
13.1.2 Drop_Nation_Tables_Generate_Script	688
13.1.3 Drop_State_Tables_Generate_Script	689
13.1.4 Geocode	690
13.1.5 Geocode_Intersection	692
13.1.6 Get_Geocode_Setting	693
13.1.7 Get_Tract	694
13.1.8 Install_Missing_Indexes	695
13.1.9 Loader_Generate_Census_Script	695
13.1.10 Loader_Generate_Script	697
13.1.11 Loader_Generate_Nation_Script	699
13.1.12 Missing_Indexes_Generate_Script	700
13.1.13 Normalize_Address	701
13.1.14 Page_Normalize_Address	702
13.1.15 Pprint_Addy	704
13.1.16 Reverse_Geocode	705
13.1.17 Topology_Load_Tiger	706
13.1.18 Set_Geocode_Setting	708

14 PostGIS Special Functions Index	710
14.1 PostGIS Aggregate Functions	710
14.2 PostGIS Window Functions	710
14.3 PostGIS SQL-MM Compliant Functions	711
14.4 PostGIS Geography Support Functions	716
14.5 PostGIS Raster Support Functions	717
14.6 PostGIS Geometry / Geography / Raster Dump Functions	722
14.7 PostGIS Box Functions	723
14.8 PostGIS Functions that support 3D	724
14.9 PostGIS Curved Geometry Support Functions	729
14.10 PostGIS Polyhedral Surface Support Functions	732
14.11 PostGIS Function Support Matrix	736
14.12 New, Enhanced or changed PostGIS Functions	745
14.12.1 PostGIS Functions new or enhanced in 2.4	745
14.12.2 PostGIS Functions new or enhanced in 2.3	746
14.12.3 PostGIS Functions new or enhanced in 2.2	747
14.12.4 PostGIS functions breaking changes in 2.2	748
14.12.5 PostGIS Functions new or enhanced in 2.1	748
14.12.6 PostGIS functions breaking changes in 2.1	749
14.12.7 PostGIS Functions new, behavior changed, or enhanced in 2.0	749
14.12.8 PostGIS Functions changed behavior in 2.0	750
14.12.9 PostGIS Functions new, behavior changed, or enhanced in 1.5	751
14.12.10 PostGIS Functions new, behavior changed, or enhanced in 1.4	751
14.12.11 PostGIS Functions new in 1.3	751
15 Meldung von Problemen	752
15.1 Software Bugs melden	752
15.2 Probleme mit der Dokumentation melden	752
A Anhang	754
A.1 Release 2.4.0	754
A.1.1 Neue Funktionalität	754
A.1.2 Verbesserungen und Fehlerbehebung	754
A.1.3 Wichtige Änderungen	755
A.2 Release 2.3.3	755
A.2.1 Fehlerbehebung und Verbesserungen	755
A.3 Release 2.3.2	756
A.3.1 Fehlerbehebung und Verbesserungen	756
A.4 Release 2.3.1	756

A.4.1 Fehlerbehebung und Verbesserungen	756
A.5 Release 2.3.0	756
A.5.1 Wichtige Änderungen	756
A.5.2 Neue Funktionalität	757
A.5.3 Bugfixes	757
A.5.4 Verbesserung der Rechenleistung	757
A.6 Release 2.2.2	758
A.6.1 Neue Funktionalität	758
A.7 Release 2.2.1	758
A.7.1 Neue Funktionalität	758
A.8 Release 2.2.0	759
A.8.1 Neue Funktionalität	759
A.8.2 Verbesserungen	760
A.9 Release 2.1.8	761
A.9.1 Bugfixes	761
A.10 Release 2.1.7	761
A.10.1 Bugfixes	761
A.11 Release 2.1.6	762
A.11.1 Verbesserungen	762
A.11.2 Bugfixes	762
A.12 Release 2.1.5	762
A.12.1 Verbesserungen	762
A.12.2 Bugfixes	762
A.13 Release 2.1.4	763
A.13.1 Verbesserungen	763
A.13.2 Bugfixes	763
A.14 Release 2.1.3	764
A.14.1 Wichtige Änderungen	764
A.14.2 Bugfixes	764
A.15 Release 2.1.2	764
A.15.1 Bugfixes	764
A.15.2 Verbesserungen	765
A.16 Release 2.1.1	765
A.16.1 Wichtige Änderungen	765
A.16.2 Bugfixes	765
A.16.3 Verbesserungen	765
A.17 Release 2.1.0	766
A.17.1 Wichtige Änderungen	766
A.17.2 Neue Funktionalität	766

A.17.3 Verbesserungen	768
A.17.4 Bugfixes	769
A.17.5 Known Issues	770
A.18 Release 2.0.5	770
A.18.1 Bugfixes	770
A.18.2 Wichtige Änderungen	770
A.19 Release 2.0.4	771
A.19.1 Bugfixes	771
A.19.2 Verbesserungen	771
A.19.3 Known Issues	771
A.20 Release 2.0.3	772
A.20.1 Bugfixes	772
A.20.2 Verbesserungen	772
A.21 Release 2.0.2	772
A.21.1 Bugfixes	772
A.21.2 Verbesserungen	773
A.22 Release 2.0.1	774
A.22.1 Bugfixes	774
A.22.2 Verbesserungen	775
A.23 Release 2.0.0	775
A.23.1 Tester - Unsere heimlichen Helden	775
A.23.2 Wichtige Änderungen	775
A.23.3 Neue Funktionalität	776
A.23.4 Verbesserungen	776
A.23.5 Bugfixes	777
A.23.6 Release specific credits	777
A.24 Release 1.5.4	777
A.24.1 Bugfixes	777
A.25 Release 1.5.3	778
A.25.1 Bugfixes	778
A.26 Release 1.5.2	778
A.26.1 Bugfixes	779
A.27 Release 1.5.1	779
A.27.1 Bugfixes	779
A.28 Release 1.5.0	780
A.28.1 API Stabilität	780
A.28.2 Kompatibilität	780
A.28.3 Neue Funktionalität	780
A.28.4 Verbesserungen	781

A.28.5 Bugfixes	781
A.29 Release 1.4.0	781
A.29.1 API Stabilität	781
A.29.2 Kompatibilität	781
A.29.3 Neue Funktionalität	781
A.29.4 Verbesserungen	782
A.29.5 Bugfixes	782
A.30 Release 1.3.6	782
A.31 Release 1.3.5	782
A.32 Release 1.3.4	783
A.33 Release 1.3.3	783
A.34 Release 1.3.2	783
A.35 Release 1.3.1	783
A.36 Release 1.3.0	783
A.36.1 Zusätzliche Funktionalität	783
A.36.2 Verbesserung der Rechenleistung	783
A.36.3 Sonstige Änderungen	784
A.37 Release 1.2.1	784
A.37.1 Änderungen	784
A.38 Release 1.2.0	784
A.38.1 Änderungen	784
A.39 Release 1.1.6	784
A.39.1 Upgraden	784
A.39.2 Bugfixes	785
A.39.3 Sonstige Änderungen	785
A.40 Release 1.1.5	785
A.40.1 Upgraden	785
A.40.2 Bugfixes	785
A.40.3 Neue Funktionalität	785
A.41 Release 1.1.4	785
A.41.1 Upgraden	786
A.41.2 Bugfixes	786
A.41.3 Java Änderungen	786
A.42 Release 1.1.3	786
A.42.1 Upgraden	786
A.42.2 Bugfixes / Fehlerfreiheit	786
A.42.3 Neue Funktionalität	787
A.42.4 JDBC Änderungen	787
A.42.5 Sonstige Änderungen	787

A.43 Release 1.1.2	787
A.43.1 Upgraden	787
A.43.2 Bugfixes	787
A.43.3 Neue Funktionalität	788
A.43.4 Sonstige Änderungen	788
A.44 Release 1.1.1	788
A.44.1 Upgraden	788
A.44.2 Bugfixes	788
A.44.3 Neue Funktionalität	788
A.45 Release 1.1.0	789
A.45.1 Danksagungen	789
A.45.2 Upgraden	789
A.45.3 Neue Funktionen	789
A.45.4 Bugfixes	790
A.45.5 Semantische Funktionsänderungen	790
A.45.6 Verbesserung der Rechenleistung	790
A.45.7 JDBC2 funktioniert	790
A.45.8 Sonstige Neuigkeiten	790
A.45.9 Sonstige Änderungen	790
A.46 Release 1.0.6	791
A.46.1 Upgraden	791
A.46.2 Bugfixes	791
A.46.3 Verbesserungen	791
A.47 Release 1.0.5	791
A.47.1 Upgraden	791
A.47.2 Bibliotheksänderungen	792
A.47.3 Änderungen beim Loader	792
A.47.4 Sonstige Änderungen	792
A.48 Release 1.0.4	792
A.48.1 Upgraden	792
A.48.2 Bugfixes	792
A.48.3 Verbesserungen	793
A.49 Release 1.0.3	793
A.49.1 Upgraden	793
A.49.2 Bugfixes	793
A.49.3 Verbesserungen	793
A.50 Release 1.0.2	793
A.50.1 Upgraden	794
A.50.2 Bugfixes	794

A.50.3 Verbesserungen	794
A.51 Release 1.0.1	794
A.51.1 Upgraden	794
A.51.2 Bibliotheksänderungen	794
A.51.3 Sonstige Änderungen/Ergänzungen	794
A.52 Release 1.0.0	795
A.52.1 Upgraden	795
A.52.2 Bibliotheksänderungen	795
A.52.3 Sonstige Änderungen/Ergänzungen	795
A.53 Release 1.0.0RC6	795
A.53.1 Upgraden	795
A.53.2 Bibliotheksänderungen	795
A.53.3 Skriptänderungen	795
A.53.4 Sonstige Änderungen	796
A.54 Release 1.0.0RC5	796
A.54.1 Upgraden	796
A.54.2 Bibliotheksänderungen	796
A.54.3 Sonstige Änderungen	796
A.55 Release 1.0.0RC4	796
A.55.1 Upgraden	796
A.55.2 Bibliotheksänderungen	796
A.55.3 Skriptänderungen	797
A.55.4 Sonstige Änderungen	797
A.56 Release 1.0.0RC3	797
A.56.1 Upgraden	797
A.56.2 Bibliotheksänderungen	797
A.56.3 Skriptänderungen	797
A.56.4 JDBC Änderungen	798
A.56.5 Sonstige Änderungen	798
A.57 Release 1.0.0RC2	798
A.57.1 Upgraden	798
A.57.2 Bibliotheksänderungen	798
A.57.3 Skriptänderungen	798
A.57.4 Sonstige Änderungen	799
A.58 Release 1.0.0RC1	799
A.58.1 Upgraden	799
A.58.2 Änderungen	799

Abstract

PostGIS ist eine Erweiterung des objektrelationalen PostgreSQL Datenbanksystems und ermöglicht so die Speicherung von GIS (Geographisches Informationssystem) Objekten in der Datenbank. PostGIS unterstützt räumliche, GIST-basierte R-Tree Indizes, sowie Funktionen für die Analyse und Bearbeitung von GIS Objekten.



Dies ist das Handbuch für die Version 2.4.0



Diese Arbeit ist unter der [Creative Commons Attribution-Share Alike 3.0 License](https://creativecommons.org/licenses/by-sa/3.0/) lizenziert. Sie können den Inhalt ungeniert nutzen, aber wir ersuchen Sie das PostGIS Projekt namentlich aufzuführen und wenn möglich einen Verweis auf <http://postgis.net> zu setzen.

Chapter 1

Einführung

PostGIS wurde von Refrations Research Inc, im Rahmen eines Forschungsprojektes über Geodatenbanktechnologie entwickelt. Refrations ist ein Beratungsunternehmen im GIS- und Datenbankbereich, das auf Datenintegration und kundenspezifische Softwareentwicklung spezialisiert ist und in Victoria, British Columbia, Kanada beheimatet ist. Wir beabsichtigen die Unterstützung und Entwicklung von PostGIS, um eine Reihe wichtiger GIS-Funktionalitäten, einschließlich OpenGIS Kompatibilität, hochentwickelte topologische Konstrukte (Coverages, Oberflächen, Netzwerke), Desktop Benutzeroberflächen zum darstellen und bearbeiten von GIS Daten sowie Werkzeuge für den Zugriff via Internettechnologie zu unterstützen.

PostGIS ist ein Inkubationsprojekt der OSGeo Foundation. PostGIS wird von vielen FOSS4G-Entwicklern und Unternehmen auf der ganzen Welt laufend verbessert bzw. finanziert, da diese einen großen Nutzen aus dessen Funktionalität und Vielseitigkeit ziehen.

1.1 Projektleitung

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

Mark Cave-Ayland Koordiniert die Wartung und Fehlerbehebung, das Abgleichen mit PostgreSQL Releases, die Auswahl und das Binden von räumlichen Indizes, Loader/Dumper und die Shapfile Loader GUI, die Einbindung von neuen Funktionen sowie die Verbesserung von Funktionen.

Regina Obe Buildbot Wartung, kompiliert produktive und experimentelle Softwarepakete für Windows, allgemeine Unterstützung von Anwendern auf der PostGIS Newsgroup, Mitarbeit an X3D, Tiger Geokodierer, an Funktionen zur Verwaltung von Geometrien, und Smoke testing von neuer Funktionalität und wichtigen Änderungen am Code.

Bborie Park Entwicklung im Bereich Raster, Integration mit GDAL, Raster-Lader, Anwender-Support, allgemeine Fehlerbehebung, Softwaretests auf verschiedenen Betriebssystemen (Slackware, Mac, Windows und andere).

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

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

1.2 Aktuelle Kernentwickler

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

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

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

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

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

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

Pierre Racine Gesamtarchitektur für Raster, Prototyping, Unterstützung bei der Programmierung

1.3 Frühere Hauptteilnehmer

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

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

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

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

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

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

1.4 Weitere Mitwirkende

Die einzelnen Mitwirkenden In alphabetischer Reihenfolge: Alex Bodnaru, Alex Mayrhofer, Andrea Peri, Andreas Forø Tollefsen, Andreas Neumann, Anne Ghisla, Barbara Phillipot, Ben Jubb, Bernhard Reiter, Brian Hamlin, Bruce Rindahl, Bruno Wolff III, Bryce L. Nordgren, Carl Anderson, Charlie Savage, Dane Springmeyer, David Skea, David Techer, Eduin Carrillo, Even Rouault, Frank Warmerdam, George Silva, Gerald Fenoy, Gino Lucrezi, Guillaume Lelarge, IIDA Tetsushi, Ingvild Nystuen, Jason Smith, Jeff Adams, Jose Carlos Martinez Llari, Julien Rouhaud, Kashif Rasul, Klaus Foerster, Kris Jurka, Leo Hsu, Loic Dachary, Luca S. Percich, Maria Arias de Reyna, Mark Sondheim, Markus Schaber, Maxime Guillaud, Maxime van Noppen, Michael Fuhr, Mike Toews, Nathan Wagner, Nathaniel Clay, Nikita Shulga, Norman Vine, Rafal Magda, Ralph Mason, Rémi Cura, Richard Greenwood, Silvio Grosso, Steffen Macke, Stephen Frost, Tom van Tilburg, Vincent Mora, Vincent Picavet

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

In alphabetischer Reihenfolge: Arrival 3D, Associazione Italiana per l'Informazione Geografica Libera (GFOSS.it), AusVet, Avencia, Azavea, Cadcorp, CampToCamp, CartoDB, City of Boston (DND), Clever Elephant Solutions, Cooperativa Alveo, Deimos Space, Faunalia, Geographic Data BC, Hunter Systems Group, Lidwala Consulting Engineers, LisaSoft, Logical Tracking & Tracing International AG, Maponics, Michigan Tech Research Institute, Natural Resources Canada, Norwegian Forest and Landscape Institute, Boundless (former OpenGeo), OSGeo, Oslandia, Palantir Technologies, Paragon Corporation, R3 GIS, Refrations Research, Regione Toscana - SITA, Safe Software, Sirius Corporation plc, Stadt Uster, UC Davis Center for Vectorborne Diseases, University of Laval, U.S Department of State (HIU), Zonar Systems

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

PostGIS 2.0.0 war die erste Version mit der wir diese Strategie verfolgten. Wir benutzten [PledgeBank](#) und hatten zwei erfolgreiche Kampagnen.

[postgistopology](#) - mehr als 10 Sponsoren förderten mit jeweils \$250 USD die Entwicklung von TopoGeometry Funktionen und das Aufmöbeln der Topologie-Unterstützung für 2.0.0. Ist passiert.

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

Wichtige Support-Bibliotheken Die [GEOS](#) Bibliothek zur Handhabung von Geometrien, und die Arbeit an den Algorithmen von Martin Davis die erst alles zum Laufen brachte, laufende Wartung und Unterstützung durch Mateusz Loskot, Sandro Santilli (strk), Paul Ramsey und andere.

Die [GDAL](#) Geospatial Data Abstraction Library von Frank Warmerdam und anderen ist die Grundlage für einen großen Teil der Rasterfunktionalität, die mit PostGIS 2.0.0 eingeführt wurde. Im Prinzip werden die zur Unterstützung von PostGIS nötigen Neuerungen in GDAL, an das GDAL-Projekt zurückgegeben.

Die [Proj4](#) Bibliothek für kartographische Projektionen, und die Arbeit von Gerald Evenden und Frank Warmerdam bei der Erstellung und Erhaltung dieser.

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

1.5 Weitere Information

- Die neueste Software, Dokumentation und Neuigkeiten finden Sie auf der PostGIS Webseite <http://postgis.net>.
- Weitere Information zur GEOS Geometriebearbeitungs-Bibliothekgeometry finden Sie unter <http://trac.osgeo.org/geos/>.
- Weitere Information zur Proj4 Koordinatentransformations-Bibliothek finden Sie unter <http://trac.osgeo.org/proj/>.
- Weitere Information zum PostgreSQL Datenbankserver finden Sie auf der PostgreSQL Hauptseite <http://www.postgresql.org>.
- Weitere Information zum GIST-Index finden Sie auf der PostgreSQL GiST Entwicklerseite: <http://www.sai.msu.su/~megeera/postgres/gist/>.
- Weitere Information zu MapServer finden sie unter <http://mapserver.org>.
- Die "[Simple Features Specification for SQL](#)" finden Sie auf der Webseite des OpenGIS Consortiums: <http://www.opengeospatial.org/>.

Chapter 2

PostGIS Installation

Dieses Kapitel erläutert die notwendigen Schritte zur Installation von PostGIS.

2.1 Kurzfassung

Zum Kompilieren müssen die Abhängigkeiten im Suchpfad eingetragen sein:

```
tar xvfz postgis-2.4.0.tar.gz
cd postgis-2.4.0
./configure
make
make install
```

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

**Note**

Die Unterstützung von Rasterdaten in PostGIS ist zur Zeit optional, wird aber standardmäßig installiert. Um die Rasterunterstützung einzuschalten wird das PostgreSQL 9.1+ "Extensions Model" benötigt. Die Aktivierung mittels Extension ist die bevorzugte Methode. Die Aktivierung von PostGIS in der Datenbank erfolgt folgendermaßen:

```
psql -d yourdatabase -c "CREATE EXTENSION postgis;"
psql -d yourdatabase -c "CREATE EXTENSION postgis_topology;"
-- Installation mit sfcgal --
psql -d yourdatabase -c "CREATE EXTENSION postgis_sfcgal;"

-- Installation von tiger geocoder --
psql -d yourdatabase -c "CREATE EXTENSION fuzzystrmatch"
psql -d yourdatabase -c "CREATE EXTENSION postgis_tiger_geocoder;"

-- Für die Installation mit pcre
-- wird auch die address standardizer extension benötigt
psql -d yourdatabase -c "CREATE EXTENSION address_standardizer;"
```

Unter Section 2.4.3 sind die Details beschrieben, wie man die installierten/vorhandenen Extensions abfragen und upgraden kann, bzw. wie man von einer non-extension Installation zu einer Extension Installation kommt.

Für alle, die aus irgendeinem Grund PostGIS ohne Raster Unterstützung kompilieren wollen, oder die einfach nur ein bißchen altmodisch sind, die längere und schmerzhaftere Anleitung:

Nach der Installation befinden sich alle .sql Dateien unter dem Ordner "share/contrib/postgis-2.3" der PostgreSQL Installation.

```
createdb yourdatabase
createlang plpgsql yourdatabase
psql -d yourdatabase -f postgis.sql
psql -d yourdatabase -f postgis_comments.sql
psql -d yourdatabase -f spatial_ref_sys.sql
psql -d yourdatabase -f topology.sql
psql -d yourdatabase -f topology_comments.sql

-- nur bei Kompilation mit Raster (GDAL)
psql -d yourdatabase -f rtpostgis.sql
psql -d yourdatabase -f raster_comments.sql

-- falls mit sfcgal Unterstützung --
psql -d yourdatabase -f sfcgal.sql
psql -d yourdatabase -f sfcgal_comments.sql
```

Der Rest des Kapitels betrachtet die oberen Installationsschritte im Detail.

Ab PostGIS 2.1.3 sind out-of-db Raster und alle Raster Treiber standardmäßig ausgeschaltet. Um diese zu aktivieren müssen folgende Umgebungsvariablen `POSTGIS_GDAL_ENABLED_DRIVERS` and `POSTGIS_ENABLE_OUTDB_RASTERS` am Server gesetzt werden. Für PostGIS 2.2 kann ein plattformübergreifender Ansatz gewählt werden, indem der entsprechende [Section 8.2](#) gesetzt wird.

Falls Offline-Raster ermöglicht werden sollen:

```
POSTGIS_ENABLE_OUTDB_RASTERS=1
```

Jede andere Einstellung oder keine Einstellung deaktiviert out-of-db Raster.

Um alle in der jeweiligen GDAL Installation verfügbaren Treiber zu aktivieren, muss folgende Umgebungsvariable gesetzt werden:

```
POSTGIS_GDAL_ENABLED_DRIVERS=ENABLE_ALL
```

Falls nur bestimmte Treiber aktiviert werden sollen, kann die Umgebungsvariable auf diese beschränkt werden:

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



Note

Unter Windows darf die Treiberliste nicht unter Anführungszeichen gesetzt werden.

Die Zuweisung von Umgebungsvariablen wechselt je nach Betriebssystem. Für PostgreSQL Installationen auf Ubuntu oder Debian via `apt-postgresql`, ist der bevorzugte Weg `/etc/postgresql/9.3/main/environment` zu editieren, wobei 9.3 auf die PostgreSQL Version verweist und main auf den Cluster hinweist.

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

Nachdem die Umgebungsvariablen gesetzt sind, ist ein Neustart des PostgreSQL-Dienstes notwendig, damit die Änderungen wirksam werden.

2.2 Systemvoraussetzungen

PostGIS hat folgende Systemanforderungen für die Kompilation und zur Anwendung:

Notwendige Systemvoraussetzungen

- PostgreSQL 9.3 oder höher. Es wird eine vollständige PostgreSQL Installation (inklusive Server headers) benötigt. PostgreSQL steht unter <http://www.postgresql.org> zur Verfügung.
Welche PostgreSQL Version von welcher PostGIS Version unterstützt wird und welche PostGIS Version von welcher GEOS Version unterstützt wird findet sich unter <http://trac.osgeo.org/postgis/wiki/UsersWikiPostgreSQLPostGIS>
- GNU C Compiler (`gcc`). Es können auch andere ANSI C Compiler zur PostGIS Kompilation verwendet werden, aber die Kompilation mit `gcc` macht die geringsten Probleme.
- GNU Make (`gmake` oder `make`). Für viele Systeme ist GNU `make` die Standardversion von `make`. Überprüfe die Version durch `make -v`. Andere Versionen von `make` können das PostGIS `Makefile` nicht richtig ausführen.
- Proj4 reprojection library, version 4.6.0 or greater. Proj4 4.9 or above is needed to take advantage of improved geodetic. The Proj4 library is used to provide coordinate reprojection support within PostGIS. Proj4 is available for download from <http://trac.osgeo.org/proj/>.
- Die Geometriebibliothek GEOS, Version 3.4 oder höher, aber GEOS 3.7+ wird empfohlen um alle neuen Funktionen und Möglichkeiten zur Gänze ausnutzen zu können. Sie sollten zumindest GEOS 3.5 installiert haben, ansonsten werden Sie ein paar wesentliche Erweiterungen, wie `ST_ClipByBox2D` oder `ST_Subdivide` vermissen. GEOS kann von <http://trac.osgeo.org/geos/> heruntergeladen werden und da 3.5+ abwärtskompatibel mit älteren Versionen ist, ist ein Upgrade ziemlich sicher.
- LibXML2, Version 2.5.x oder höher. LibXML2 wird derzeit für einige Import Funktionen genutzt (`ST_GeomFromGML` und `ST_GeomFromKML`). LibXML2 steht unter <http://xmlsoft.org/downloads.html> zur Verfügung.
- JSON-C, Version 0.9 oder höher. JSON-C wird zurzeit benutzt um GeoJSON über die Funktion `ST_GeomFromGeoJson` zu importieren. JSON-C kann unter <https://github.com/json-c/json-c/releases/> bezogen werden.
- GDAL, Version 1.8 oder höher (Version 1.9 oder höher wird dringend empfohlen, da niedrigere Versionen in manchen Bereichen nicht gut funktionieren bzw. zu verschiedenartigen Verhalten führen können). GDAL wird zur Rasterunterstützung und zur Installation mittels `CREATE EXTENSION postgis` benötigt. Dies ist insbesondere für PostgreSQL 9.1+ in hohem Maße empfohlen. <http://trac.osgeo.org/gdal/wiki/DownloadSource>.

Optionale Systemanforderungen

- GDAL (scheinbar optional) nur wenn man auf Rasterunterstützung und auf `CREATE EXTENSION postgis` verzichten will. Beachte, das andere Extensions PostGIS als Extension benötigen und somit nicht installiert werden können, solange PostGIS nicht als Extension installiert ist. Daher wird die Kompilation mit GDAL stark empfohlen.
Stelle sicher, das die benötigten Treiber nach Section 2.1 aktiviert sind.
- GTK (benötigt GTK+2.0, 2.8+) um den "shp2pgsql-gui shape file loader" zu kompilieren. <http://www.gtk.org/>.
- SFCGAL, Version 1.1 (oder höher) bietet zusätzliche, hoch entwickelte 2D und 3D Analysefunktionen für PostGIS cf Section 8.10. Ermöglicht auch die Anwendung von SFCGAL anstatt von GEOS für einige 2D Funktionen, die von beiden Backends unterstützt werden (wie `ST_Intersection` oder `ST_Area`). Eine PostgreSQL Konfigurationsvariable `postgis.backend` ermöglicht es den Endanwendern zwischen dem Backend zu wählen, falls SFCGAL installiert ist (Standardwert ist GEOS). Anmerkung: SFCGAL 1.2 benötigt mindestens CGAL 4.3 und Boost 1.54 (cf: <http://oslandia.github.io/SFCGAL/-installation.html>) <https://github.com/Oslandia/SFCGAL>.
- Um den Chapter 12 zu kompilieren wird <http://www.pcre.org> benötigt (ist normalerweise auf Unix-Systemen bereits vorinstalliert). Regex: `:Assemble perl CPAN package` ist nur für eine Neukodierung der Daten in `parseaddress-stcities.h` erforderlich. Chapter 12 wird selbsttätig erzeugt, wenn eine PCRE Bibliothek gefunden wird, oder ein gültiger `--with-pcre-dir=/path/to/pcre` im Konfigurationsschritt angegeben wird.
- To enable `ST_AsMVT` protobuf-c library (for usage) and the `protoc-c` compiler (for building) are required. Also, `pkg-config` is required to verify the correct minimum version of protobuf-c. See [protobuf-c](#).

- CUnit (CUnit). Wird für Regressionstest benötigt. <http://cunit.sourceforge.net/>
- DocBook (xsltproc) ist für die Kompilation der Dokumentation notwendig. Docbook steht unter <http://www.docbook.org/> zur Verfügung.
- DBLatex (dblatex) ist zur Kompilation der Dokumentation im PDF-Format nötig. DBLatex liegt unter <http://dblatex.sourceforge.net/> vor.
- ImageMagick (convert) wird zur Erzeugung von Bildern für die Dokumentation benötigt. ImageMagick kann von <http://www.imagemagick.org/> bezogen werden.

2.3 Nutzung des Quellcodes

Das PostGIS Quellarchiv kann von der Download Webseite <http://download.osgeo.org/postgis/source/postgis-2.4.0.tar.gz> bezogen werden.

```
wget http://download.osgeo.org/postgis/source/postgis-2.4.0.tar.gz
tar -xvzf postgis-2.4.0.tar.gz
```

Dadurch wird das Verzeichnis `postgis-2.4.0` im aktuellen Arbeitsverzeichnis erzeugt.

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

```
svn checkout http://svn.osgeo.org/postgis/trunk/ postgis-2.4.0
```

Um die Installation fortzusetzen ist in das neu erstellte Verzeichnis `postgis-2.4.0` zu wechseln.

2.4 Kompilation und Installation mittels Quellcode: Detaillierte Beschreibung

Note

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



Dieser Abschnitt enthält die allgemeinen Installationsanweisungen. Für das Kompilieren unter Windows oder unter einem anderen Betriebssystem findet sich zusätzliche, detailliertere Hilfe unter [PostGIS User contributed compile guides](#) und [PostGIS Dev Wiki](#).

Vorkompilierte Pakete für unterschiedliche Betriebssysteme sind unter [PostGIS Pre-built Packages](#) aufgelistet.

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

PostGIS ist eine Erweiterung des PostgreSQL Servers. Daher *benötigt* PostGIS 2.4.0 vollen Zugriff auf die PostgreSQL server headers für die Kompilation. PostGIS kann in Abhängigkeit von PostgreSQL Versionen 9.3 oder höher kompiliert werden. Niedrigere Versionen von PostgreSQL werden *nicht* unterstützt.

Beziehen Sie sich auf die PostgreSQL Installationshilfe, falls Sie PostgreSQL noch nicht installiert haben. <http://www.postgresql.org>

Note

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



```
LD_FLAGS=-lstdc++ ./configure [IHRE OPTIONEN]
```

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 beschreiben die Konfiguration und Kompilation des PostGIS Quellcodes. Sie gelten für Linux Anwender und funktionieren nicht für Windows oder Mac.

2.4.1 Konfiguration

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

./configure

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

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

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



Caution

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

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

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

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

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

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

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

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

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

- with-gui** Kompilieren Sie die Datenimport-GUI (benötigt GTK+2.0). Dies erzeugt die graphische Schnittstelle "shp2pgsql-gui" für shp2pgsql.
- with-raster** Kompiliert mit Rasterunterstützung. Dies erzeugt die Bibliothek rtpostgis-2.4.0 und die Datei "rtpostgis.sql". Dies wird bei der endgültigen Release nicht mehr nötig sein, da die standardmäßige Unterstützung von Rastern geplant ist
- without-topology** Ausschalten der Topologie Unterstützung. Es existiert keine entsprechende Bibliothek, da sich die gesamte benötigte Logik in der postgis-2.4.0 Bibliothek befindet.
- with-gettext=no** By default PostGIS will try to detect gettext support and compile with it, however if you run into incompatibility issues that cause breakage of loader, you can disable it entirely with this command. Refer to ticket <http://trac.osgeo.org/postgis/ticket/748> for an example issue solved by configuring with this. NOTE: that you aren't missing much by turning this off. This is used for international help/label support for the GUI loader which is not yet documented and still experimental.
- with-sfcgal=PATH** Ohne diesen Switch wird PostGIS ohne sfcgal Unterstützung installiert. PATH ist ein optionaler Parameter, welcher einen alternativen Pfad zu sfcgal-config angibt.

Note


Falls Sie PostGIS vom SVN [Repository](#) heruntergeladen haben, so ist die Ausführung des Skripts der erste Schritt.

./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 das Sie **./autogen.sh** ausführen, da **configure** bereits erzeugt wurde.

2.4.2 Build-Prozess

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

make

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

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

make comments

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

Einige bereits Vorgefertigte können von [PostGIS / PostgreSQL Study Guides](#) als HTML oder PDF heruntergeladen werden

make cheatsheets

2.4.3 Build-Prozess für PostGIS Extensions und Softwareverteilung

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

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

make comments

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

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

```
cd extensions
cd postgis
make clean
make
make install
cd ..
cd postgis_topology
make clean
make
make install
cd ..
cd postgis_sfcgal
make clean
make
make install

cd ..
cd address_standardizer
make clean
make
make install
make installcheck

cd ..
cd postgis_tiger_geocoder
make clean
make
make install
make installcheck
```

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.

- These are the control files that denote information such as the version of the extension to install if not specified. `postgis.control`, `postgis_topology.control`.
- All the files in the /sql folder of each extension. Note that these need to be copied to the root of the PostgreSQL share/extension folder `extensions/postgis/sql/*.sql`, `extensions/postgis_topology/sql/*.sql`

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

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

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

name	default_version	installed_version
address_standardizer	2.4.0	2.4.0
address_standardizer_data_us	2.4.0	2.4.0
postgis	2.4.0	2.4.0
postgis_sfcgal	2.4.0	

```

postgis_tiger_geocoder      | 2.4.0      | 2.4.0
postgis_topology           | 2.4.0      |
(6 rows)

```

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

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

```

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

```

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

```

\connect mygisdb
\x
\dx postgis*

```

```

List of installed extensions
-[ RECORD 1 ]-----
Name          | postgis
Version       | 2.4.0
Schema        | public
Description   | PostGIS geometry, geography, and raster spat..
-[ RECORD 2 ]-----
Name          | postgis_tiger_geocoder
Version       | 2.4.0
Schema        | tiger
Description   | PostGIS tiger geocoder and reverse geocoder
-[ RECORD 3 ]-----
Name          | postgis_topology
Version       | 2.4.0
Schema        | topology
Description   | PostGIS topology spatial types and functions

```

Warning



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

If you installed 2.4.0, without using our wonderful extension system, you can change it to be extension based by first upgrading to the latest micro version running the upgrade scripts: `postgis_upgrade_22_minor.sql`, `raster_upgrade_22_minor.sql`, `topology_upgrade_22_minor.sql`.

Wenn Sie PostGIS ohne Rasterunterstützung installiert haben, so müssen Sie zuerst die Rasterunterstützung mittels `rtpostgis.sql` installieren

Then you can run the below commands to package the functions in their respective extension.

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

2.4.4 Softwaretest

Wenn Sie die Kompilation von PostGIS überprüfen wollen:

make check

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



Note

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



Caution

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

Wenn der Test erfolgreich war, sollte die Ausgabe etwa so aussehen:

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

```
Suite: computational_geometry
Test: test_lw_segment_side ...passed
Test: test_lw_segment_intersects ...passed
Test: test_lwline_crossing_short_lines ...passed
Test: test_lwline_crossing_long_lines ...passed
Test: test_lwline_crossing_bugs ...passed
Test: test_lwpoint_set_ordinate ...passed
Test: test_lwpoint_get_ordinate ...passed
Test: test_point_interpolate ...passed
Test: test_lwline_clip ...passed
Test: test_lwline_clip_big ...passed
Test: test_lwmline_clip ...passed
Test: test_geohash_point ...passed
Test: test_geohash_precision ...passed
Test: test_geohash ...passed
Test: test_geohash_point_as_int ...passed
Test: test_isclosed ...passed
Test: test_lwgeom_simplify ...passed
```

```
Suite: buildarea
  Test: buildarea1 ...passed
  Test: buildarea2 ...passed
  Test: buildarea3 ...passed
  Test: buildarea4 ...passed
  Test: buildarea4b ...passed
  Test: buildarea5 ...passed
  Test: buildarea6 ...passed
  Test: buildarea7 ...passed
Suite: geometry_clean
  Test: test_lwgeom_make_valid ...passed
Suite: clip_by_rectangle
  Test: test_lwgeom_clip_by_rect ...passed
Suite: force_sfs
  Test: test_sfs_11 ...passed
  Test: test_sfs_12 ...passed
  Test: test_sqlmm ...passed
Suite: geodetic
  Test: test_sphere_direction ...passed
  Test: test_sphere_project ...passed
  Test: test_lwgeom_area_sphere ...passed
  Test: test_signum ...passed
  Test: test_gbox_from_spherical_coordinates ...passed
  Test: test_gserialized_get_gbox_geocentric ...passed
  Test: test_clairaut ...passed
  Test: test_edge_intersection ...passed
  Test: test_edge_intersects ...passed
  Test: test_edge_distance_to_point ...passed
  Test: test_edge_distance_to_edge ...passed
  Test: test_lwgeom_distance_sphere ...passed
  Test: test_lwgeom_check_geodetic ...passed
  Test: test_gserialized_from_lwgeom ...passed
  Test: test_spheroid_distance ...passed
  Test: test_spheroid_area ...passed
  Test: test_lwpoly_covers_point2d ...passed
  Test: test_gbox_utils ...passed
  Test: test_vector_angle ...passed
  Test: test_vector_rotate ...passed
  Test: test_lwgeom_segmentize_sphere ...passed
  Test: test_ptarray_contains_point_sphere ...passed
  Test: test_ptarray_contains_point_sphere_iowa ...passed
Suite: GEOS
  Test: test_geos_noop ...passed
  Test: test_geos_subdivide ...passed
  Test: test_geos_linemerge ...passed
Suite: Clustering
  Test: basic_test ...passed
  Test: nonsequential_test ...passed
  Test: basic_distance_test ...passed
  Test: single_input_test ...passed
  Test: empty_inputs_test ...passed
Suite: Clustering Union-Find
  Test: test_unionfind_create ...passed
  Test: test_unionfind_union ...passed
  Test: test_unionfind_ordered_by_cluster ...passed
Suite: homogenize
  Test: test_coll_point ...passed
  Test: test_coll_line ...passed
  Test: test_coll_poly ...passed
  Test: test_coll_coll ...passed
  Test: test_geom ...passed
  Test: test_coll_curve ...passed
```



```
Suite: encoded_polyline_input
  Test: in_encoded_polyline_test_geoms ...passed
  Test: in_encoded_polyline_test_precision ...passed
Suite: geojson_input
  Test: in_geojson_test_srid ...passed
  Test: in_geojson_test_bbox ...passed
  Test: in_geojson_test_geoms ...passed
Suite: twkb_input
  Test: test_twkb_in_point ...passed
  Test: test_twkb_in_linestring ...passed
  Test: test_twkb_in_polygon ...passed
  Test: test_twkb_in_multipoint ...passed
  Test: test_twkb_in_multilinestring ...passed
  Test: test_twkb_in_multipolygon ...passed
  Test: test_twkb_in_collection ...passed
  Test: test_twkb_in_precision ...passed
Suite: serialization/deserialization
  Test: test_typmod_macros ...passed
  Test: test_flags_macros ...passed
  Test: test_serialized_srid ...passed
  Test: test_gserialized_from_lwgeom_size ...passed
  Test: test_gbox_serialized_size ...passed
  Test: test_lwgeom_from_gserialized ...passed
  Test: test_lwgeom_count_vertices ...passed
  Test: test_on_gser_lwgeom_count_vertices ...passed
  Test: test_geometry_type_from_string ...passed
  Test: test_lwcollection_extract ...passed
  Test: test_lwgeom_free ...passed
  Test: test_lwgeom_flip_coordinates ...passed
  Test: test_f2d ...passed
  Test: test_lwgeom_clone ...passed
  Test: test_lwgeom_force_clockwise ...passed
  Test: test_lwgeom_calculate_gbox ...passed
  Test: test_lwgeom_is_empty ...passed
  Test: test_lwgeom_same ...passed
  Test: test_lwline_from_lwmpoint ...passed
  Test: test_lwgeom_as_curve ...passed
  Test: test_lwgeom_scale ...passed
  Test: test_gserialized_is_empty ...passed
  Test: test_gbox_same_2d ...passed
Suite: measures
  Test: test_mindistance2d_tolerance ...passed
  Test: test_rect_tree_contains_point ...passed
  Test: test_rect_tree_intersects_tree ...passed
  Test: test_lwgeom_segmentize2d ...passed
  Test: test_lwgeom_locate_along ...passed
  Test: test_lw_dist2d_pt_arc ...passed
  Test: test_lw_dist2d_seg_arc ...passed
  Test: test_lw_dist2d_arc_arc ...passed
  Test: test_lw_arc_length ...passed
  Test: test_lw_dist2d_pt_ptarrayarc ...passed
  Test: test_lw_dist2d_ptarray_ptarrayarc ...passed
  Test: test_lwgeom_tcpa ...passed
  Test: test_lwgeom_is_trajectory ...passed
Suite: effectivearea
  Test: do_test_lwgeom_effectivearea_lines ...passed
  Test: do_test_lwgeom_effectivearea_polys ...passed
Suite: miscellaneous
  Test: test_misc_force_2d ...passed
  Test: test_misc_simplify ...passed
  Test: test_misc_count_vertices ...passed
  Test: test_misc_area ...passed
```

```
Test: test_misc_wkb ...passed
Test: test_grid ...passed
Suite: noding
Test: test_lwgeom_node ...passed
Suite: encoded_polyline_output
Test: out_encoded_polyline_test_geoms ...passed
Test: out_encoded_polyline_test_srid ...passed
Test: out_encoded_polyline_test_precision ...passed
Suite: geojson_output
Test: out_geojson_test_precision ...passed
Test: out_geojson_test_dims ...passed
Test: out_geojson_test_srid ...passed
Test: out_geojson_test_bbox ...passed
Test: out_geojson_test_geoms ...passed
Suite: gml_output
Test: out_gml_test_precision ...passed
Test: out_gml_test_srid ...passed
Test: out_gml_test_dims ...passed
Test: out_gml_test_geodetic ...passed
Test: out_gml_test_geoms ...passed
Test: out_gml_test_geoms_prefix ...passed
Test: out_gml_test_geoms_nodims ...passed
Test: out_gml2_extent ...passed
Test: out_gml3_extent ...passed
Suite: kml_output
Test: out_kml_test_precision ...passed
Test: out_kml_test_dims ...passed
Test: out_kml_test_geoms ...passed
Test: out_kml_test_prefix ...passed
Suite: svg_output
Test: out_svg_test_precision ...passed
Test: out_svg_test_dims ...passed
Test: out_svg_test_relative ...passed
Test: out_svg_test_geoms ...passed
Test: out_svg_test_srid ...passed
Suite: x3d_output
Test: out_x3d3_test_precision ...passed
Test: out_x3d3_test_geoms ...passed
Test: out_x3d3_test_option ...passed
Suite: ptarray
Test: test_ptarray_append_point ...passed
Test: test_ptarray_append_ptarray ...passed
Test: test_ptarray_locate_point ...passed
Test: test_ptarray_isccw ...passed
Test: test_ptarray_signed_area ...passed
Test: test_ptarray_unstroke ...passed
Test: test_ptarray_insert_point ...passed
Test: test_ptarray_contains_point ...passed
Test: test_ptarrayarc_contains_point ...passed
Test: test_ptarray_scale ...passed
Suite: printing
Test: test_lwprint_default_format ...passed
Test: test_lwprint_format_orders ...passed
Test: test_lwprint_optional_format ...passed
Test: test_lwprint_oddball_formats ...passed
Test: test_lwprint_bad_formats ...passed
Suite: SFCGAL
Test: test_sfcgal_noop ...passed
Suite: split
Test: test_lwline_split_by_point_to ...passed
Test: test_lwgeom_split ...passed
Suite: stringbuffer
```

```
Test: test_stringbuffer_append ...passed
Test: test_stringbuffer_aprintf ...passed
Suite: surface
Test: triangle_parse ...passed
Test: tin_parse ...passed
Test: polyhedralsurface_parse ...passed
Test: surface_dimension ...passed
Suite: Internal Spatial Trees
Test: test_tree_circ_create ...passed
Test: test_tree_circ_pip ...passed
Test: test_tree_circ_pip2 ...passed
Test: test_tree_circ_distance ...passed
Test: test_tree_circ_distance_threshold ...passed
Suite: triangulate
Test: test_lwgeom_delaunay_triangulation ...passed
Suite: twkb_output
Test: test_twkb_out_point ...passed
Test: test_twkb_out_linestring ...passed
Test: test_twkb_out_polygon ...passed
Test: test_twkb_out_multipoint ...passed
Test: test_twkb_out_multilinestring ...passed
Test: test_twkb_out_multipolygon ...passed
Test: test_twkb_out_collection ...passed
Test: test_twkb_out_idlist ...passed
Suite: varint
Test: test_zigzag ...passed
Test: test_varint ...passed
Test: test_varint_roundtrip ...passed
Suite: wkb_input
Test: test_wkb_in_point ...passed
Test: test_wkb_in_linestring ...passed
Test: test_wkb_in_polygon ...passed
Test: test_wkb_in_multipoint ...passed
Test: test_wkb_in_multilinestring ...passed
Test: test_wkb_in_multipolygon ...passed
Test: test_wkb_in_collection ...passed
Test: test_wkb_in_circularstring ...passed
Test: test_wkb_in_compoundcurve ...passed
Test: test_wkb_in_curvpolygon ...passed
Test: test_wkb_in_multicurve ...passed
Test: test_wkb_in_multisurface ...passed
Test: test_wkb_in_malformed ...passed
Suite: wkb_output
Test: test_wkb_out_point ...passed
Test: test_wkb_out_linestring ...passed
Test: test_wkb_out_polygon ...passed
Test: test_wkb_out_multipoint ...passed
Test: test_wkb_out_multilinestring ...passed
Test: test_wkb_out_multipolygon ...passed
Test: test_wkb_out_collection ...passed
Test: test_wkb_out_circularstring ...passed
Test: test_wkb_out_compoundcurve ...passed
Test: test_wkb_out_curvpolygon ...passed
Test: test_wkb_out_multicurve ...passed
Test: test_wkb_out_multisurface ...passed
Test: test_wkb_out_polyhedralsurface ...passed
Suite: wkt_input
Test: test_wkt_in_point ...passed
Test: test_wkt_in_linestring ...passed
Test: test_wkt_in_polygon ...passed
Test: test_wkt_in_multipoint ...passed
Test: test_wkt_in_multilinestring ...passed
```

```

Test: test_wkt_in_multipolygon ...passed
Test: test_wkt_in_collection ...passed
Test: test_wkt_in_circularstring ...passed
Test: test_wkt_in_compoundcurve ...passed
Test: test_wkt_in_curvpolygon ...passed
Test: test_wkt_in_multicurve ...passed
Test: test_wkt_in_multisurface ...passed
Test: test_wkt_in_tin ...passed
Test: test_wkt_in_polyhedralsurface ...passed
Test: test_wkt_in_errlocation ...passed
Suite: wkt_output
Test: test_wkt_out_point ...passed
Test: test_wkt_out_linestring ...passed
Test: test_wkt_out_polygon ...passed
Test: test_wkt_out_multipoint ...passed
Test: test_wkt_out_multilinestring ...passed
Test: test_wkt_out_multipolygon ...passed
Test: test_wkt_out_collection ...passed
Test: test_wkt_out_circularstring ...passed
Test: test_wkt_out_compoundcurve ...passed
Test: test_wkt_out_curvpolygon ...passed
Test: test_wkt_out_multicurve ...passed
Test: test_wkt_out_multisurface ...passed

Run Summary:
  Type   Total    Ran  Passed  Failed  Inactive
  suites    38     38    n/a      0        0
  tests   251    251    251      0        0
  asserts 2468   2468   2468      0        n/a

Elapsed time = 0.298 seconds

Creating database 'postgis_reg'
Loading PostGIS into 'postgis_reg'
  /projects/postgis/branches/2.2/regress/00-regress-install/share/contrib/postgis/postgis. ←
  sql
  /projects/postgis/branches/2.2/regress/00-regress-install/share/contrib/postgis/ ←
  postgis_comments.sql
Loading SFCGAL into 'postgis_reg'
  /projects/postgis/branches/2.2/regress/00-regress-install/share/contrib/postgis/sfcgal. ←
  sql
  /projects/postgis/branches/2.2/regress/00-regress-install/share/contrib/postgis/ ←
  sfcgal_comments.sql
PostgreSQL 9.4.4, compiled by Visual C++ build 1800, 32-bit
Postgis 2.2.0dev - r13980 - 2015-08-23 06:13:07
scripts 2.2.0dev r13980
GEOS: 3.5.0-CAPI-1.9.0 r4088
PROJ: Rel. 4.9.1, 04 March 2015
SFCGAL: 1.1.0

Running tests

loader/Point ..... ok
loader/PointM ..... ok
loader/PointZ ..... ok
loader/MultiPoint ..... ok
loader/MultiPointM ..... ok
loader/MultiPointZ ..... ok
loader/Arc ..... ok
loader/ArcM ..... ok
loader/ArcZ ..... ok
loader/Polygon ..... ok
loader/PolygonM ..... ok

```

```
loader/PolygonZ ..... ok
loader/TSTPolygon ..... ok
loader/TSIPolygon ..... ok
loader/TSTIPolygon ..... ok
loader/PointWithSchema ..... ok
loader/NoTransPoint ..... ok
loader/NotReallyMultiPoint ..... ok
loader/MultiToSinglePoint ..... ok
loader/ReprojectPts ..... ok
loader/ReprojectPtsGeog ..... ok
loader/Latin1 .... ok
loader/Latin1-implicit .... ok
loader/mfile .... ok
dumper/literalsrid ..... ok
dumper/realtable ..... ok
affine .. ok
bestsrid .. ok
binary .. ok
boundary .. ok
cluster .. ok
concave_hull .. ok
ctors .. ok
dump .. ok
dumppoints .. ok
empty .. ok
forcecurve .. ok
geography .. ok
in_geohash .. ok
in_gml .. ok
in_kml .. ok
in_encodedpolyline .. ok
iscollection .. ok
legacy .. ok
long_xact .. ok
lwgeom_regress .. ok
measures .. ok
operators .. ok
out_geometry .. ok
out_geography .. ok
polygonize .. ok
polyhedralsurface .. ok
postgis_type_name .. ok
regress .. ok
regress_bdpoly .. ok
regress_index .. ok
regress_index_nulls .. ok
regress_management .. ok
regress_selectivity .. ok
regress_lrs .. ok
regress_ogc .. ok
regress_ogc_cover .. ok
regress_ogc_prep .. ok
regress_proj .. ok
relate .. ok
remove_repeated_points .. ok
removepoint .. ok
setpoint .. ok
simplify .. ok
simplifyvw .. ok
size .. ok
snaptogrid .. ok
split .. ok
```

```
sql-mm-serialize .. ok
sql-mm-circularstring .. ok
sql-mm-compoundcurve .. ok
sql-mm-curvepoly .. ok
sql-mm-general .. ok
sql-mm-multicurve .. ok
sql-mm-multisurface .. ok
swapordinates .. ok
summary .. ok
temporal .. ok
tickets .. ok
twkb .. ok
typmod .. ok
wkb .. ok
wkt .. ok
wmsservers .. ok
knn .. ok
hausdorff .. ok
regress_buffer_params .. ok
offsetcurve .. ok
relatemark .. ok
isvaliddetail .. ok
sharedpaths .. ok
snap .. ok
node .. ok
unaryunion .. ok
clean .. ok
relate_bnr .. ok
delaunaytriangles .. ok
clipbybox2d .. ok
subdivide .. ok
in_geojson .. ok
regress_sfcgal .. ok
sfcgal/empty .. ok
sfcgal/geography .. ok
sfcgal/legacy .. ok
sfcgal/measures .. ok
sfcgal/regress_ogc_prep .. ok
sfcgal/regress_ogc .. ok
sfcgal/regress .. ok
sfcgal/tickets .. ok
sfcgal/concave_hull .. ok
sfcgal/wmsservers .. ok
sfcgal/approximate medial axis .. ok
uninstall . /projects/postgis/branches/2.2/regress/00-regress-install/share/contrib/ ↔
    postgis/uninstall_sfcgal.sql
    /projects/postgis/branches/2.2/regress/00-regress-install/share/contrib/postgis/ ↔
    uninstall_postgis.sql
. ok (4336)

Run tests: 118
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/

Suite: Shapefile Loader File shp2pgsql Test
Test: test_ShpLoaderCreate() ...passed
Test: test_ShpLoaderDestroy() ...passed
```

```
Suite: Shapefile Loader File pgsq12shp Test
Test: test_ShpDumperCreate() ...passed
Test: test_ShpDumperDestroy() ...passed
```

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

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

Für den `address_standardizer`:

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

Die Ausgabe sollte folgendermaßen aussehen:

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

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

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

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

Die Ausgabe sollte folgendermaßen aussehen:

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

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

2.4.5 Installation

Um PostGIS zu installieren geben Sie bitte folgendes ein

make install

Dies kopiert die PostGIS Installationsdateien in ihr entsprechenden Unterverzeichnisse, welches durch den **--prefix** Konfigurationsparameter 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, dann können Sie die SQL-Dateien folgendermaßen installieren:

make comments-install



Note

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

2.5 Eine Geodatenbank über EXTENSIONS anlegen

Wenn Sie PostgreSQL 9.1+ verwenden und das Extensions/PostGIS-Modul installiert haben, können Sie Geodatenbanken auf neue Art und Weise erstellen.

createdb [yourdatabase]

Die Core-Extension installiert PostGIS-Geometrie, -Geographie, -Raster, die `spatial_ref_sys` Tabelle und alle Funktionen und Kommentare mit einem einfachen

```
CREATE EXTENSION postgis;
```

Befehl.

psql -d [yourdatabase] -c "CREATE EXTENSION postgis;"

Die topologische Funktionalität ist in einer eigenen Extension paketierrt und kann mit folgendem Befehl installiert werden:

psql -d [yourdatabase] -c "CREATE EXTENSION postgis_topology;"

Falls Sie die Sicherung einer Vorgängerversion in die neue Datenbank einspielen wollen, führen Sie bitte folgendes aus:

psql -d [yourdatabase] -f legacy.sql

Um die veralteten Funktionen loszuwerden, können Sie anschließend an die Wiederherstellung und Aufräumarbeiten `uninstall_legacy.sql` ausführen.

2.6 Erstellung einer Geodatenbank ohne Extensions



Note

Dies ist üblicherweise nur dann notwendig, wenn Sie PostGIS ohne die Rasterunterstützung installieren wollen. Da die Rasterfunktionen ein Teil der PostGIS Extension sind, wird die Unterstützung von Extensions nicht aktiviert, wenn PostGIS ohne Rasterunterstützung installiert wird.

Der erste Schritt zur Erstellung einer PostGIS-Datenbank ist das Anlegen einer einfachen PostgreSQL Datenbank.

createdb [yourdatabase]

Viele der PostGIS Funktionen sind in der prozeduralen Sprache PL/pgSQL geschrieben. Daher ist der nächste Schritt zur Erstellung einer PostGIS Datenbank die Aktivierung von PL/pgSQL. Dies wird durch den unten angeführten Befehl erreicht. Ab PostgreSQL 8.4 ist PL/pgSQL üblicherweise bereits installiert.

createlang plpgsql [yourdatabase]

Nun erstellen Sie bitte die Definitionen der PostGIS-Objekte und -Funktionen in Ihre Datenbank, indem Sie die Definitionen mit der Datei `postgis.sql` laden (diese befindet sich in dem beim Konfigurationsschritt festgelegten Verzeichnis `[prefix]/share/contrib`).

psql -d [yourdatabase] -f postgis.sql

Für einen vollständigen Satz an EPSG Koordinatensystemen, können Sie die Definitionen auch über die Datei `spatial_ref_sys.sql` laden und die `spatial_ref_sys` Tabelle auf diese Weise befüllen. Diese Tabelle ermöglicht die Ausführung von `ST_Transform()` auf die Geometrien.

psql -d [yourdatabase] -f spatial_ref_sys.sql

Falls Sie Kommentare zu den PostGIS-Funktionen hinzufügen wollen, so ist der letzte Schritt das Laden von `postgis_comments.sql` in Ihre Geodatenbank. Die Kommentare können mit dem einfachen Aufruf von `\dd [function_name]` in der **psql** Konsole angezeigt werden.

psql -d [yourdatabase] -f postgis_comments.sql

Installation der Rasterunterstützung

psql -d [yourdatabase] -f rtpostgis.sql

Die Installation der Kommentare zur Rasterunterstützung stellt eine schnelle Hilfe für jede Rasterfunktion bereit. Diese kann dann über `psql`, `PgAdmin` oder andere PostgreSQL Werkzeuge die Funktionskommentare anzeigen können, aufgerufen werden.

psql -d [yourdatabase] -f raster_comments.sql

Installation der Topologieunterstützung

psql -d [yourdatabase] -f topology/topology.sql

Die Installation der Kommentare zur Topologie-Unterstützung stellt eine schnelle Hilfe für jede topologische Funktion und jeden topologischen Datentyp bereit. Diese kann dann über `psql`, `PgAdmin` oder andere PostgreSQL Werkzeuge die Funktionskommentare anzeigen können, aufgerufen werden.

psql -d [yourdatabase] -f topology/topology_comments.sql

Falls Sie die Sicherung einer Vorgängerversion in die neue Datenbank einspielen wollen, führen Sie bitte folgendes aus:

psql -d [yourdatabase] -f legacy.sql



Note

There is an alternative `legacy_minimal.sql` you can run instead which will install barebones needed to recover tables and work with apps like MapServer and GeoServer. If you have views that use things like distance / length etc, you'll need the full blown `legacy.sql`

Um die veralteten Funktionen loszuwerden, können Sie anschließend an die Wiederherstellung und Aufräumarbeiten `uninstall_legacy.sql` ausführen.

2.7 Installation und Verwendung des Adressennormierers

The `address_standardizer` extension used to be a separate package that required separate download. From PostGIS 2.2 on, it is now bundled in. For more information about the `address_standardize`, what it does, and how to configure it for your needs, refer to Chapter 12.

This standardizer can be used in conjunction with the PostGIS packaged tiger geocoder extension as a replacement for the **Normalize_Address** discussed. To use as replacement refer to Section 2.8.3. You can also use it as a building block for your own geocoder or use it to standardize your addresses for easier compare of addresses.

The address standardizer relies on PCRE which is usually already installed on many Nix systems, but you can download the latest at: <http://www.pcre.org>. If during Section 2.4.1, PCRE is found, then the address standardizer extension will automatically be built. If you have a custom pcre install you want to use instead, pass to configure `--with-pcredir=/path/to/pcre` where `/path/to/pcre` is the root folder for your pcre include and lib directories.

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

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

```
CREATE EXTENSION address_standardizer;
```

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

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

Die Ausgabe sollte wie folgt sein:

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

2.7.1 Installation von Regexp::Assemble

Perl Regexp::Assemble is no longer needed for compiling `address_standardizer` extension since the files it generates are part of the source tree. However if you need to edit the `usps-st-city-orig.txt` or `usps-st-city-orig.txt usps-st-city-adc` tx, you need to rebuild `parseaddress-stcities.h` which does require Regexp::Assemble.

```
cpan Regexp::Assemble
```

oder wenn Sie sich auf einer Ubuntu / Debian Distribution arbeiten, müssen Sie möglicherweise folgendes ausführen:

```
sudo perl -MCPAN -e "install Regexp::Assemble"
```

2.8 Installation, Aktualisierung des Tiger Geokodierers und Daten laden

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

2.8.1 Aktivierung des Tiger Geokodierer in Ihrer PostGIS Datenbank: Verwendung von Extension

Falls Sie PostgreSQL 9.1+ und PostGIS 2.1+ verwenden, können Sie Vorteil aus dem Extension-Modell ziehen, um den Tiger Geokodierer zu installieren. Um dies zu tun:

1. Besorgen Sie sich zuerst die Binärdateien für PostGIS 2.1+ oder kompilieren und installieren Sie diese wie üblich. Dies sollte alle notwendigen Extension-Dateien als auch den Tiger Geokodierer installieren.
2. Connect to your database via psql or pgAdmin or some other tool and run the following SQL commands. Note that if you are installing in a database that already has postgis, you don't need to do the first step. If you have fuzzystmatch extension already installed, you don't need to do the second step either.

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

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

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

If you made custom entries or changes to `tiger.loader_platform` and `tiger.loader_variables` you may need to update these.

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

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

Dies sollte folgendes ausgeben:

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

4. Create a new record in `tiger.loader_platform` table with the paths of your executables and server. So for example to create a profile called `debbie` that follows `sh` convention. You would do:

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

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

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

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

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

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

8. Run the generated nation load commandline scripts.

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

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

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

```
count
-----
  3233
(1 row)
```

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

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

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

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

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

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



Warning

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

12.

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

13. Die vorher erzeugten, befehlszeilenorientierten Skripts ausführen.

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

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

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

2.8.1.1 Umwandlung einer normalen Installation des Tiger-Geokodierers in das Extension Modell

Falls Sie den Tiger Geokodierer ohne Extension Modell installiert haben, können Sie wie folgt auf das Extension-Modell wechseln:

1. Für ein Upgrade ohne Extension-Modell, folgen Sie bitte den Anweisungen unter [Section 2.8.5](#).
2. Verbinden Sie sich über "psql" mit Ihrer Datenbank und führen Sie folgenden Befehl aus:

```
CREATE EXTENSION postgis_tiger_geocoder FROM unpackaged;
```

2.8.2 Den Tiger Geokodierer in Ihrer PostGIS Datenbank aktivieren: Ohne die Verwendung von Extension

Zuerst installieren Sie PostGIS entsprechend den vorherigen Anweisungen.

If you don't have an extras folder, download <http://download.osgeo.org/postgis/source/postgis-2.4.0.tar.gz>

```
tar xvfz postgis-2.4.0.tar.gz
```

```
cd postgis-2.4.0/extras/tiger_geocoder
```

Edit the `tiger_loader_2015.sql` (or latest loader file you find, unless you want to load different year) to the paths of your executables server etc or alternatively you can update the `loader_platform` table once installed. If you don't edit this file or the `loader_platform` table, it will just contain common case locations of items and you'll have to edit the generated script after the fact when you run the [Loader_Generate_Nation_Script](#) and [Loader_Generate_Script](#) SQL functions.

If you are installing Tiger geocoder for the first time edit either the `create_geocode.bat` script If you are on windows or the `create_geocode.sh` if you are on Linux/Unix/Mac OSX with your PostgreSQL specific settings and run the corresponding script from the commandline.

Verify that you now have a `tiger` schema in your database and that it is part of your database `search_path`. If it is not, add it with a command something along the line of:

```
ALTER DATABASE geocoder SET search_path=public, tiger;
```

The normalizing address functionality works more or less without any data except for tricky addresses. Run this test and verify things look like this:

```
SELECT pprint_addy(normalize_address('202 East Fremont Street, Las Vegas, Nevada 89101')) ↵
      As pretty_address;
pretty_address
-----
202 E Fremont St, Las Vegas, NV 89101
```

2.8.3 Die Adressennormierer-Extension zusammen mit dem Tiger Geokodierer verwenden

One of the many complaints of folks is the address normalizer function `Normalize_Address` function that normalizes an address for prepping before geocoding. The normalizer is far from perfect and trying to patch its imperfectness takes a vast amount of resources. As such we have integrated with another project that has a much better address standardizer engine. To use this new `address_standardizer`, you compile the extension as described in Section 2.7 and install as an extension in your database.

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

2.8.4 Tiger-Daten laden

The instructions for loading data are available in a more detailed form in the `extras/tiger_geocoder/tiger_2011/README`. This just includes the general steps.

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

Um Daten laden zu können benötigen Sie folgende Werkzeuge:

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

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

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

- Das `shp2pgsql` Kommandozeilenprogramm, welches standardmäßig mit PostGIS mitinstalliert wird.
- `wget`, ein Download-Manager, der üblicherweise auf den meisten UNIX/Linux Systemen vorinstalliert ist.

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

If you are upgrading from `tiger_2010`, you'll need to first generate and run `Drop_Nation_Tables_Generate_Script`. Before you load any state data, you need to load the nation wide data which you do with `Loader_Generate_Nation_Script`. Which will generate a loader script for you. `Loader_Generate_Nation_Script` is a one-time step that should be done for upgrading (from 2010) and for new installs.

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

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

```
SELECT install_missing_indexes();
```

as described in `Install_Missing_Indexes`.

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

2.8.5 Upgrade Ihrer Tiger Geokodierer Installation

If you have Tiger Geocoder packaged with 2.0+ already installed, you can upgrade the functions at any time even from an interim tar ball if there are fixes you badly need. This will only work for Tiger geocoder not installed with extensions.

If you don't have an `extras` folder, download <http://download.osgeo.org/postgis/source/postgis-2.4.0.tar.gz>

```
tar xvfz postgis-2.4.0.tar.gz
```

```
cd postgis-2.4.0/extras/tiger_geocoder/tiger_2011
```

Locate the `upgrade_geocoder.bat` script if you are on windows or the `upgrade_geocoder.sh` if you are on Linux/Unix/Mac OSX. Edit the file to have your postgis database credentials.

If you are upgrading from 2010 or 2011, make sure to unremark out the loader script line so you get the latest script for loading 2012 data.

Dann führen Sie das dazugehörige Skript von der Befehlszeile aus.

Next drop all nation tables and load up the new ones. Generate a drop script with this SQL statement as detailed in [Drop_Nation_Tables_Script](#)

```
SELECT drop_nation_tables_generate_script();
```

Run the generated drop SQL statements.

Generate a nation load script with this SELECT statement as detailed in [Loader_Generate_Nation_Script](#)

Auf Windows:

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

Auf Unix/Linux:

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

Refer to Section [2.8.4](#) for instructions on how to run the generate script. This only needs to be done once.



Note

You can have a mix of 2010/2011 state tables and can upgrade each state separately. Before you upgrade a state to 2011, you first need to drop the 2010 tables for that state using [Drop_State_Tables_Generate_Script](#).

2.9 Erzeugung einer Geodatenbank mit einem Template

Some packaged distributions of PostGIS (in particular the Win32 installers for PostGIS $\geq 1.1.5$) load the PostGIS functions into a template database called `template_postgis`. If the `template_postgis` database exists in your PostgreSQL installation then it is possible for users and/or applications to create spatially-enabled databases using a single command. Note that in both cases, the database user must have been granted the privilege to create new databases.

Von der Shell aus:

```
# createdb -T template_postgis my_spatial_db
```

Mit SQL:

```
postgres=# CREATE DATABASE my_spatial_db TEMPLATE=template_postgis
```

2.10 Upgrading

Das Upgrade einer bestehenden Geodatenbank kann trickreich sein, wenn der Ersatz oder die Einführung von neuen Objektdefinitionen in PostGIS nötig ist.

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

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

Bevor Sie versuchen PostGIS zu aktualisieren, sollten Sie eine Sicherung Ihrer Daten vornehmen. Wenn Sie die Flag `-Fc` von `pg_dump` verwenden, können Sie die Daten über ein HARD UPGRADE immer wieder herstellen.

2.10.1 Soft upgrade

If you installed your database using extensions, you'll need to upgrade using the extension model as well. If you installed using the old sql script way, then you should upgrade using the sql script way. Please refer to the appropriate.

2.10.1.1 Soft Upgrade Pre 9.1+ oder ohne Extensions/Erweiterungen

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

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

After compiling and installing (make install) you should find a `postgis_upgrade.sql` and `rtpostgis_upgrade.sql` in the installation folders. For example `/usr/share/postgresql/9.3/contrib/postgis_upgrade.sql`. Install the `postgis_upgrade.sql`. If you have raster functionality installed, you will also need to install the `/usr/share/postgresql/9.3/contrib/postgis_upgrade.sql`. If you are moving from PostGIS 1.* to PostGIS 2.* or from PostGIS 2.* prior to r7409, you need to do a HARD UPGRADE.

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

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

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

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



Note

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

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

2.10.1.2 Soft Upgrade von PostGIS 9.1+ mittels 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.

```
ALTER EXTENSION postgis UPDATE TO "2.4.0";  
ALTER EXTENSION postgis_topology UPDATE TO "2.4.0";
```

Falls eine Fehlermeldung angezeigt wird, unternehmen Sie bitte etwas ähnliches wie:

```
No migration path defined for ... to 2.4.0
```

Then you'll need to backup your database, create a fresh one as described in Section 2.5 and then restore your backup ontop of this new database.

Falls Sie eine ähnliche Meldung wie folgt bekommen:

```
Version "2.4.0" of extension "postgis" is already installed
```

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


```
ALTER EXTENSION postgis UPDATE TO "2.4.0next";
ALTER EXTENSION postgis_topology UPDATE TO "2.4.0next";
```

**Note**

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

2.10.2 Hard upgrade

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

The dump/reload process is assisted by the `postgis_restore.pl` 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.

Zusätzliche Anweisungen für Windows Benutzer sind unter [Windows Hard upgrade](#) verfügbar.

Die Vorgehensweise ist wie folgt:

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

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

2. Eine frische Installation von PostGIS in einer neuen Datenbank erstellen - wir verweisen auf diese Datenbank mit `newdb`. Anleitungen dazu finden Sie unter [Section 2.6](#) und [Section 2.5](#).

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

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

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

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

In folgenden Fällen können Fehler auftreten:

1. Some of your views or functions make use of deprecated PostGIS objects. In order to fix this you may try loading `legacy.sql` script prior to restore or you'll have to restore to a version of PostGIS which still contains those objects and try a migration again after porting your code. If the `legacy.sql` way works for you, don't forget to fix your code to stop using deprecated functions and drop them loading `uninstall_legacy.sql`.
2. Some custom records of `spatial_ref_sys` in dump file have an invalid SRID value. Valid SRID values are bigger than 0 and smaller than 999000. Values in the 999000.999999 range are reserved for internal use while values > 999999 can't be used at all. All your custom records with invalid SRIDs will be retained, with those > 999999 moved into the reserved range,

but the `spatial_ref_sys` table would lose a check constraint guarding for that invariant to hold and possibly also its primary key (when multiple invalid SRIDS get converted to the same reserved SRID value).

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

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

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

2.11 Übliche Probleme bei der Installation

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

1. Check that you have installed PostgreSQL 9.3 or newer, and that you are compiling against the same version of the PostgreSQL source as the version of PostgreSQL that is running. Mix-ups can occur when your (Linux) distribution has already installed PostgreSQL, or you have otherwise installed PostgreSQL before and forgotten about it. PostGIS will only work with PostgreSQL 9.3 or newer, and strange, unexpected error messages will result if you use an older version. To check the version of PostgreSQL which is running, connect to the database using `psql` and run this query:

```
SELECT version();
```

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

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

```
SELECT postgis_full_version();
```

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

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

2.12 Loader/Dumper

Der Loader und der Dumper werden automatisch, als Teil von PostGIS kompiliert und installiert. Um diese händisch zu kompilieren und zu installieren:

```
# cd postgis-2.4.0/loader  
# make  
# make install
```

Der Loader heisst `shp2pgsql` und konvertiert ESRI Shapefiles in SQL, das in PostGIS/PostgreSQL geladen werden kann. Der Dumper heisst `pgsql2shp` und kopiert PostGIS Tabellen (oder Abfragen) in ESRI Shapefiles. Für eine ausführlichere Beschreibung, siehe die Online Hilfe und das Handbuch.

Chapter 3

Häufige Fragen zu PostGIS

1. *Wo kann ich Lernprogramme, Anleitungen und Seminare für die Arbeit mit PostGIS finden?*

OpenGeo hat ein schrittweises Lernprogramm [Introduction to PostGIS](#). Es beinhaltet sowohl paketierte Daten als auch ein Einführung in das Arbeiten mit der OpenGeo-Suite. Es ist wahrscheinlich das beste Lernprogramm für PostGIS. Außerdem hat BostonGIS ein [PostGIS almost idiot's guide on getting started](#). Dieser ist eher für Windows-Benutzer gedacht.

2. *Meine Anwendungen und Arbeitsflächenwerkzeuge funktionierten mit PostGIS 1.5, nicht jedoch mit PostGIS 2.0. Wie kann ich dies beheben?*

Viele missbilligte Funktionen wurden aus der Codebasis von PostGIS 2.0 entfernt. Dies betraf Anwendungen sowie Werkzeuge von Drittanbietern wie Geoserver, MapServer, QuantumGIS und OpenJump, um einige zu nennen. Es gibt ein paar Wege, dies zu beheben. Für die Anwendungen von Drittanbietern können Sie versuchen, sie auf die neuste Version zu aktualisieren, bei der viele dieser Probleme behoben wurden. Ihren eigenen Code könne Sie so ändern, dass er die entfernten Funktionen nicht mehr benutzt. Die meisten dieser Funktionen sind Pseudonyme von `ST_Union`, `ST_Length` etc. ohne den Präfix "ST_". Als letzten Ausweg können Sie die gesamte `legacy.sql`, oder die Teile die Sie benötigen installieren. Die Datei `legacy.sql` liegt im selben Verzeichnis wie »`postgis.sql`«. Sie können diese Datei nach der Installation von »`postgis.sql`« und »`spatial_ref_sys.sql`« installieren, um alle etwa 200 alten Funktionen zurückzubekommen, die entfernt wurden.

3. *Wenn ich OpenStreetMap-Daten mit »`osm2pgsql`« lade, bekomme ich eine Fehlermeldung: »operator class "gist_geometry_ops" does not exist for access method "gist" Error occurred.« In PostGIS 1.5 klappte das gut.*

In PostGIS 2 wurde die Standardgeometrieoperatorklasse »`gist_geometry_ops`« in »`gist_geometry_ops_2d`« geändert und »`gist_geometry_ops`« wurde vollständig entfernt. Grund ist, dass PostGIS auch räumliche Nd-Indizes für 3D-Unterstützung einführt und der alte Name als verwirrend und fehlerhaft angesehen wurde. Einige ältere Anwendungen, die als Teil des Prozesses Tabellen und Indizes erstellen, referenzieren explizit den Operatorklassennamen. Dies war unnötig, falls Sie den Standard-2D-Index wollten. Falls Sie dies also gut hinbekommen möchten, ändern Sie die Indexerstellung von: SCHLECHT:

```
CREATE INDEX idx_my_table_geom ON my_table USING gist (geom gist_geometry_ops);
```

in GUT:

```
CREATE INDEX idx_my_table_geom ON my_table USING gist (geom);
```

Der einzige Fall, in dem Sie die Operatorklasse angeben MÜSSEN, ist, wenn Sie einen räumlichen 3D-Index wie folgt möchten:

```
CREATE INDEX idx_my_super3d_geom ON my_super3d USING gist (geom gist_geometry_ops_nd);
```

Falls Sie unglücklicherweise kompilierten Code, den Sie nicht mehr ändern können, am Hals haben und bei dem , bei dem altes »`gist_geometry_ops`« hart codiert ist, können Sie die alte Klasse mittels des in PostGIS 2.0.2+ paketierten `legacy_gist.sql` erstellen. Falls Sie jedoch diese Fehlerbehebung verwenden, wird Ihnen dringend empfohlen, zu einem späteren Zeitpunkt den Index zu löschen und ihn ohne die Operatorklasse neu zu erstellen. Dies wird Ihnen in der Zukunft, wenn Sie erneut ein Upgrade durchführen, Kummer ersparen.

4. *Ich führe PostgreSQL 9.0 aus und kann Geometrien nicht länger in OpenJump, Safe FME und einigen anderen Werkzeugen lesen/schreiben.*

In PostgreSQL 9.0+ wurde die Standardcodierung für »bytea«-Daten in hexadezimal geändert und ältere JDBC-Treiber gehen immer noch vom Escape-Format aus. Dies beeinflusste einige Anwendungen wie Java-Programme, die ältere JDBC-Treiber benutzen oder .NET-Anwendungen, die ältere »npgsql«-Treiber verwenden, die das frühere Verhalten von ST_AsBinary erwarten. Es gibt zwei Herangehensweisen, dies wieder zum Laufen zu bringen. Sie können Ihren JDBC-Treiber auf die neueste PostgreSQL-9.0-Version aktualisieren. Diese erhalten Sie unter <http://jdbc.postgresql.org/download.html>. Falls Sie eine .NET-Anwendung ausführen, können Sie Npgsql 2.0.11 oder neuer verwenden. Dies können Sie von http://pgfoundry.org/frs/?group_id=1000140, wie im [Blog-Eintrag von Francisco Figueiredo zur Veröffentlichung von Npgsql 2.0.11](#) beschrieben, herunterladen. Falls das Aktualisieren Ihres PostgreSQL-Treibers nicht in Frage kommt, können Sie die Voreinstellung mit der folgenden Änderung auf das vorherige Verhalten zurücksetzen:

```
ALTER DATABASE mypostgisdb SET bytea_output='escape';
```

5. *Ich habe versucht, meine Geometriespalte mit PgAdmin anzusehen, aber sie ist leer. Was ist los?*

PgAdmin zeigt bei großen Geometrien nichts an. Was ist der beste Weg, um zu prüfen, ob sich in Ihren Geometriespalten Daten befinden?

```
-- Diese Abfrage gibt keine Datenbankeinträge zurück, falls alle Geometriefelder ←
  ausgefüllt sind
SELECT somefield FROM mytable WHERE geom IS NULL;
```

```
-- Um nur mitzuteilen, wie groß Ihre Geometrie ist, führen Sie eine Abfrage in der ←
  folgenden Form aus.
-- Sie wird Ihnen die Höchstzahl von Punkten mitteilen, die Sie in Ihren ←
  Geometriespalten haben.
SELECT MAX(ST_NPoints(geom)) FROM sometable;
```

6. *Welche Art geometrischer Objekte kann ich speichern?*

Sie können Punkte, Linien, Polygone, Objekte aus mehreren Punkten, Linien und Polygonen, sowie Geometriesammlungen speichern. In PostGIS 2.0 und höher können Sie auch TINs und Polyederflächen im Basistyp »geometry« speichern. Diese werden im »Well-known Text«-Open-GIS-Textformat (mit XYZ-, XYM- und XYZM-Erweiterungen) angegeben. Derzeit werden drei Datentypen unterstützt: Der Standard-OGC-Datentyp »geometry«, der für die Messung ein flaches Koordinatensystem verwendet, der Datentyp »geography«, der ein geodätisches Koordinatensystem benutzt (nicht OGC, aber Sie finden einen ähnlichen Typ in Microsoft SQL Server 2008+). Nur WGS 84 long lat (SRID:4326) wird vom Datentyp »geography« unterstützt. Das neueste Familienmitglied der räumlichen PostGIS-Typenfamilie ist »raster« zum Speichern und Analysieren von Rasterdaten. »raster« hat seine eigene FAQ. Weitere Einzelheiten finden Sie unter Chapter 10 und Chapter 9.

7. *Ich bin ganz verwirrt. Welchen Datenspeicher soll ich verwenden, »geometry« oder »geography«?*

Kurze Antwort: »geography« ist ein neuer Datentyp, der große Bereiche von Distanzvermessungen unterstützt, die meisten Berechnungen damit sind derzeit langsamer als bei »geometry«. Falls Sie »geography« benutzen, müssen Sie nicht viel über flache Koordinatensysteme lernen. »geography« ist im Allgemeinen am besten, falls alles, was Sie interessiert, das Vermessen von Entfernungen und Längen ist und Sie Daten von überall in der Welt haben. Der Datentyp »geometry« ist ein alter Datentyp, den viel mehr Funktionen unterstützen, der eine größere Unterstützung von Werkzeugen Dritter genießt und mit dem Transaktionen generell schneller sind, manchmal um bis zum Zehnfachen schneller für große Geometrien. »geometry« ist die beste Wahl, wenn Sie sich in räumlichen Bezugssystemen sicher fühlen oder Sie mit lokalisierten Daten umgehen, bei denen alle Ihre Daten in ein einziges **räumliches Bezugssystem (spatial reference system/SRID)** passen oder falls Sie eine große Menge räumlicher Daten verarbeiten wollen. Hinweis: Es ist ziemlich einfach, einmalige Umwandlungen zwischen den zwei Typen vorzunehmen, um von den Vorteilen beider zu profitieren. Was derzeit unterstützt wird und was nicht, erfahren Sie unter Section 14.11. Lange Antwort: Eine ausführlichere Erörterung finden Sie unter Section 4.2.2 und **function type matrix**.

8. *Ich habe tiefergehende Fragen über »geography«, wie beispielsweise welche Größe einer geografischen Region kann ich in eine »geography«-Spalte stopfen und trotzdem noch vernünftige Antworten erhalten. Gibt es Beschränkungen wie Pole, etwas im Feld, das in eine Hemisphäre passen muss (wie bei SQL Server 2008), Geschwindigkeit etc.?*

Ihre Fragen sind zu tiefgehend und komplex, um in diesem Abschnitt angemessen beantwortet werden zu können. Bitte lesen Sie unsere Section 4.2.3.

9. Wie füge ich ein GIS-Objekt in die Datenbank ein?

Zuerst müssen Sie eine Tabelle mit einer Spalte des Typs »geometry« oder »geography« erstellen, die Ihre GIS-Daten bereithält. Das Speichern des Datentyps »geography« unterscheidet sich etwas vom Speichern des Datentyps »geometry«. Einzelheiten über das Speichern von »geography« finden Sie unter Section 4.2.1. Für »geometry«: Verbinden Sie Ihre Datenbank mit `psql` und probieren Sie folgenden SQL-Befehl:

```
CREATE TABLE gtest ( gid serial primary key, name varchar(20)
, geom geometry(LINESTRING) );
```

Falls die Definition der Spalte »geometry« fehlschlägt, haben Sie wahrscheinlich die PostGIS-Funktionen und -Objekte nicht in Ihre Datenbank geladen oder Sie verwenden eine Version von PostGIS vor 2.0. Siehe Section 2.4. Dann können Sie mittels eines SQL-INSERT-Befehls eine Geometrie in Ihre Tabelle einfügen. Das GIS-Objekt selbst ist mit dem Format »well-known-text« des OpenGIS-Konsortiums formatiert:

```
INSERT INTO gtest (ID, NAME, GEOM)
VALUES (
  1,
  'First Geometry',
  ST_GeomFromText('LINESTRING(2 3,4 5,6 5,7 8)')
);
```

Mehr Informationen über weitere GIS-Objekte finden Sie in der [Objektreferenz](#). So sehen Sie sich Ihre GIS-Daten in der Tabelle an:

```
SELECT id, name, ST_AsText(geom) AS geom FROM gtest;
```

Der Rückgabewert sollte etwa so aussehen:

```
id | name           | geom
---+-----+-----
  1 | First Geometry | LINESTRING(2 3,4 5,6 5,7 8)
(1 row)
```

10. Wie erstelle ich eine räumliche Abfrage?

Auf die gleiche Weise, wie alle anderen Datenbankabfragen erstellt werden, als Kombination von Rückgabewerten, Funktionen und booleschen Tests. Es gibt bei räumlichen Abfragen zwei Aspekte, die Sie beim Erstellen Ihrer Abfrage im Hinterkopf behalten sollten: Es gibt einen räumlichen Index, von dem Sie Gebrauch machen können, und führen Sie aufwendige Berechnungen auf einer großen Zahl von Geometrien durch? Im Allgemeinen werden Sie den »Überschneidungsoperator« (&&) benutzen wollen, der prüft, ob sich die Hüllquader der Objekte überschneiden. Der Hauptnutzen des &&-Operators besteht darin, dass er, falls ein räumlicher Index zum Beschleunigen des Tests verfügbar ist, Gebrauch davon macht. Dies kann Abfragen sehr stark beschleunigen. Sie werden auch von räumlichen Funktionen wie `Distance()`, `ST_Intersects()`, `ST_Contains()` und `ST_Within()` Gebrauch machen, um die Ergebnisse Ihrer Suche einzugrenzen. Die meisten räumlichen Abfragen enthalten sowohl einen indizierten als auch einen räumlichen Funktionstest. Der Indextest begrenzt die Auswahl von Tupeln auf nur diejenigen Tupel, die die Bedingung, die von Interesse ist, treffen könnten. Die räumlichen Funktionen werden dann verwendet, um die Bedingung genau zu prüfen.

```
SELECT id, the_geom
FROM thetable
WHERE
  ST_Contains(the_geom, 'POLYGON((0 0, 0 10, 10 10, 10 0, 0 0))');
```

11. Wie kann ich räumliche Abfragen auf großen Tabellen beschleunigen?

Schnelle Abfragen großer Tabellen sind die *Daseinsberechtigung* räumlicher Datenbanken (neben der Unterstützung von Transaktionen). Daher ist es wichtig, über einen guten Index zu verfügen. Um einen räumlichen Index für eine Tabelle mit einer `geometry`-Spalte zu erstellen, benutzen Sie die Funktion »CREATE INDEX« wie folgt:

```
CREATE INDEX [indexname] ON [tabellenname] USING GIST ( [geometry_spalte] );
```

Die Option »USING GIST« teilt dem Server mit, dass er einen GiST-Index (Generalized Search Tree/Verallgemeinerter Suchbaum) verwenden soll.

**Note**

Es wird von GiST-Indizes angenommen, dass sie verlustbehaftet sind. Verlustbehaftete Indizes verwenden ein Ersatzobjekt (im Fall räumlicher Objekte einen Hüllquader), um einen Index zu erstellen.

Sie sollten außerdem sicherstellen, dass das PostgreSQL-Abfrageplanungsprogramm ausreichende Informationen über Ihren Index hat, um vernünftige Entscheidungen treffen zu können, wann er benutzt wird. Zu diesem Zweck müssen Sie für Ihre Geometrietabellen »Statistiken erstellen«. Unter PostgreSQL 8.0.x und neuer führen Sie einfach den Befehl **VACUUM ANALYZE** aus. Unter PostgreSQL 7.4.x und älter führen Sie den Befehl **SELECT UPDATE_GEOMETRY_STATS()** aus.

12. Warum werden keine R-Baum-Indizes von PostgreSQL unterstützt?

Ältere Versionen von PostGIS verwendeten die PostgreSQL-R-Baum-Indizes. PostgreSQL-R-Bäume wurden jedoch seit Version 0.6 komplett ausrangiert und räumliche Indizierung wird mit dem Schema »R-Tree-over-GiST« bereitgestellt. Unsere Tests haben gezeigt, dass die Suchgeschwindigkeit für native R-Bäume und GiST vergleichbar ist. Native PostgreSQL-R-Bäume haben zwei Einschränkungen, wegen der sie für den Gebrauch mit GIS-Objekten unerwünscht sind (beachten Sie, dass diese Einschränkungen aufgrund der Implementierung nativer PostgreSQL-R-Bäume und nicht wegen des Konzepts der R-Bäume im Allgemeinen bestehen):

- R-Baum-Indizes können in PostgreSQL nicht mit Objekten umgehen, die größer als 8k sind. GiST-Indizes können dies mittels des »verlustbehafteten« Tricks, das Objekt selbst durch seinen Hüllquader zu ersetzen.
- R-Baum-Indizes sind in PostgreSQL nicht »nullsicher«, daher wird das Bilden eines Index auf einer »geometry«-Spalte, die Null-Geometrien enthält, scheitern.

13. Warum soll ich die Funktion `AddGeometryColumn()` und all das andere OpenGIS-Zeug verwenden?

Falls Sie keine OpenGIS-Hilfsfunktionen nutzen möchten, müssen Sie dies nicht. Erstellen Sie einfach Ihre Tabellen in älteren Versionen, indem Sie Ihre »geometry«-Spalten im CREATE-Befehl erstellen. Alle Ihre Geometrien werden SRIDs von -1 haben und die OpenGIS-Metadatentabellen werden *nicht* ordentlich gefüllt. Dies wird jedoch die meisten Anwendungen, die auf PostGIS basieren, zum Abstürzen bringen und es wird generell empfohlen, dass Sie zum Erstellen von »geometry«-Tabellen `AddGeometryColumn()` verwenden. MapServer ist eine dieser Anwendungen, die Gebrauch von `geometry_columns`-Metadaten machen. Insbesondere kann MapServer die SRID der »geometry«-Spalte nutzen, um direkt eine Rückprojektion von Objekten in die korrekte Abbildungsprojektion vorzunehmen.

14. Was ist der beste Weg, alle Objekte innerhalb eines Radius eines anderen Objekts zu finden?

Um die Datenbank möglichst effizient zu nutzen, ist es am besten Radiusabfragen zu verwenden. Diese kombinieren den Radiustest mit einem Hüllquadertest: Der Hüllquadertest benutzt den räumlichen Index, der schnellen Zugriff auf eine Untermenge von Daten gewährt, auf die dann der Radiustest angewendet wird. Die Funktion `ST_DWithin(geometry, geometry, distance)` ist eine praktische Art, eine Entfernungssuche per Index durchzuführen. Dazu wird ein Suchrechteck erstellt, das groß genug ist, um den Entfernungsradius einzuschließen. Dann wird in der indizierten Untermenge der Ergebnisse die genaue Entfernung gesucht. Um zum Beispiel alle Objekte mit 100 Metern Entfernung zu `POINT(1000 1000)` zu finden, wäre die folgende Abfrage gut geeignet:

```
SELECT * FROM geotable
WHERE ST_DWithin(geocolumn, 'POINT(1000 1000)', 100.0);
```

15. Wie kann ich die Koordinaten-Rückprojektion als Teil einer Abfrage durchführen?

Um eine Rückprojektion durchzuführen, müssen sowohl die Quell- als auch die Zielkoordinatensysteme in einer `SPATIAL_REF_S` Tabelle definiert sein und die SRIDs der Geometrien, die rückprojiziert werden, müssen bereits gesetzt sein. Sobald dies erledigt ist, ist die Rückprojektion so einfach wie Bezug auf die gewünschte Ziel-SRID zu nehmen. Nachfolgend wird eine Geometrie auf »NAD 83 long lat« projiziert. Das Folgende funktioniert nur, falls die SRID der Geometrie nicht -1 ist (keine undefinierte räumliche Referenz).

```
SELECT ST_Transform(die_geometrie, 4269) FROM geometrietabelle;
```


16. *Ich führte ein ST_AsEWKT und ST_AsText auf meiner eher großen Geometrie aus und erhielt ein leeres Feld zurück. Was ist passiert?*

Sie verwenden vermutlich PgAdmin oder irgendein anderes Werkzeug, das keine großen Texte ausgibt. Falls Ihre Geometrie groß genug ist, wird sie in diesen Werkzeugen leer erscheinen. Falls Sie sie wirklich sehen oder in WKT ausgeben möchten, verwenden Sie PSQL.

```
--zum Prüfen, ob die Anzahl der Geometrien wirklich leer ist
SELECT count(gid) FROM geotable WHERE the_geom IS NULL;
```

17. *Wenn ich ein ST_Intersects ausführe, sagt es, dass sich meine beiden Geometrien nicht überschneiden, obwohl ICH WEIß, DASS SIE DIES TUN. Was ist passiert?*

Im Allgemeinen kommt das in zwei Fällen vor. Ihre Geometrie ist ungültig – prüfen Sie dies mit **ST_IsValid** oder Sie gehen davon aus, dass sie sich überschneiden, da ST_AsText die Zahlen rundet und Sie viele Dezimalstellen dahinter haben, die Ihnen nicht angezeigt werden.

18. *Ich veröffentliche Software, die PostGIS verwendet. Bedeutet das, dass meine Software wie PostGIS unter der GPL lizenziert werden muss? Muss ich all meinen Code veröffentlichen, falls ich PostGIS benutze?*

Höchstwahrscheinlich nicht. Oracle-Datenbanken laufen zum Beispiel unter Linux. Linux steht unter der GPL, Oracles Datenbank nicht. Muss Oracles Datenbank, die unter Linux läuft, unter der GPL verteilt werden? Nein. In ähnlicher Weise kann Ihre Software die PostgreSQL/PostGIS Datenbank soviel nutzen wie sie will, und kann unter irgendeiner, von Ihnen gewünschten Lizenz vorliegen. Die einzige Ausnahme wäre, wenn Sie Veränderungen am PostGIS-Quellcode vorgenommen hätten und die *veränderte Version verteilen* würden. In diesem Fall müssten Sie den Code Ihres veränderten PostGIS freigeben (aber nicht den Code von Anwendungen, die darauf laufen). Sogar in diesem begrenzten Fall müssten Sie nur den Quellcode an Leute verteilen, denen Sie Binärcode weitergegeben haben. Die GPL verlangt nicht, dass Sie Ihren Quellcode *veröffentlichen*, nur dass Sie ihn mit Leuten teilen, denen Sie Binärcode geben. Die oberen Angaben treffen auch zu, wenn Sie PostGIS in Verbindung mit den optionalen CGAL-aktivierten Funktionen nutzen. Teile von CGAL liegen unter GPL vor, so wie bereits das gesamte PostGIS: die Verwendung von CGAL macht PostGIS nicht mehr GPL als es bereits von vornherein ist.

Chapter 4

PostGIS Anwendung: Datenmanagement und Abfragen

4.1 GIS Objekte

Die GIS Objekte, die von PostGIS unterstützt werden, sind eine Obermenge der durch das OpenGIS Consortium (OGC) festgelegten "Simple Features". Ab der Version 0.9 unterstützt PostGIS sämtliche Objekte und Funktionen, die in der OGC "Simple Features for SQL" Spezifikation normiert sind.

PostGIS hat den Standard um die Unterstützung für 3DZ-, 3DM -und 4D-Koordinaten erweitert.

4.1.1 OpenGIS WKB und WKT

Die OpenGIS Spezifikation standardisiert zwei Möglichkeiten um Geoobjekte darzustellen: die Well-known-Text (WKT) und die Well-Known-Binary (WKB) Darstellung. Sowohl WKT als auch WKB enthalten Information über den Objekttyp und die Koordinaten, die das Objekt bilden.

Beispiele für die Textdarstellung (WKT) von Geoobjekten:

- POINT(0 0)
- LINESTRING(0 0,1 1,1 2)
- POLYGON(((0 0,4 0,4 0,4,0 0),(1 1, 2 1, 2 2, 1 2,1 1)))
- MULTIPOINT((0 0),(1 2))
- MULTILINESTRING((0 0,1 1,1 2),(2 3,3 2,5 4))
- MULTIPOLYGON((((0 0,4 0,4 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))

Die OpenGIS Spezifikation verlangt auch, dass das der Identifikator des Koordinatenreferenzsystem (SRID) im internen Format der Geoobjekte mit abgespeichert ist.

Für die Ein- und Ausgabe dieser Formate stehen die folgenden Schnittstellen zur Verfügung

```
bytea WKB = ST_AsBinary(geometry);  
text WKT = ST_AsText(geometry);  
geometry = ST_GeomFromWKB(bytea WKB, SRID);  
geometry = ST_GeometryFromText(text WKT, SRID);
```


Eine gültige Einfügeanweisung, um ein räumliches OGC-Objekt zu erzeugen und einzufügen, wäre:

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

4.1.2 PostGIS EWKB, EWKT und Normalformen/kanonische Formen

Das OGC Format unterstützt nur 2D-Geometrien, und die dazugehörige SRID ist **niemals** in den Eingabe/Ausgabe-Darstellungen eingebettet.

Zur Zeit sind die erweiterten Formate von PostGIS eine Obermenge von den OGC-Formaten (jeder gültige WKT/WKB ist auch ein gültiger EWKT/EWKB). Dies kann sich in der Zukunft allerdings ändern, insbesondere dann, wenn OGC ein neues Format einführt, welches mit diesen Erweiterungen im Widerspruch steht. Daher sollten SIE sich **BITTE NICHT** auf diese Eigenschaft verlassen!

PostGIS EWKB/EWKT fügt die Unterstützung von 3DM-,3DZ-,4D-Koordinaten mit eingebetteter SRID Information hinzu.

Beispiele für die Textdarstellung (EWKT) von erweiterten Geoobjekten:

- 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 9, 9 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)))

Für die Ein- und Ausgabe dieser Formate stehen die folgenden Schnittstellen zur Verfügung

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

Zum Beispiel würde eine gültige Eingabeanweisung, um räumliche PostGIS Objekte zu erzeugen und einzufügen, wie folgt lauten:

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

Die "kanonische Form" eines PostgreSQL Datentyps ist jene Darstellung die man mit einer einfachen Abfrage (ohne Funktionsaufruf) erhält und bei der sichergestellt ist, dass sie von einem einfachen INSERT, UPDATE oder COPY angenommen wird. Beim geometrischen Datentyp von PostGIS sind dies:

```
- Ausgabe
  - binary: EWKB
    ascii: HEXEWKB (EWKB in hex form)
- Eingabe
  - binary: EWKB
    ascii: HEXEWKB|EWKT
```

Zum Beispiel liest die folgende Anweisung EWKT ein und gibt HEXEWKB in Normalform aus:

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

geometry
-----
0101000020040000000000000000000000000000000000000000000000000000
(1 row)
```

4.1.3 SQL-MM Part 3

Die "SQL Multimedia Applications Spatial" Spezifikation erweitert die SQL Spezifikation für Simple Features indem es eine Reihe von kreisförmig interpolierten Kurven definiert.

Die Definitionen in SQL-MM schließen 3DM-, 3DZ- und 4D-Koordinaten ein, erlauben allerdings nicht das Einbinden von Information über SRID.

Die Erweiterungen zur Well-known-Text-Darstellung werden zur Zeit noch nicht zur Gänze unterstützt. Im Folgenden werden Beispiele für einige einfache gekrümmte Geometrien gezeigt:

- CIRCULARSTRING(0 0, 1 1, 1 0)

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

The CIRCULARSTRING is the basic curve type, similar to a LINESTRING in the linear world. A single segment required three points, the start and end points (first and third) and any other point on the arc. The exception to this is for a closed circle, where the start and end points are the same. In this case the second point MUST be the center of the arc, ie the opposite side of the circle. To chain arcs together, the last point of the previous arc becomes the first point of the next arc, just like in LINESTRING. This means that a valid circular string must have an odd number of points greater than 1.

- COMPOUNDCURVE(CIRCULARSTRING(0 0, 1 1, 1 0),(1 0, 0 1))

Eine zusammengesetzte Kurve ist eine einzelne, durchgängige Kurve, die sowohl gekrümmte (kreisförmige) als auch gerade Segmente aufweist. Dies bedeutet, daß die Komponenten nicht nur wohlgeformt sein müssen, sondern auch der Endpunkt einer jeden Komponente (außer der letzten) mit dem Anfangspunkt der nachfolgenden Komponente zusammenfallen muss.

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

Beispiel einer zusammengesetzten Kurve in einem Kurvenpolygon: CURVEPOLYGON(COMPOUNDCURVE(CIRCULARSTRING(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))

Ein CurvePolygon hat, genau wie ein Polygon, einen äußeren Ring und keinen oder mehrere innere Ringe. Der Unterschied liegt darin, dass ein Ring aus Kreisbögen, Geraden oder zusammengesetzten Strecken bestehen kann.

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

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

Eine MultiCurve ist eine Sammelgeometrie von Kurven, welche aus Geraden, Kreisabschnitte oder zusammengesetzten Abschnitten bestehen kann.

- 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)))

Dies ist eine Sammelgeometrie von Oberflächen, welche (lineare) Polygone oder Kurvenpolygone sein können.

**Note**

Vorgängerversionen von PostGIS 1.4 hatten keine Unterstützung für zusammengesetzte Kurven in einem Kurvenpolygon. Ab PostGIS 1.4 werden zusammengesetzte Kurven in einem Kurvenpolygon unterstützt.

**Note**

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

4.2 PostGIS geographischer Datentyp

Der geographische Datentyp bietet native Unterstützung für Geoobjekte die durch "geographische" Koordinaten (manchmal auch als "geodätische" Koordinaten, "Länge/Breite" oder "Breite/Länge" bezeichnet) festgelegt sind. Geographische Koordinaten sind Kugelkoordinaten, die durch Winkel (in Grad) angegeben werden.

Der geometrische Datentyp von PostGIS beruht auf der Ebene. Die kürzeste Entfernung zwischen zwei Punkten einer Ebene entspricht einer Gerade. Das bedeutet, dass geometrische Berechnungen (wie Flächen, Distanzen, Längen, Schnittpunkte, etc.) im kartesischen Koordinatensystem mit geradlinigen Vektoren ausgeführt werden können.

Der geographische Datentyp von PostGIS beruht auf einer Kugel. Die kürzeste Verbindung zwischen zwei Punkten auf einer Kugeloberfläche ist ein Bogenteil eines Großkreises. D.h., dass Berechnungen auf geographische Datentypen (wie Flächen, Distanzen, Längen, Schnittpunkte, etc.) auf der Kugeloberfläche mit einer komplexeren Mathematik durchgeführt werden müssen. Für genauere Messungen müssen die Berechnungen das Rotationsellipsoid der Erde in Betracht ziehen, wodurch die Mathematik so richtig kompliziert wird.

Da die zugrunde liegende Mathematik wesentlich schwieriger ist, gibt es weniger Funktionen für den geographischen Datentyp, als für den geometrischen Datentyp. Mit der Zeit werden neue Algorithmen hinzugefügt und die Möglichkeiten des geographischen Datentyps erweitert werden.

Eine Beschränkung ist, dass nur das geodätische WGS 84 (SRID:4326) unterstützt wird. Es verwendet einen geographischen Datentyp, der Geograpy genannt wird. Keine der GEOS Funktionen unterstützt diesen neuen Datentyp. Als provisorische Lösung kann man zwischen den geometrischen und den geographischen Datentypen hin- und herkonvertieren.

Der neue geographische Datentyp verwendet die Typmod-Formatangabe von PostgreSQL 8.3+, sodass eine Tabelle mit einem geographischen Attribut in einem einzigen Schritt erstellt werden kann. Es werden alle Formate des OGC-Standards unterstützt, mit Ausnahme von Kurven.

4.2.1 Geographie Grundlagen

Der geographische Datentyp unterstützt nur die einfachsten "Simple Features". Daten vom geometrischen Datentyp, welche eine SRID von 4326 aufweisen, werden implizit in den geographischen Datentyp umgewandelt. Sie können Daten auch entsprechend der EWKT- und EWKB-Konvention einfügen.

- POINT: Erzeugung einer Tabelle mit 2D-Punktgeometrie:

```
CREATE TABLE testgeog(gid serial PRIMARY KEY, the_geog geography(POINT,4326) );
```

Erstellung einer Tabelle für Punkte mit Z-Koordinaten

```
CREATE TABLE testgeog(gid serial PRIMARY KEY, the_geog geography(POINTZ,4326) );
```

- LINESTRING
- POLYGON
- MULTIPOINT

- MULTILINESTRING
- MULTIPOLYGON
- GEOMETRYCOLLECTION

Die neuen geographischen Attribute werden nicht in `geometry_columns` registriert. Sie werden in dem neuen View "geography_columns" registriert. Da dies ist ein View auf den Systemkatalog ist, wird er automatisch auf dem letzten Stand gehalten und es wird keine Funktion ähnlich wie `AddGeom...` benötigt.

Nun überprüfen Sie bitte ob Ihre Tabelle in der gespeicherten Abfrage "geography_columns" aufscheint.

Eine neue Tabelle mit einer Spalte vom Typ Geometrie kann mit der Syntax `CREATE TABLE` erstellt werden. Anders als bei der `GEOMETRY` muss kein eigener `AddGeometryColumns()` Arbeitsgang ausgeführt werden, um die Spalte in den Metadaten zu registrieren.

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

Beachten Sie bitte, dass die Spalte "location" den geographischen Datentyp verwendet und dieser zwei optionale Modifikatoren unterstützt: Einen Typmodifikator, der die geometrische Form und die Dimension der Geometriespalte festlegt; ein Modifikator für SRID, der den Identifikator für das Koordinatenreferenzsystem auf eine bestimmte Zahl einschränkt.

Für den Typmodifikator sind folgende Werte erlaubt: `POINT`, `LINESTRING`, `POLYGON`, `MULTIPOINT`, `MULTILINESTRING`, `MULTIPOLYGON`. Der Modifikator unterstützt auch Einschränkungen der Dimensionalität durch Nachsilben: `Z`, `M` und `ZM`. So erlaubt zum Beispiel ein Modifikator mit dem Wert '`LINESTRINGM`' nur die Eingabe von Linienzügen mit drei Dimensionen, wobei die dritte Dimension als Kilometrierung/measure behandelt wird. Ebenso verlangt '`POINTZM`' die Eingabe von vierdimensionalen Daten.

Der Modifikator für SRID ist zurzeit nur von beschränktem Nutzen, da nur 4326 (WGS84) als Eingabewert erlaubt ist. Wenn Sie keine SRID angeben, so wird der Wert auf 0 (undefiniertes Referenzellipsoid) gesetzt, wobei sämtliche Berechnungen jedenfalls in WGS84 ausgeführt werden.

In der Zukunft werden weitere SRID's eingeführt und Berechnungen auf andere Referenzellipsoide als WGS84 ermöglicht werden.

Sobald Sie Ihre Tabelle erstellt haben, können Sie diese in der Tabelle "geography_columns" sehen:

```
-- Metadaten abfragen
SELECT * FROM geography_columns;
```

Sie können Daten auf dieselbe Art und Weise in die Tabelle einfügen wie Sie dies bei einer Geometriespalte tun würden:

```
-- Ein paar Daten in die Testtabelle einfügen
INSERT INTO global_points (name, location) VALUES ('Town', ST_GeographyFromText('SRID=4326; ↵
    POINT(-110 30)')) );
INSERT INTO global_points (name, location) VALUES ('Forest', ST_GeographyFromText('SRID ↵
    =4326;POINT(-109 29)')) );
INSERT INTO global_points (name, location) VALUES ('London', ST_GeographyFromText('SRID ↵
    =4326;POINT(0 49)')) );
```

Die Erstellung eines Index funktioniert gleich wie beim Datentyp `GEOMETRY`. PostGIS erkennt, dass es sich um den Datentyp `GEOGRAPHY` handelt und erzeugt einen entsprechenden, auf einer Kugeloberfläche basierenden Index anstelle des üblichen planaren Index, der für den Datentyp `GEOMETRY` verwendet wird.

```
-- Einen sphärischen Index auf die Testtabelle legen
CREATE INDEX global_points_gix ON global_points USING GIST ( location );
```

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.

```
-- Eine Distanzabfrage; Beachten Sie bitte, dass London ausserhalb der 1000km Toleranz liegt
SELECT name FROM global_points WHERE ST_DWithin(location, ST_GeographyFromText('SRID =4326;POINT(-110 29)'), 1000000);
```

Sie können die Mächtigkeit von GEOGRAPHY erleben, indem Sie berechnen, wie nahe ein Flugzeug, das von Seattle nach London (LINESTRING(-122.33 47.606, 0.0 51.5)) fliegt, an Reykjavik (POINT(-21.96 64.15)) vorbeikommt.

```
-- Die Entfernung mittels GEOGRAPHY ausrechnen (122.2km)
SELECT ST_Distance('LINESTRING(-122.33 47.606, 0.0 51.5)::geography, 'POINT(-21.96 64.15):: geography');
```

```
-- Die Entfernung mittels GEOMETRIE ausrechnen (13.3 "degrees")
SELECT ST_Distance('LINESTRING(-122.33 47.606, 0.0 51.5)::geometry, 'POINT(-21.96 64.15):: geometry');
```

Mit dem Datentyp GEOGRAPHY wird die wahre, kürzeste Entfernung auf der Kugeloberfläche zwischen Reykjavik und der Flugstrecke entlang des Großkreises von Seattle nach London errechnet.

Great Circle mapper Beim geometrischen Datentyp wird die Entfernung sinnloserweise in einem kartesischen Koordinatensystem zwischen Reykjavik und einer Geraden von Seattle nach London errechnet und auf einer ebenen Weltkarte angezeigt. Dem Namen nach mag das Ergebnis in der Einheit "Grad" angegeben sein, da es aber in keiner Weise irgendeinem wahren Winkel zwischen den Punkten entspricht, ist sogar die Verwendung der Bezeichnung "Grad" falsch.

4.2.2 Wann sollte man den GEOGRAPHY Datentyp dem GEOMETRY Datentyp vorziehen

Der neue GEOGRAPHY Datentyp ermöglicht es, die Koordinaten in Form von Länge und Breite zu speichern. Allerdings mit dem Nachteil, dass für den GEOGRAPHY Datentyp weniger Funktionen als für den GEOMETRY Datentyp vorhanden sind und diese auch mehr CPU-Zeit beanspruchen.

Welchen Datentyp Sie wählen sollte aufgrund der zu erwartene Flächenausdehnung ihrer Anwendung festgelegt werden. Erstrecken sich Ihre Daten über den gesamten Globus oder über eine große kontinentale Fläche, oder sind sie auf einen Staat, ein Land 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 es der geographische Datentyp ein System aufzubauen, bei dem Sie sich nicht um Projektionsdetails kümmern müssen. Sie speichern die Daten als Länge und Breite und verwenden dann jene Funktionen die für den geographischen Datentyp definiert sind.
- Wenn Sie keine Ahnung von Projektionen haben, sich nicht näher damit beschäftigen wollen und die Einschränkungen der verfügbaren Funktionalität für den geographischen Datentyp in Kauf nehmen können, ist es vermutlich einfacher für Sie den geographischen anstatt des geometrischen Datentyps zu verwenden.

Für einen Vergleich, welche Funktionalität von Geography vs. Geometry unterstützt wird, siehe Section 14.11. Für eine kurze Liste mit der Beschreibung der geographischen Funktionen, siehe Section 14.4

4.2.3 Fortgeschrittene FAQ's zum GEOGRAPHY Datentyp

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

Standardmäßig werden alle Entfernungs- und Flächenberechnungen auf dem Referenzellipsoid ausgeführt. Das Ergebnis der Berechnung sollte in lokalen Gebieten gut mit dem planaren Ergebnis zusammenpasst - eine gut gewählte lokale Projektion vorausgesetzt. Bei größeren Gebieten ist die Berechnung über das Referenzellipsoid genauer als irgendeine Berechnung die auf der projizierten Ebene ausgeführt wird. Alle geographischen Funktionen verfügen über eine Option um die Berechnung auf einer Kugel durchzuführen. Dies erreicht man, indem der letzte boolesche Eingabewert auf 'FALSE' gesetzt wird. Dies beschleunigt die Berechnung einigermaßen, insbesondere wenn die Geometrie sehr einfach gestaltet ist.

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

Alle diese Berechnungen wissen weder über Datumsgrenzen noch über Pole Bescheid. Da es sich um sphärische Koordinaten handelt (Länge und Breite), unterscheidet sich eine Geometrie die eine Datumsgrenze überschreitet vom Gesichtspunkt der Berechnung her durch nichts 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 Großkreises 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. Nur weil Sie ganz Europa in einem Polygon speichern *können* heißt das nicht, dass Sie dies auch tun *sollten*.

4.3 Verwendung von OGC-Standards

Die OpenGIS "Simple Features Specification for SQL" standardisiert die Datentypen von Geoobjekten, die Funktionen die benötigt werden um diese zu verarbeiten, sowie die Metadatentabellen. Um sicherzustellen, dass die Metadaten konsistent bleiben, werden Vorgänge wie das Erstellen oder das Löschen einer Geometriespalte, durch dafür eigens von OpenGIS festgelegten Prozeduren ausgeführt.

Es gibt zwei OpenGIS Metadatentabellen: SPATIAL_REF_SYS und GEOMETRY_COLUMNS. Die SPATIAL_REF_SYS Tabelle enthält die numerischen Identifikatoren und textlichen Beschreibungen der in der Datenbank verwendeten Koordinatensysteme.

4.3.1 Die SPATIAL_REF_SYS Tabelle und Koordinatenreferenzsysteme

Die Tabelle "spatial_ref_sys" ist eine mit PostGIS kommende und OGC-konforme Datenbanktabelle, die über 3000 bekannte **Koordinatenreferenzsysteme** enthält, sowie Details zur Koordinatentransformation zwischen diesen.

Obwohl in der PostGIS Tabelle "spatial_ref_sys" über 3000 der gebräuchlichsten Koordinatenreferenzsysteme definiert sind, die mit der Bibliothek "Proj4" gehandhabt werden können, enthält sie nicht alle bekannten Projektionen. Sie können auch ihre eigenen Projektionen in der Tabelle definieren, falls Sie mit den Konstrukten von "Proj4" vertraut sind. Sie sollten nicht außer Acht lassen, dass die meisten Koordinatenreferenzsysteme regional sind und außerhalb des vorgesehenen Bereichs keinen Sinn haben.

Eine hervorragende Quelle zum Auffinden von Koordinatenreferenzsystemen, welche nicht in der Grundmenge enthalten sind, ist <http://spatialreference.org/>

Einige der häufiger eingesetzten Koordinatenreferenzsysteme sind: **4326 - WGS 84 Long Lat**, **4269 - NAD 83 Long Lat**, **3395 - WGS 84 World Mercator**, **2163 - US National Atlas Equal Area**, Koordinatenreferenzsysteme für jede NAD 83, WGS 84 und UTM Zone - UTM Zonen sind ideal für Messungen, decken aber nur 6 Grad breite, vertikale Zonen ab.

Verschiedenste Koordinatenreferenzsysteme "US State Plane" (auf Meter und Fuß basierend) - üblicherweise 2 pro US Staat. Die meisten auf Meter basierten befinden sich in der Grundmenge, aber viele der auf Fuß basierten, oder von ESRI erzeugten müssen von spatialreference.org heruntergeladen werden.

Genauere Angaben zur Ermittlung der UTM Zone für ein bestimmtes Gebiet finden Sie unter **utmzone PostGIS plpgsql helper function**.

Die SPATIAL_REF_SYS Tabelle ist folgendermaßen definiert:

```
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 SPATIAL_REF_SYS Spalten folgendermaßen:

SRID Ein ganzzahliger Wert, der das Koordinatenreferenzsystem (SRS) innerhalb der Datenbank eindeutig ausweist.

AUTH_NAME Der Name des Standards oder der Normungsorganisation unter dem dieses Koordinatenreferenzsystem zitiert wird. Zum Beispiel ist "EPSG" ein gültiger AUTH_NAME.

AUTH_SRID Die von der in AUTH_NAME zitierten Quelle festgelegte ID des Koordinatenreferenzsystems. Im Falle von EPSG ist dies der EPSG Projektionscode.

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

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

Für eine Auflistung der EPSG Projektionscodes und deren entsprechende WKT Darstellung siehe <http://www.opengeospatial.org/>. Eine allgemeine Erläuterung zu WKT finden Sie in der OpenGIS "Coordinate Transformation Services Implementation Specification" unter <http://www.opengeospatial.org/standards>. Information zur European Petroleum Survey Group (EPSG) und deren Datenbank über Koordinatenreferenzsysteme finden Sie unter <http://www.epsg.org>.

PROJ4TEXT PostGIS verwendet die Bibliothek "Proj4" zur Koordinatentransformation. Die Spalte PROJ4TEXT enthält eine Proj4 Zeichenfolge mit der Definition des Koordinatensystems für eine bestimmte SRID. Zum Beispiel:

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

Weiterführende Information finden Sie auf der Proj4 Webseite unter <http://trac.osgeo.org/proj/>. Die Datei spatial_ref_sys.sql enthält sowohl SRTEXT als auch PROJ4TEXT Definitionen aller EPSG Projektionen.

4.3.2 Die gespeicherte Abfrage GEOMETRY_COLUMNS

In Vorgängerversionen von PostGIS 2.0.0 war "geometry_columns" als Tabelle angelegt, die direkt editiert werden konnte und manchmal asynchron zur aktuellen Definition der Geometriespalte wurde. In PostGIS 2.0.0 wurde GEOMETRY_COLUMNS zu einem View, der nach aussen hin dieselbe Struktur wie die Tabelle der Vorgängerversionen aufweist, die Information jedoch aus dem Systemkatalog der Datenbank holt. Der View ist folgendermaßen aufgebaut:

```
\d geometry_columns
```

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

Die Bedeutung der Attribute hat sich seit den Vorgängerversionen nicht geändert:

F_TABLE_CATALOG, F_TABLE_SCHEMA, F_TABLE_NAME Der vollständige Name der Tabelle welche die Geometriespalte enthält. Die Bezeichnungen "catalog" und "schema" sind Oracle-isch. Es gibt keine Entsprechung in PostgreSQL für "catalog", weshalb diese Spalte leer bleibt - für "schema" wird der Name des Schemas in PostgreSQL verwendet (standardmäßig public).

F_GEOMETRY_COLUMN Der Name der Geometriespalte in der Feature-Tabelle.

COORD_DIMENSION Die räumliche Dimension (2-, 3- oder 4-dimensional) der Geometriespalte.

SRID Der Identifikator des Koordinatenreferenzsystems, welches für die Geometrie in dieser Tabelle verwendet wird. Dieser ist ein Fremdschlüssel der sich auf die Tabelle SPATIAL_REF_SYS bezieht.

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 (gemischte Datentypen) können Sie den Datentyp "GEOMETRY" verwenden.



Note

Dieses Attribut gehört (wahrscheinlich) nicht zur OpenGIS Spezifikation, wird aber benötigt um homogene Datentypen zu gewährleisten.

4.3.3 Erstellung einer räumlichen Tabelle

Die Erzeugung einer Tabelle mit spatialen Daten kann in einem Schritt ausgeführt werden. Dies wird im folgende Beispiel demonstriert, welches eine Straßentabelle mit einer geometrischen Spalte für 2D Linienzüge in WGS84 Länge/Breite erzeugt

```
CREATE TABLE ROADS ( ID int4
                      , ROAD_NAME varchar(25), geom geometry(LINESTRING,4326) );
```

Wir können zusätzliche Spalten hinzufügen, indem wir den normalen ALTER TABLE Befehl verwenden. Wir zeigen dies im nächsten Beispiel, wo wir einen 3D-Linienzug hinzufügen.

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

Aus Gründen der Abwärtskompatibilität können Sie eine räumliche Tabelle immer noch in zwei Schritten mit den Management Funktionen erstellen.

- Erstellen Sie zuerst eine normale, nicht-räumliche Tabelle.

Zum Beispiel: **CREATE TABLE ROADS (ID int4, ROAD_NAME varchar(25))**

- Eine räumliche Spalte zu einer Tabelle mit der OpenGIS Funktion "AddGeometryColumn" hinzufügen. Siehe [AddGeometryColumn](#) für weitere Details.

Die Syntax lautet:


```
AddGeometryColumn(  
    <schema_name>,  
    <table_name>,  
    <column_name>,  
    <srid>,  
    <type>,  
    <dimension>  
)
```

Oder, im aktuellen Schema:

```
AddGeometryColumn(  
    <table_name>,  
    <column_name>,  
    <srid>,  
    <type>,  
    <dimension>  
)
```

Beispiel1: **SELECT AddGeometryColumn('public', 'roads', 'geom', 423, 'LINESTRING', 2)**

Beispiel2: **SELECT AddGeometryColumn('roads', 'geom', 423, 'LINESTRING', 2)**

Ein SQL-Beispiel das eine Tabelle erzeugt und eine räumliche Spalte hinzufügt (die Existenz einer SRID von 128 vorausgesetzt):

```
CREATE TABLE parks (  
    park_id    INTEGER,  
    park_name  VARCHAR,  
    park_date  DATE,  
    park_type  VARCHAR  
);  
SELECT AddGeometryColumn('parks', 'park_geom', 128, 'MULTIPOLYGON', 2 );
```

Ein anderes Beispiel, welches den allgemeinen "geometry" Datentyp und die unbestimmte SRID, mit dem Wert 0, verwendet:

```
CREATE TABLE roads (  
    road_id INTEGER,  
    road_name VARCHAR  
);  
SELECT AddGeometryColumn('roads', 'roads_geom', 0, 'GEOMETRY', 3 );
```

4.3.4 Geometrische Spalten in "geometry_columns" händisch registrieren

Der Ansatz mit AddGeometryColumn() erzeugt eine Geometriespalte mit einem bestimmten Datentyp. Dieser Datentyp und die geometrische Dimension können über den View `geometry_columns` abgefragt werden. Ab PostGIS 2.0.0 werden sämtliche Geometriespalten automatisch registriert und "geometry_columns" kann nicht mehr editiert werden.

Wenn Ihre Geometriespalten in einer Tabelle oder einem View generisch erstellt wurden und keine Constraints zugewiesen haben, so werden diese in dem View "geometry_columns" zwar aufgelistet, haben aber keine Dimension, keinen Datentyp und keine SRID.

Zwei Fälle bei denen dies passieren kann und wo Sie die Funktion "AddGeometryColumn" nicht verwenden können sind SQL-Views und Masseneinserts. Beim Fall von Masseneinserts können Sie die Registrierung in der Tabelle "geometry_columns" korrigieren, indem Sie auf die Spalte einen CONSTRAINT setzen oder ein "ALTER TABLE" durchführen. Ab PostGIS 2.0, 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.

```
--Angenommen Sie erzeugen folgende gespeicherte Abfrage
CREATE VIEW public.vwmytablemercator AS
    SELECT gid, ST_Transform(geom,3395) As geom, f_name
    FROM public.mytable;

-- Um diese in PostGIS 2.0+ richtig zu registrieren
-- benötigt es eine Typumwandlung der Geometrie
--
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;

-- Wenn Sie sicher sind, das es sich bei der Geometrie um ein 2D-Polygon handelt, können ←
-- Sie folgendes tun
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;
```

```
--Angenommen Sie haben eine abgeleitete Tabelle über ein Masseninsert erzeugt
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);

--Einen 2D Index auf die neue Tabelle legen
CREATE INDEX idx_myschema_myspecialpois_geom_gist
ON myschema.my_special_pois USING gist(geom);

-- Falls Ihre Punkte 3D-Punkte oder 3M-Punkte sind,
-- können Sie einen ND-Index anstatt eines 2D-Indexes erstellen
-- das geht so:
CREATE INDEX my_special_pois_geom_gist_nd
ON my_special_pois USING gist(geom gist_geometry_ops_nd);

-- Um die Geometriespalte der neuen Tabelle in geometry_columns händisch zu registrieren
-- Beachten Sie bitte, dass dieser Ansatz sowohl für PostGIS2.0+ als auch für PostGIS 1.4+ ←
-- funktioniert
-- Bei PostGIS 2.0+ wird dabei auch die zugrundeliegende Struktur der Tabelle geändert
-- um die Spalte Typmod basiert zu machen.
-- Bei Vorgängerversionen von PostGIS 2.0, kann diese Methode auch verwendet werden, um ←
-- Views zu registrieren
SELECT populate_geometry_columns('myschema.my_special_pois'::regclass);

-- Wenn Sie PostGIS 2.0 verwenden und aus welchem Grund auch immer
-- das alte Verhalten mit auf CONSTRAINTs basierender Definition benötigen
-- (wie im Fall von vererbten Tabellen bei denen nicht alle Kindtabellen denselben Datentyp ←
-- und dieselbe SRID aufweisen)
-- das neue optionale Argument "use_typmod" auf FALSE setzen
SELECT populate_geometry_columns('myschema.my_special_pois'::regclass, false);
```

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

```
CREATE TABLE pois_ny(gid SERIAL PRIMARY KEY
    , poi_name text, cat varchar(20)
    , geom geometry(POINT,4326) );
SELECT AddGeometryColumn('pois_ny', 'geom_2160', 2160, 'POINT', 2, false);
```

In psql:

```
\d pois_ny;
```

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

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

Beide registrieren sich korrekt in "geometry_columns"

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

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

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

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

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

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

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

Dies kann sich bei zukünftigen Versionen von PostGIS ändern, vorerst müssen Sie aber folgendes ausführen, um die auf Constraint basierende Spalte eines View korrekt zu registrieren:

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

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

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

4.3.5 Wahrung der OGC-Konformität von Geometrien

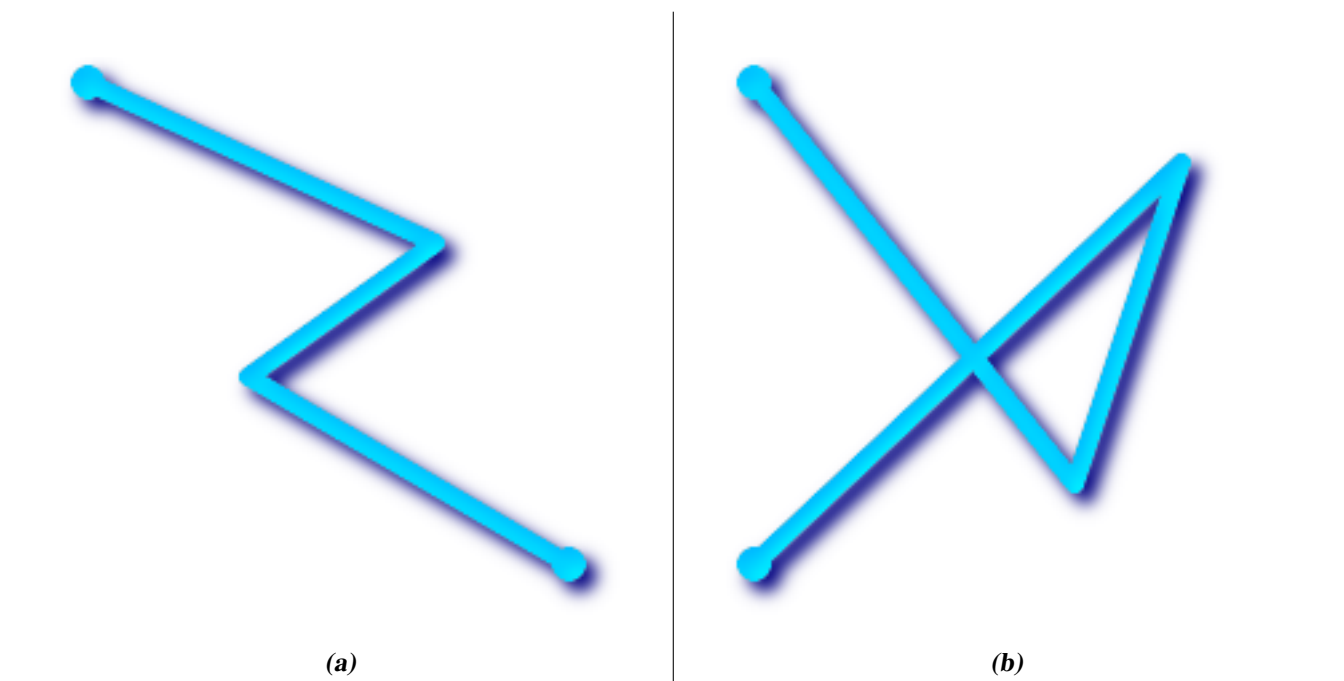
PostGIS ist mit den Open Geospatial Consortium (OGC) OpenGIS Spezifikationen konform. Daher setzen viele PostGIS Methoden voraus, dass die Geometrien mit denen sie rechnen sowohl "Simple" als auch "Valid" sind. Zum Beispiel hat es keinen Sinn, die Fläche eines Polygons zu berechnen, das eine Insel aufweist die ausserhalb des Polygons festgelegt ist, oder ein Polygon aus einer Begrenzungslinie zu konstruieren, welche nicht "simple" ist.

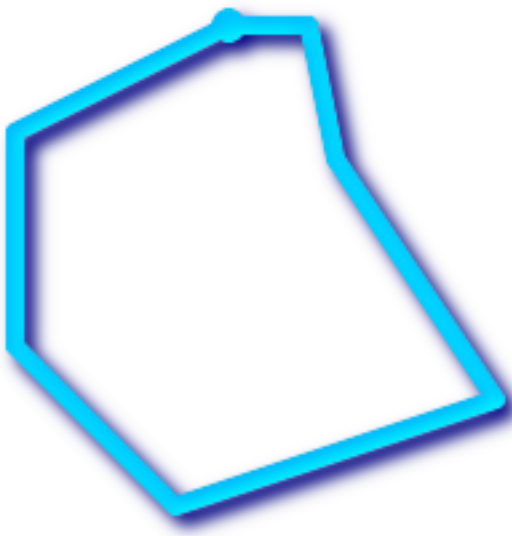
Entsprechend der OGC Spezifikationen ist eine *simple/einfache* Geometrie eine solche, die sich nicht selbst überschneidet oder berührt und bezieht sich in erster Linie auf 0- und 1-dimensionale Geometrien (insbesondere `[MULTI]POINT`, `[MULTI]LINESTRING`). Andererseits bezieht sich die Validität/Gültigkeit einer Geometrie hauptsächlich auf 2-dimensionale Geometrien (insbesondere `[MULTI]POLYGON`) und definiert die Menge an Aussagen, welche ein valides/gültiges Polygon auszeichnen. Die Beschreibung einer jeden geometrischen Klasse schließt bestimmte Bedingungen mit ein, welche die Simplität und Validität von Geometrien näher beschreiben.

Da ein `POINT` ein 0-dimensionales geometrisches Objekt ist, ist er von vornherein *simple*.

`MULTIPOINTS` sind *simple*, if no two coordinates wenn sich nicht zwei Koordinate (`POINTS`) decken (identische Koordinatenpaare).

Ein `LINESTRING` ist *simple*, wenn er nicht zweimal durch denselben `POINT` geht (ausgenommen bei Endpunkten, wo dieser als linearer Ring benannt wird und zusätzlich als geschlossen angesehen wird).





(c)



(d)

(a) und (c) sind simple LINESTRINGS, (b) und (d) nicht.

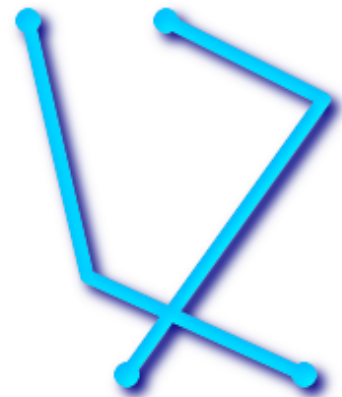
Ein MULTILINESTRING ist nur dann *simple*, wenn alle seine Elemente "simple" sind und die einzigen Überschneidungen zwischen zwei Elementen nur an jenen POINTs auftreten, die an den Begrenzungen der beiden Elemente liegen.



(e)



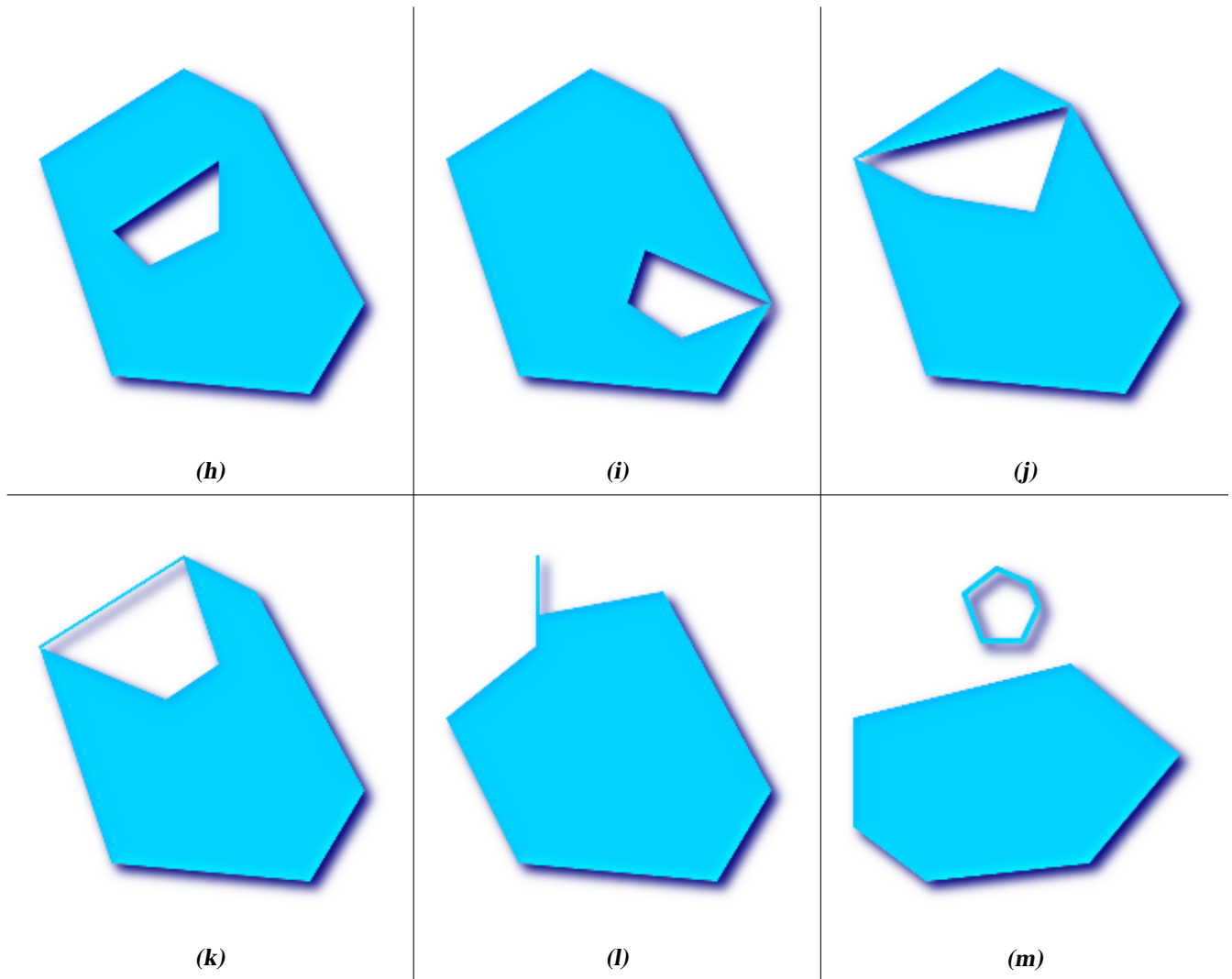
(f)



(g)

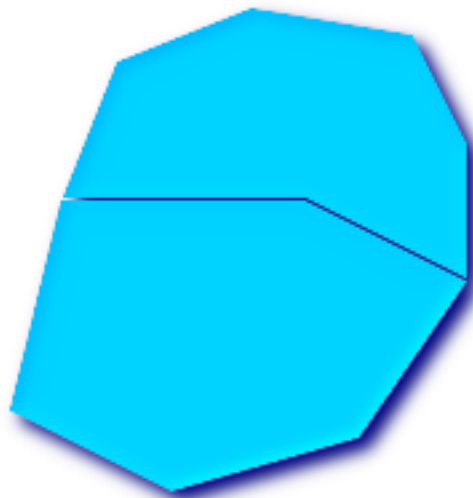
(e) und (f) sind simple MULTILINESTRINGS, (g) nicht.

Definitionsgemäß ist ein POLYGON immer *simple*. Es ist *valid*, wenn sich keine zwei Ringe an der Begrenzung (bestehend aus einem äußeren Ring und inneren Ringen) kreuzen. Die Begrenzung eines POLYGONS darf an einem POINT schneiden, allerdings nur als Tangente (insbesondere nicht an einer Linie). Ein POLYGON darf keine Schnittlinien oder "Spikes" aufweisen und die inneren Ringe müssen zur Gänze im äußeren Ring enthalten sein.

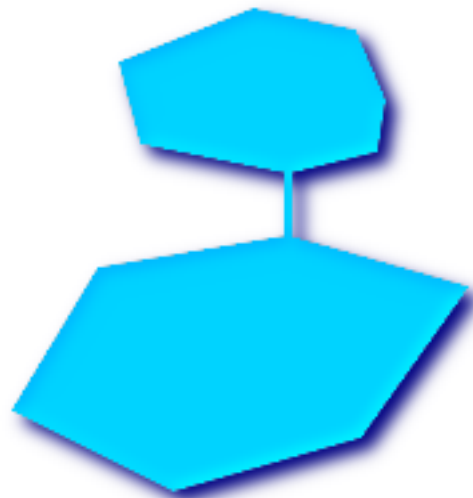


(h) und **(i)** sind valide `POLYGONE`, **(j-m)** können nicht als einzelne `POLYGONE` dargestellt werden, aber **(j)** und **(m)** können als ein valides `MULTIPOLYGON` dargestellt werden.

Ein `MULTIPOLYGON` ist dann und nur dann *valide*, wenn alle seine Elemente valide sind und sich das Innere zweier Elemente nicht überschneidet. Die Begrenzungen zweier Elemente können sich berühren, allerdings nur an einer endlichen Anzahl von `POINTS`.



(n)



(o)

(n) und (o) sind keine validen MULTIPOLYGONE. Hingegen ist (p) valid.

Die meisten der von der GEOS Bibliothek implementierten Funktionen beruhen auf der Annahme, dass Ihre Geometrien entsprechend der OpenGIS Simple Feature Spezifikation, valide sind. Um die Simplität und Validität von Geometrien festzustellen, können Sie **ST_IsSimple()** und **ST_IsValid()** verwenden.

```
-- Üblicherweise hat es keinen Sinn lineare Geometrien
-- auf Validität zu überprüfen, da immer TRUE zurückgegeben wird.
-- Aber in diesem Beispiel erweitert PostGIS die OGC Definition von IsValid
-- indem es FALSE zurückgibt, wenn ein LineString weniger als 2 *eindeutige* Stützpunkte ←
-- aufweist.
gisdb=# SELECT
      ST_IsValid('LINESTRING(0 0, 1 1)'),
      ST_IsValid('LINESTRING(0 0, 0 0, 0 0)');

 st_isvalid | st_isvalid
-----+-----
      t      |          f
```

Standardmäßig überprüft PostGIS eine Geometrieingabe nicht auf Validität, da Validitätstests von komplexen Geometrien, insbesondere Polygonen, viel CPU Zeit beanspruchen. Fall Sie Ihren Datenquellen nicht trauen, können Sie eine Überprüfung Ihrer Tabellen durch eine "Check Constraint"/Prüfbeschränkung erzwingen:

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

Falls Sie irgendwelche seltsamen Fehlermeldungen, wie "GEOS Intersection() threw an error!" oder "JTS Intersection() threw an error!", erhalten, obwohl sie eine PostGIS Funktion mit validen Eingabegemetrien aufgerufen haben, ist es wahrscheinlich dass Sie einen Fehler in PostGIS oder einer von PostGIS verwendeten Bibliothek gefunden haben. In diesem Fall sollten Sie das PostGIS Entwicklerteam kontaktieren. Dasselbe gilt, wenn eine PostGIS Funktion auf eine valide Eingabegeometrie eine invalide Geometrie zurückgibt.

**Note**

Eine streng konforme OGC-Geometrie hat keine Z- oder M-Werte. Die Funktion `ST_IsValid()` betrachtet höhere geometrische Dimensionen nicht als invalide! Aufrufe von `AddGeometryColumn()` fügen eine Check-Constraint für die geometrische Dimension hinzu, weshalb es hier ausreicht 2 anzugeben.

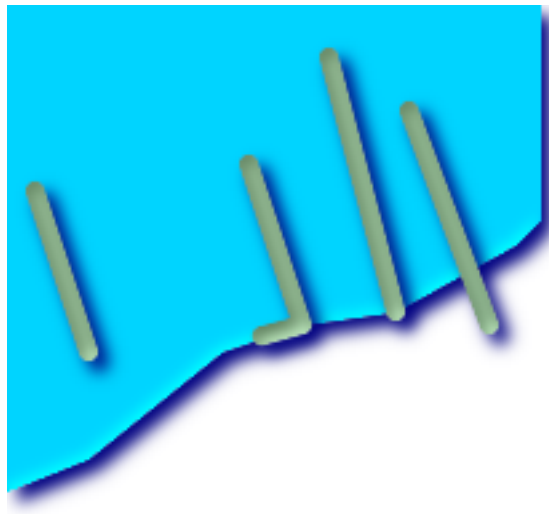
4.3.6 DE-9IM-Matrix (DE-9IM)

Manchmal kommt es vor, dass die typischen räumlichen Aussagen (`ST_Contains`, `ST_Crosses`, `ST_Intersects`, `ST_Touches`, ...) an sich nicht ausreichen, um den verlangten räumlichen Filter auf geeignete Weise zu liefern.



Betrachten Sie zum Beispiel einen linearen Datensatz, der ein Straßennetz darstellt. Es kann sein, dass ein GIS-Analyst die Aufgabe hat, alle Straßenabschnitte herauszufinden, die sich gegenseitig nicht an einem Punkt sondern entlang einer Linie kreuzen, da dies möglicherweise irgendeiner Unternehmensvorschrift widerspricht. In diesem Fall liefert `ST_Crosses` nicht den passenden räumlichen Filter, da bei linearen Geoobjekten nur dann TRUE zurückgegeben wird wenn sie sich an einem Punkt kreuzen.

Eine zweistufige Lösung kann sein, dass man zuerst die eigentliche Verschneidung (`ST_Intersection`) von Straßenabschnittsparen die sich räumlich überschneiden (`ST_Intersects`) ausführt, und anschließend den `ST_GeometryType` der Verschneidung mit 'LINESTRING' vergleicht (vermutlich muss man sich mit Fällen auseinandersetzen die GEOMETRYCOLLECTIONS von [MULTI]POINTS, [MULTI]LINESTRINGS, etc. zurückgeben). Eine elegantere/schnellere Lösung wäre sicherlich wünschenswert.



Ein zweites [theoretisches] Beispiel wäre, dass ein GIS-Analyst versucht alle Anlegestellen oder Kais welche die Begrenzung eines Sees entlang einer Linie überschneiden und bei denen nur ein Ende der Anlegestelle an der Küste liegt. Anders ausgedrückt, wo ein Kai im, aber nicht zur Gänze im See liegt, da er den See entlang einer Linie schneidet und seine Endpunkte sowohl zur Gänze in und auf der Begrenzung des Sees liegen. Dazu kann es nötig sein, dass der Analyst eine Kombination von Aussagen ausführen muss, um die gesuchten Geoobjekte herauszufiltern:

- `ST_Contains(lake, wharf) = TRUE`
- `ST_ContainsProperly(lake, wharf) = FALSE`
- `ST_GeometryType(ST_Intersection(wharf, lake)) = 'LINESTRING'`
- `ST_NumGeometries(ST_Multi(ST_Intersection(ST_Boundary(wharf), ST_Boundary(lake)))) = 1`
... (überflüssig zu erwähnen, dass dies ziemlich kompliziert werden kann)

Somit stürzen wir uns auf die DE-9IM-Matrix, oder kurz DE-9IM

4.3.6.1 Theorie

Entsprechend der [OpenGIS Simple Features Implementation Specification for SQL](#), "Der grundlegende Ansatz um zwei Geometrien zu vergleichen, besteht in einer paarweisen Überprüfung des Inneren, der Begrenzung und des Äusseren zweier Geometrien und der Einstufung der Beziehung zwischen den zwei Geometrien an Hand den Einträgen in die resultierende 'Verschneidungs'-Matrix." [frei übersetzt]

Boundary

Die Begrenzung einer Geometrie ist die geometrische Grundmenge der nächst kleineren Dimension. Bei `POINTS`, die die Dimension 0 haben ist die Begrenzung die leere Menge. Die Begrenzung eines `LINESTRINGs` sind die zwei Endpunkte. Bei `POLYGONen` entspricht die Begrenzung jenen Linien, die die äußeren und inneren Ringe zusammensetzen.

Interior

Die Innenseite/interior einer Geometrie besteht aus jenen Punkten einer Geometrie, die zurückbleiben, wenn die Außenbegrenzung/boundary entfernt wird. Bei `POINTS` ist die Innenseite der `POINT` selbst. Die Innenseite eines `LINESTRINGs` ist die Menge der echten Punkte zwischen den Endpunkten. Bei `POLYGONen` entspricht die Innenseite der Fläche innerhalb des Polygons.

Exterior

Die Außenseite/exterior einer Geometrie ist durch die Grundgesamtheit gegeben. Das ist jene Fläche, die nicht auf der Innenseite/interior oder auf der Begrenzung der Geometrie liegt.

Gegeben sei die Geometrie a , wobei $I(a)$, $B(a)$, und $E(a)$ das *Innere/Interior*, die *Begrenzung/Boundary* und das *Äussere/Exterior* von a sind; die mathematische Formulierung der Matrix lautet:

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

Wobei $\dim(a)$, so wie von **ST_Dimension** festgelegt, die Dimension von a ist, aber zu der Domäne von $\{0,1,2,T,F,*\}$ gehört.

- 0 => point
- 1 => line
- 2 => area
- T => {0,1,2}
- F => Leere Menge
- * => braucht nicht zu kümmern

Bildlich schaut dies für zwei überlappende Polygoneometrien folgendermaßen aus:





	Interior	Boundary	Exterior
Interior	 $\dim(...) = 2$	 $\dim(...) = 1$	 $\dim(...) = 2$
Boundary	 $\dim(...) = 1$	 $\dim(...) = 0$	 $\dim(...) = 1$
Exterior	 $\dim(...) = 2$	 $\dim(...) = 1$	 $\dim(...) = 2$

Von links nach rechts und von oben nach unten gelesen wird die Dimensionsmatrix durch **'212101212'** dargestellt.

Eine Beziehungsmatrix, welche das erste Beispiel von zwei Linien, die sich auf einer Linie schneiden, abbildet, würde **'102101FF2'** entsprechen.

```
-- Identifizierung der Strassenabschnitte, die eine Linie kreuzen
SELECT a.id
FROM roads a, roads b
WHERE a.id != b.id
AND a.geom && b.geom
AND ST_Relate(a.geom, b.geom, '1*1***1**');
```

Eine Beziehungsmatrix, welche das zweite Beispiel mit den Kais, die teilweise an der Uferlinie des Sees liegen, abbildet, würde **'102101FF2'** entsprechen.

```
-- Ermittlung von Dämmen, die teilweise an der Uferlinie eines Sees liegen
SELECT a.lake_id, b.wharf_id
```

```
FROM lakes a, wharfs b
WHERE a.geom && b.geom
AND ST_Relate(a.geom, b.geom, '102101FF2');
```

Für weiterführende Information siehe:

- [OpenGIS Simple Features Implementation Specification for SQL](#) (Version 1.1, Abschnitt 2.1.13.2)
- [DE-9IM-Matrix \(DE-9IM\)](#)
- [Geotools: Mengentheoretische Topologie und die DE-9IM-Matrix](#)
- *Encyclopedia of GIS* By Hui Xiong

4.4 GIS (Vector) Daten laden

Sobald Sie eine räumliche Tabelle erstellt haben, sind Sie bereit um GIS Daten in Ihre Datenbank zu laden. Aktuell existieren zwei Wege um Daten in die PostGIS/PostgreSQL Datenbank zu importieren: die Verwendung von formatierten SQL-Anweisungen oder des Shapefile Loader/Dumper.

4.4.1 Daten via SQL laden

Falls Sie Ihre Daten in eine Textdarstellung konvertieren können, dann ist möglicherweise die Verwendung von formatiertem SQL der leichteste Weg um Ihre Daten in PostGIS zu importieren. Wie bei Oracle und anderen Datenbanken, können die Daten über Masseneinsätze geladen werden, indem eine große Textdatei, in der sich lauter SQL "INSERT" Anweisungen befinden, an die SQL-Konsole weitergeleitet wird.

Eine Importdatei (z.B. `roads.sql`) könnte folgendermaßen aussehen:

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

Diese Datei kann dann über die "psql" SQL-Konsole sehr leicht nach PostgreSQL weitergeleitet werden:

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

4.4.2 shp2pgsql: Using the ESRI Shapefile Loader

Der `shp2pgsql` Datenlader wandelt ESRI Shapefiles in eine SQL-Datei um, die für das Einfügen in eine PostGIS/PostgreSQL Datenbank mit der "psql"-Konsole, sowohl im Geometrie- als auch im Geographie-Format, geeignet ist. Der Loader besitzt eine Reihe von Betriebsmodi, die durch Flags auf der Befehlszeile ausgewählt werden:

Zusätzlich zu dem befehszeilenorientierten Lader "shp2pgsql" gibt es auch die graphische Schnittstelle `shp2pgssql-gui`, welche fast ebensoviele Optionen wie der befehszeilenorientierte Lader zur Verfügung stellt und für viele nicht so Befehlszeilen-versierte bzw. PostGIS-Neuankömmlinge vielleicht einfacher zu bedienen ist. Es gibt auch ein Plugin davon für PgAdminIII.

(claldp) Dies sind sich gegenseitig ausschließende Optionen:

- c Erstellt eine neue Tabelle und füllt sie von einem Shapefile her. *Dies ist der Standardmodus.*
- a Fügt Daten aus dem Shapefile zu der Datenbanktabelle hinzu. Beachten Sie bitte, falls Sie diese Option verwenden um mehrere Dateien zu laden, dass die Attribute und Datentypen in den Dateien übereinstimmen müssen.
- d Löscht die Datenbanktabelle, bevor eine neue Tabelle mit den Daten vom Shapefile befüllt 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.
- s [**<FROM_SRID>**;**>**;**<SRID>**] Erzeugt und befüllt die Geometrietabelle in einer bestimmten SRID. Optional kann FROM_SRID für die Shapefile angegeben werden, wodurch die Geometrie von FROM_SRID in die Ziel-SRID projiziert wird. FROM_SRID und -D können nicht gleichzeitig angegeben werden.
- k Erhält die Groß- und Kleinschreibung (Spalte, Schema und Attribute). Beachten Sie bitte, dass die Attributnamen in Shapefile immer Großbuchstaben haben.
- i Wandeln Sie alle Ganzzahlen in standard 32-bit Integer um, erzeugen Sie keine 64-bit BigInteger, auch nicht dann wenn der DBF-Header dies unterstellt.
- I Einen GIST Index auf die Geometriespalte anlegen.
- m -m *a_file_name* bestimmt eine Datei, in welcher die Abbildungen der (langen) Spaltennamen in die 10 Zeichen langen DBF Spaltennamen festgelegt sind. Der Inhalt der Datei besteht aus einer oder mehreren Zeilen die jeweils zwei, durch Leerzeichen getrennte Namen enthalten, aber weder vorne noch hinten mit Leerzeichen versehen werden dürfen. Zum Beispiel:


```
COLUMNNAME DBFFIELD1
AVERYLONGCOLUMNNAME DBFFIELD2
```
- S Erzeugt eine Einzel- anstatt einer Mehrfachgeometrie. Ist nur erfolgversprechend, wenn die Geometrie auch tatsächlich eine Einzelgeometrie ist (insbesondere gilt das für ein Mehrfachpolygon/MULTIPOLYGON, dass nur aus einer einzelnen Begrenzung besteht, oder für einen Mehrfachpunkt/MULTIPOINT, der nur einen einzigen Knoten aufweist).
- t **<dimensionality>** Zwingt die Ausgabegeometrie eine bestimmte Dimension anzunehmen. Sie können die folgenden Zeichenfolgen verwenden, um die Dimensionalität anzugeben: 2D, 3DZ, 3DM, 4D.
Wenn die Eingabe weniger Dimensionen aufweist als angegeben, dann werden diese Dimensionen bei der Ausgabe mit Nullen gefüllt. Wenn die Eingabe mehr Dimensionen als angegeben aufweist werden diese abgestreift.
- w Ausgabe im Format WKT anstatt WKB. Beachten Sie bitte, dass es hierbei zu Koordinatenverschiebungen infolge von Genauigkeitsverlusten kommen kann.
- e Jede Anweisung einzeln und nicht in einer Transaktion ausführen. Dies erlaubt den Großteil auch dann zu laden, also die guten Daten, wenn eine Geometrie dabei ist die Fehler verursacht. Beachten Sie bitte das dies nicht gemeinsam mit der -D Flag angegeben werden kann, da das "dump" Format immer eine Transaktion verwendet.
- W **<encoding>** Gibt die Codierung der Eingabedaten (dbf-Datei) an. Wird die Option verwendet, so werden alle Attribute der dbf-Datei von der angegebenen Codierung nach UTF8 konvertiert. Die resultierende SQL-Ausgabe enthält dann den Befehl `SET CLIENT_ENCODING to UTF8`, damit das Back-end wiederum die Möglichkeit hat, von UTF8 in die, für die interne Nutzung konfigurierte Datenbankcodierung zu decodieren.
- N **<policy>** Umgang mit NULL-Geometrien (insert*, skip, abort)
- n -n Es wird nur die *.dbf-Datei importiert. Wenn das Shapefile nicht Ihren Daten entspricht, wird automatisch auf diesen Modus geschaltet und nur die *.dbf-Datei geladen. Daher müssen Sie diese Flag nur dann setzen, wenn sie einen vollständigen Shapefile-Satz haben und lediglich die Attributdaten, und nicht die Geometrie, laden wollen.

- G** Verwendung des geographischen Datentyps in WGS84 (SRID=4326), anstelle des geometrischen Datentyps (benötigt Längen- und Breitenangaben).
- T <tablespace>** Den Tablespace für die neue Tabelle festlegen. Solange der -X Parameter nicht angegeben wird, benutzen die Indizes weiterhin den standardmäßig festgelegten Tablespace. Die PostgreSQL Dokumentation beinhaltet eine gute Beschreibung, wann es sinnvoll ist, eigene Tablespaces zu verwenden.
- X <tablespace>** Den Tablespace bestimmen, in dem die neuen Tabellenindizes angelegt werden sollen. Gilt für den Primärschlüsselindex und wenn "-l" verwendet wird, auch für den räumlichen GIST-Index.

Eine beispielhafte Sitzung, in welcher der Loader verwendet wird, um eine Eingabedatei zu erzeugen und anschließend hochzuladen, könnte folgendermaßen aussehen:

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

Konvertierung und Import können über UNIX-Pipes in einem Schritt erfolgen:

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

4.5 Abfrage von GIS-Daten

Daten können entweder über SQL oder mit dem Shapefile Loader/Dumper aus der Datenbank geholt werden. Im Abschnitt über SQL werden wir einige Operatoren besprechen, die für Gegenüberstellungen und Abfragen von Geometrietabellen zur Verfügung stehen.

4.5.1 Daten via SQL abfragen

Das geradlinigste Mittel um Daten aus der Datenbank zu holen, besteht in der Anwendung einer SQL Select-Anfrage. Dadurch kann man die Anzahl der zurückgegebenen Datensätze und Spalten reduzieren und die resultierenden Datenbankspalten in eine lesbare Textdatei überspielen:

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

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

(6 rows)

Wie auch immer, manchmal wird eine Einschränkung notwendig sein, um die Anzahl der zurückgegebenen Werte zu reduzieren. Falls es sich um eine Beschränkung auf ein Attribut handelt, können Sie die selbe SQL-Syntax verwenden wie bei jeder anderen Nicht-Geometrietabelle. Für räumliche Beschränkungen sind folgende Operatoren verfügbar/nützlich:

&& Dieser Operator stellt fest, ob sich die Umgebungsrechtecke zweier Geometrien schneiden.

ST_OrderingEquals Überprüft, ob zwei Geometrien geometrisch identisch sind. Zum Beispiel, ob 'POLYGON((0 0,1 1,1 0,0 0))' identisch mit 'POLYGON((0 0,1 1,0 0,0 0))' ist (ist es).

= Dieser Operator stellt fest, ob die Umgebungsrechtecke zweier Geometrien identisch sind.

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 mit der Funktion "ST_GeomFromText()" in eine Geometrie umwandeln müssen. 312 ist ein fiktives Koordinatenreferenzsystem dass zu Unseren Daten passt. Also, zum Beispiel:

```
SELECT road_id, road_name
FROM roads
WHERE ST_OrderingEquals(roads_geom , ST_GeomFromText('LINESTRING(191232 243118,191108 ↵
243242)',312) ) ;
```

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

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.

```
SELECT road_id, road_name
FROM roads
WHERE roads_geom && ST_GeomFromText('POLYGON((...))',312);
```

Die obere Abfrage verwendet das Umgebungsrechteck des Polygons zu Vergleichszwecken.

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

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

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

4.5.2 Verwendung des Dumper

Der Tabellendumper `pgsql2shp` verbindet sich direkt mit der Datenbank und konvertiert eine Tabelle (evtl. durch eine Abfrage festgelegt) in eine Shapedatei. Die grundlegende Syntax lautet:

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

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

Optionen auf der Befehlszeile:

- f <filename>** Ausgabe in eine bestimmte Datei.
- h <host>** Der Datenbankserver, mit dem eine Verbindung aufgebaut werden soll.
- p <port>** Der Port über den der Verbindungsaufbau mit dem Datenbank Server hergestellt werden soll.
- P <password>** Das Passwort, das zum Verbindungsaufbau mit der Datenbank verwendet werden soll.
- u <user>** Das Benutzernamen, 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** Verwendung eines "binären Cursors"/Iterators. Macht die Berechnung schneller, funktioniert aber nicht wenn irgendein geometrielooses Attribut nicht in den Typ "text" umgewandelt werden kann.
- r** RAW-Modus. Das Attribut `gid` wird nicht verworfen und Spaltennamen werden nicht maskiert.

- d Für Abwärtskompatibilität: schreiben Sie eine 3-dimensionale Shapedatei, wenn Sie aus alten (pre-1.0.0) PostGIS Datenbanken exportieren (standardmäßig würde in diesem Fall eine 2-dimensionale Shapedatei herausgeschrieben). Ab PostGIS-1.0.0+ wird die Dimension vollständig codiert.
- m **filename** Bildet die Identifikatoren in Namen mit 10 Zeichen ab. Der Inhalt der Datei besteht aus Zeilen von jeweils zwei leerzeichengetrennten Symbolen, jedoch ohne vor- oder nachgestellte Leerzeichen: VERYLONGSYMBOL SHORTONE ANOTHERVERYLONGSYMBOL SHORTER etc.

4.6 Indizes aufbauen

Indizes ermöglichen erst das Arbeiten mit großen Datensätzen in einer Geodatenbank. Ohne Indizierung würde jede Featureanfrage einen "Full Table Scan" in der Datenbank benötigen. Die Indizierung beschleunigt die Suche, indem die Daten in einem Suchbaum strukturiert werden, der dann schnell durchlaufen werden kann, um einen bestimmten Datensatz zu finden. PostgreSQL unterstützt standardmäßig drei Arten von Indizes: B-Baum, R-Baum und GIST Indizes.

- Ein B-Baum wird verwendet, wenn die Daten entlang einer Achse sortiert werden können; zum Beispiel Zahlen, Buchstaben und Datum. Geodaten können nicht vernünftig entlang einer Achse sortiert werden (was ist größer - (0,0), (0,1) oder (1,0)?), weshalb die Indizierung mittels B-Baum für Uns hier keine Verwendung findet.
- R-Bäume unterteilen die Daten in Rechtecke, die wiederum in Rechtecke unterteilt werden, usw.. R-Bäume werden von einigen Geodatenbanken zur Indizierung von Geodaten verwendet. Bei PostgreSQL ist die der R-Baum allerdings nicht so robust umgesetzt wie die GIST Implementierung.
- GiST (Generalized Search Tree) Indizes unterteilen die Daten in "Dinge auf einer Seite", "Dinge die sich überlagern", "Dinge die innerhalb liegen". Sie können auf eine Vielzahl von Datentypen, inklusive Geodaten, angewendet werden. PostGIS verwendet einen R-Baum der auf dem GIST Index aufgesetzt wurde, um Geodaten zu indizieren.

4.6.1 GiST-Indizes

GIST, "Generalized Search Tree", steht für eine generische Datenstruktur. Zusätzlich zu der indizierung von GIS-Daten, wird GIST zur schnelleren Abfrage aller möglichen unregelmäßigen Datenstrukturen (Ganzzahl-Felder, Spektraldaten, etc.), welche über gewöhnlicher B-Baum Indizierung nicht zugänglich sind.

Sobald eine Geodatentabelle ein paar tausend Zeilen überschreitet, werden Sie einen Index erzeugen wollen, um die räumlichen Abfragen auf die Daten zu beschleunigen (außer Ihre Suche basiert lediglich auf Attributen, in diesem Fall werden Sie einen gewöhnlichen Index auf die Attribute setzen).

Die Syntax, mit der ein GIST-Index auf eine Geometriespalte angelegt wird, lautet wie folgt:

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

Die obere Syntax erzeugt immer einen 2D-Index. Um einen n-dimensionalen Index zu erhalten, welcher ab PostGIS 2.0+ für den geometrischen Datentyp unterstützt wird, können Sie die folgende Syntax verwenden:

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

Die Erstellung eines räumlichen Indexes ist eine rechenintensive Aufgabe: Bei Tabellen mit um die 1 Millionen Zeilen und auf einer 300MHz Solaris, haben wir festgestellt, dass der Aufbau des GIST-Index ca. 1 Stunde dauerte. Nachdem der Index aufgebaut ist, sollte man unbedingt die Tabellenstatistik von PostgreSQL erneuern; diese wird benutzt um den Auswertungsplan/"Query Plan" zu optimieren:

```
VACUUM ANALYZE [table_name] [(column_name)];
-- Dies wird nur für Installationen von PostgreSQL 7.4, oder niedriger, benötigt
SELECT UPDATE_GEOMETRY_STATS([table_name], [column_name]);
```

Gegenüber R-Baum Indizes haben GIST-Indizes in PostgreSQL zwei Vorteile. Erstens: GIST-Indizes sind "NULL safe", d.h. Spalten die NULL- Werte enthalten können indiziert werden. GIST-Indizes sind verlustfrei/"Concept of lossiness", was bei GIS-Objekten wichtig ist, die größer als die 8k Speicherseitengröße/"Page Size" von PostgreSQL sind. Diese Verlustfreiheit ermöglicht es PostgreSQL nur den "wichtigen" Teil eines Objektes in einem Index zu speichern -- bei GIS Objekten das Umgebungsrechteck. GIS-Objekte, die größer als 8K sind, können mit R-Baum-Indizes nicht erstellt werden.

4.6.2 BRIN Indizes

Die Bezeichnung BRIN steht für "Block Range Index", eine generische Form des Indizierens und wurde mit PostgreSQL 9.5 eingeführt. BRIN ist ein verlustbehafteter Index, dessen Hauptzweck ist, einen Kompromiss sowohl bei der Lese- als auch bei der Schreibgeschwindigkeit anzubieten. Der Hauptverwendungszweck liegt bei sehr großen Tabellen, in denen einige Spalten einen natürlichen Bezug zu dem physischen Speicherplatz innerhalb der Tabelle haben. Zusätzlich zur Indizierung von GIS-Daten, werden BRIN-Indizes zur Beschleunigung von Suchabfragen auf unterschiedliche regelmäßige und unregelmäßige Datenstrukturen (Ganzzahlen, Felder etc.) verwendet.

Sobald eine Geodatentabelle ein paar tausend Zeilen überschreitet, werden Sie einen Index erzeugen wollen, um die räumlichen Abfragen auf die Daten zu beschleunigen (außer Ihre Suche basiert lediglich auf Attributen, in diesem Fall werden Sie einen gewöhnlichen Index auf die Attribute setzen). GIST Indizes sind sehr performant, solange ihre Dateigröße den verfügbaren Arbeitsspeicher der Datenbank nicht überschreitet, genügend Festplattenspeicher vorhanden ist und die Systembelastung durch Schreibvorgänge akzeptiert werden kann. Andernfalls bietet der BRIN Index eine Alternative.

Die Idee hinter einem BRIN-Index ist, dass nur das Umgebungsrechteck abgespeichert wird, dass die gesamte Geometrie eines oder mehrerer Tabellenblöcke umschließt; dies wird als "Range" bezeichnet. Es ist klar, dass diese Indizierungsmethode nur dann effizient sein kann, wenn die Daten physikalisch so angeordnet sind, dass sich die resultierenden Umgebungsrechtecke der "Block Ranges" gegenseitig ausschließen. Der resultierende Index ist zwar sehr klein, in vielen Fällen allerdings weniger effizient als ein GIST Index.

Die Erstellung eines BRIN-Index benötigt wesentlich weniger Zeit, als die Erstellung eines GIST-Index. Es ist durchaus üblich, dass die Erstellung des BRIN Index mehr als zehnmal so schnell ist, als die eines GIST Index. Da ein BRIN Index nur ein Umgebungsrechteck für einen oder mehrere Tabellenblöcke speichert, benötigt dieser oft bis zu tausendmal weniger Festplattenspeicher.

Sie können die Anzahl der Blöcke festlegen, die zu einen "Range" aufsummiert werden sollen. Wenn Sie die Anzahl verringern, wird der Index zwar größer, höchstwahrscheinlich aber zu einer besseren Performanz verhelfen.

Der Syntax zur Erstellung eines BRIN-Indizes auf eine geometrische Spalte lautet wie folgt:

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

Die obere Syntax erzeugt immer einen 2D-Index. Um einen 3-dimensionalen Index zu erhalten, können Sie die folgende Syntax verwenden:

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

Sie können auch einen 4-dimensionalen Index über die 4D-Operatorklasse erstellen

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

Die oberen Syntaxen verwenden die Standardeinstellung für die Anzahl der Blöcke in einem "Range", nämlich 128. Wenn Sie die Anzahl der Blöcke, die in einem Range zusammengefasst werden sollen, selbst festlegen wollen, verwenden Sie bitte die folgende Syntax

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

Beachten Sie bitte auch, dass ein BRIN Index nur einen Indexwert für eine große Anzahl von Zeilen speichert. Wenn Ihre Tabelle eine Geometrie mit unterschiedlichen Dimensionen speichert, dann ist es wahrscheinlich dass der Index eine schlechte Performanz aufweist. Sie können diesen Performanzrückgang vermeiden, indem Sie die Operatorklasse mit der niedrigsten Dimension der gespeicherten Geometrie wählen.

Der BRIN-Index wird auch vom geographischen Datentyp unterstützt. Die Syntax zur Erstellung eines BRIN-Index auf eine "geographische" Spalte lautet wie folgt:

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

Der obere Syntax erzeugt den 2D-Index für Geoobjekte immer auf dem Referenzellipsoid.

Aktuell wird hierbei nur die "Inklusionsunterstützung" betrachtet; d.h. dass nur die Operatoren &&, ~ und @ für 2D (sowohl für den "geometrischen", als auch für den "geographischen" Datentyp), und nur der Operator &&& für 3D-Geometrie verwendet werden kann. Die kNN-Suche wird zur Zeit nicht unterstützt.

```
VACUUM ANALYZE [table_name] [(column_name)];
-- Dies wird nur für Installationen von PostgreSQL 7.4, oder niedriger, benötigt
SELECT UPDATE_GEOMETRY_STATS([table_name], [column_name]);
```

4.6.3 Verwendung von Indizes

Üblicherweise beschleunigen Indizes den Datenzugriff: Sobald der Index aufgebaut ist, entscheidet der Anfrageoptimierer ob der Index den Auswertungsplan beschleunigt. Unglücklicherweise optimiert der Anfrageoptimierer von PostgreSQL die Verwendung von GIST Indizes nicht sehr gut, so dass manchmal eine Suche, welche die Verwendung eines räumlichen Index bedingen sollte, über einen Full Table Scan ausgeführt wird.

Wenn Sie bemerken, dass Ihre räumlichen Indizes (oder Ihre Attributindizes) nicht verwendet werden, können Sie eine Reihe von Dingen unternehmen:

- Zuerst stellen Sie bitte sicher, dass eine Statistik über die Anzahl und die Verteilung der Werte in der Tabelle erfasst wurde. Dadurch wird dem Anfrageoptimierer bessere Information zur Verfügung gestellt, um die Entscheidungen bezüglich der Anwendung des Index zu treffen. Bei PostgreSQL 7.4 und niedriger wird dies durch **update_geometry_stats([table_name], column_name)** (compute distribution) und **VACUUM ANALYZE [table_name] [column_name]** (compute number of values).
- Wenn VACUUM nicht funktioniert, können Sie den Anfrageoptimierer dazu zwingen den Index zu verwenden, indem Sie den Befehl **SET ENABLE_SEQSCAN=OFF** ausführen. Sie sollten diesen Befehl sehr spärlich und nur für räumlich indizierte Anfragen verwenden: d.h. der Anfrageoptimierer weiß besser Bescheid als Sie, ob ein gewöhnlicher B-Baum Index verwendet werden soll oder nicht. Sobald Sie Ihre Abfrage ausgeführt haben sollten Sie daher **ENABLE_SEQSCAN** wieder auf ON stellen, damit weitere Abfragen den Anfrageoptimierer wie üblich nutzen können.



Note

Ab Version 0.6 sollte es nicht mehr notwendig sein, den Anfrageoptimierer mittels **ENABLE_SEQSCAN** zur Anwendung des Index zu zwingen.

- Wenn Sie herausfinden, daß sich der Anfrageoptimierer bezüglich der Kosten des Full Table Scan im Verhältnis zum Index Scan irrt, können Sie versuchen den Wert von "random_page_cost" in "postgresql.conf" zu reduzieren, oder **SET random_page_cost=#** ausführen. Der Standardwert des Parameters ist 4, versuchen Sie ihn auf 1 oder 2 zu setzen. Eine Verringerung des Wertes führt dazu, dass der Anfrageoptimierer eher den Index Scan verwendet.

4.7 Komplexe Abfragen

Sinn und Zweck der Geodatenbankfunktionalität ist es, Abfragen innerhalb der Datenbank auszuführen, welche üblicherweise die Funktionalität eines Desktop-GIS benötigen würden. Um PostGIS effizient zu nutzen, müssen Sie die verfügbaren räumlichen Funktionen kennen und sicherstellen, dass die geeigneten Indizes vorhanden sind um eine gute Performanz zu gewährleisten. Die SRID von 312, die in diesen Beispielen verwendet wird, ist für bloße Demonstrationszwecke gedacht. Sie sollten eine ECHTE SRID aus der Tabelle "spatial_ref_sys" verwenden, die auch mit der Projektion Ihrer Daten übereinstimmen muss. Falls Ihren Daten kein Koordinatenreferenzsystem zugewiesen ist, sollten Sie genau eruieren warum dies so ist.

Wenn der Grund darin liegt, dass Sie irgendetwas modellieren, für das kein Koordinatenreferenzsystem festgelegt ist, wie der innere Aufbau eines Moleküls oder der Grundriss eines noch nicht gebauten Vergnügungsparks, so ist dies in Ordnung. Wenn der Standort des Vergnügungsparks bereits geplant wurde, dann ist die Wahl eines geeigneten Koordinatenreferenzsystems sinnvoll, auch wenn es nur darum geht sicherzustellen, dass der Vergnügungspark keine bereits bestehenden Strukturen überdeckt.

Sogar wenn Sie eine Mars Expedition planen, um die menschliche Rasse nach einem nuklearen Holocaust zu transportieren und Sie den Planeten Mars für die Besiedelung kartieren wollen, so können Sie ein Koordinatenreferenzsystem wie **Mars 2000** erstellen und dieses in die Tabelle `spatial_ref_sys` einfügen. Obwohl diese Koordinatensystem für den Mars nicht planar ist (es ist in Grad des Referenzellipsoids), können Sie es mit dem geographischen Datentyp nutzen, um Längen- und Abstandsmessungen in Meter anstatt in Grad anzuzeigen.

4.7.1 Indizes nutzen

Wenn Sie eine Abfrage erstellen müssen Sie beachten, dass nur die auf den Umgebungsrechtecken basierenden Operatoren wie `&&` die Vorteile eines räumlichen GIST Index ausnutzen können. Funktionen wie `ST_Distance()` können den Index nicht zur Optimierung heranziehen. Zum Beispiel würde die folgende Abfrage auf eine große Tabelle ziemlich langsam ablaufen:

```
SELECT the_geom
FROM geom_table
WHERE ST_Distance(the_geom, ST_GeomFromText('POINT(100000 200000)', 312)) < 100
```

Diese Abfrage wählt all jene Geometrie der Tabelle "geom_table" aus, die weniger als 100 Einheiten von dem Punkt (100000, 200000) entfernt liegt. Sie ist sehr langsam, da die Entfernung zwischen jedem Punkt in der Tabelle und Unserem angegebenen Punkt berechnet werden muss, d.h. eine `ST_Distance()` Berechnung pro Tabellenzeile. Wir können dies vermeiden, indem wir den Operator `"&&"` verwenden und so die Anzahl der benötigten Entfernungsberechnungen verringern:

```
SELECT the_geom
FROM geom_table
WHERE ST_DWithin(the_geom, ST_MakeEnvelope(90900, 190900, 100100, 200100, 312), 100)
```

Diese Anfrage wählt dieselbe Geometrie aus, allerdings auf effizientere Weise. Angenommen es existiert ein GIST Index auf der Spalte "the_geom" und der Anfrageoptimierer erkennt, dass der Index angewendet werden kann, um die Zeilenanzahl zu verringern, bevor das Ergebnis durch die Funktion `ST_Distance()` errechnet wird. Anmerkung: die Geometrie `ST_MakeEnvelope`, die vom Operator `"&&"` verwendet wird, ist ein Quadrat mit einer Seitenlänge von 200 Einheiten, dessen Mittelpunkt auf dem ursprünglichen Punkt liegt - dies ist Unsere "Abfrage Box". Der Operator `"&&"` verwendet diesen Index, um die Ergebnismenge rasch auf die Geometrie zu reduzieren, deren Umgebungsrechtecke die "Abfrage Box" überlagern. Falls Unsere "Abfrage Box" wesentlich kleiner als die Gesamtausdehnung der gesamten Geometrietabelle ist, wird dadurch die Anzahl der Entfernungsberechnungen drastisch verringert- dies ist genau das, was wir wollen.



Änderung der Verhaltensweise

Ab PostGIS 1.3.0 implizieren die meisten geometrischen Relationsfunktionen die Verwendung von Überlappungsoperatoren auf die Umgebungsrechtecke. Erwähnenswerte Ausnahmen sind `ST_Disjoint` und `ST_Relate`.

4.7.2 Beispiele für Spatial SQL

Die Beispiele in diesem Abschnitt verwenden zwei Tabellen, eine Tabelle mit linearen Strassen, und eine Tabelle mit polygonalen Verwaltungsgrenzen. Die Tabellendefinition der Tabelle `bc_roads` lautet:

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

Die Tabellendefinition für die Tabelle `bc_municipality`:

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

1. Gesamtlänge aller Straßen in Kilometer?

Sie können diese Frage mit einer sehr einfachen SQL Anweisung beantworten:

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

km_roads
70842.1243039643

(1 row)

2. Wieviele Hektar hat die Stadt Prince George?

Diese Abfrage kombiniert eine Attributbedingung (auf den Gemeindenamen) mit einer räumlichen Berechnung (der Fläche):

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

hectares
32657.9103824927

(1 row)

3. Welche ist die flächengrößte Gemeinde der Provinz?

Diese Abfrage verwendet eine räumliche Messung als Abfragefilter. Es gibt verschiedene Wege, um diese Problem anzugehen, aber die effizienteste Methode ist folgende:

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

name	hectares
TUMBLER RIDGE	155020.02556131

(1 row)

Um diese Anfrage zu beantworten müssen wir die Fläche eines jeden Polygons berechnen. Wenn wir dies oft machen müssen, kann es aufgrund der Rechenleistung sinnvoll sein, eine eigene Flächenspalte an die Tabelle anzuhängen und mit einem Index zu versehen. Indem wir das Ergebnis in absteigender Reihenfolge sortieren und den PostgreSQL Befehl "LIMIT" einsetzen, können wir die größten Werte herausfiltern, ohne eine Aggregatfunktion wie max() verwenden zu müssen.

4. Welche Länge haben die Straßen, die zur Gänze innerhalb einer Gemeinde liegen?

Dies ist ein Beispiel für einen "Spatial Join", da wir die Daten aus zwei Tabellen zusammenführen (einen Join ausführen) und als Join-Bedingung eine räumliche Interaktion ("contained") verwenden - anstelle des üblichen relationalen Ansatzes bei dem die Tabellen über einen gemeinsamen Schlüssel verknüpft werden:

```
SELECT
  m.name,
  sum(ST_Length(r.the_geom))/1000 as roads_km
FROM
  bc_roads AS r,
  bc_municipality AS m
WHERE
  ST_Contains(m.the_geom,r.the_geom)
GROUP BY m.name
```

```
ORDER BY roads_km;
```

name	roads_km
SURREY	1539.47553551242
VANCOUVER	1450.33093486576
LANGLEY DISTRICT	833.793392535662
BURNABY	773.769091404338
PRINCE GEORGE	694.37554369147
...	

Diese Abfrage dauert ein Weilchen, da sämtliche Straßen in der Tabelle in das endgültige Ergebnis aufsummiert werden müssen (über 250k Straßen in Unserem speziellen Beispiel). Bei kleineren Überlagerungen (ein paar tausend Datensätze auf ein paar Hundert) kann die Antwort sehr schnell zurückkommen.

5. Eine neue Tabelle erzeugen, die alle Straßen der Stadt Prince George beinhaltet.

Dies ist ein Beispiel für ein "Overlay", das zwei Tabellen entgegennimmt und eine neue Tabelle ausgibt, welche die aus- und abgeschnittene Ergebnisgeometrie enthält. Anders als bei dem oben gezeigten "Spatial Join" erzeugt diese Abfrage eine neue Geometrie. Ein "Overlay" ist wie ein "Spatial Join" mit Turbolader und wird für genauere Analysen verwendet:

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

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

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

kilometers
4.89151904172838

(1 row)

7. Welches ist das größte Gemeindepolygon mit einer Lücke?

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

gid	name	area
12	SPALLUMCHEEN	257374619.430216

(1 row)

Chapter 5

Rasterdaten: Verwaltung, Abfrage und Anwendung

5.1 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.

5.1.1 Verwendung von `raster2pgsql` zum Laden von Rastern

`raster2pgsql` ist ein ausführbarer Rasterlader, der die von GDAL unterstützten Rasterformate in SQL umwandelt um sie anschließend in eine PostGIS Rastertabelle zu laden. Er kann sowohl ganze Verzeichnisse mit Rasterdateien laden, als auch Rasterübersichten erzeugen.

Da "raster2pgsql" meistens als Teil von PostGIS kompiliert ist (solange Sie nicht Ihre eigene GDAL Bibliothek kompilieren), sind die von "raster2pgsql" unterstützten Rastertypen die selben wie in der GDAL Bibliothek. Um eine Liste der Rastertypen, die von Ihrem jeweiligen "raster2pgsql" unterstützt werden, zu erhalten, benutzen Sie bitte den `-G` Switch. Falls Sie dieselbe GDAL Bibliothek für beide verwenden, sollte diese Liste mit der Ihrer PostGIS Installation, die durch `ST_GDALDrivers` bereitgestellt wird, ident sein.

**Note**

Die frühere Version dieses Tools war ein Python Skript. Die lauffähige Version hat das Python Skript ersetzt. Sollten Sie weiterhin das Python Skript benötigen, können Sie unter [GDAL PostGIS Raster Driver Usage](#) Beispiele für Python finden. Beachten Sie bitte, dass zukünftige Versionen von PostGIS Raster das "raster2pgsql" Pythonskript nicht mehr unterstützen.

**Note**

Bei einem bestimmten Faktor kann es vorkommen, dass die Raster in der Übersicht/Overview nicht bündig angeordnet sind, obwohl sie die Raster selbst dies sind. Siehe <http://trac.osgeo.org/postgis/ticket/1764> für ein solches Beispiel.

ANWENDUNGSBEISPIEL:

```
raster2pgsql raster_options_go_here raster_file someschema.sometable > out.sql
```

`-?` Zeigt die Hilfe an, auch dann, wenn keine Argumente übergeben werden.

-G Gibt die unterstützten Rasterformate aus.

(claldlp) 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 bestehenden Tabelle hinzufügen.
- d** Tabelle löschen, eine Neu erzeugen und mit einem oder mehreren Raster befüllen
- p** Beim vorbereitenden Modus wird lediglich eine Tabelle erstellt.

Raster-Verarbeitung: Anwendung von Constraint's zur ordnungsgemäßen Registrierung im Rasterkatalog

- C** Anwendung von Raster-Constraints, wie SRID, Zellgröße etc., um die ordnungsgemäße Registrierung des Rasters in der `raster_columns` View sicherzustellen.
- x** Unterbindet das Setzen der "Max-Extent" Bedingung. Wird nur angewandt, wenn auch die **-C** Flag gesetzt ist.
- r** Setzt die Constraints (räumlich eindeutig und die Coverage-Kachel) der regelmäßigen Blöcke. Wird nur angewandt, wenn auch die **-C** Flag gesetzt ist.

Rasterdaten-Verarbeitung: Optionale Parameter zur Manipulation von Input Raster Datensätzen

- s <SRID>** Dem Output-Raster eine bestimmte SRID zuweisen. Wenn keine SRID oder Null angegeben wird, werden die Raster-Metadaten auf eine geeignete SRID hin überprüft.
- b BAND** Die Kennung (1-basiert) des Bandes, das aus dem Raster entnommen werden soll. Um mehrere Bänder anzugeben, trennen Sie die Kennungen bitte durch ein Komma (,).
- t TILE_SIZE** Zerlegt den Raster in Kacheln, um eine Kachel pro Tabellenzeile einzufügen. `TILE_SIZE` wird entweder in `BREITExHöhe` ausgedrückt, oder auf den Wert "auto" gesetzt, wodurch der Raster-Lader eine passende Kachelgröße an Hand des ersten Raster's ermittelt und diese dann auf die anderen Raster anwendet.
- P** Die ganz rechts und ganz unten liegenden Kacheln aufstocken, damit für alle Kacheln gleiche Breite und Höhe sichergestellt ist.
- R, --register** Einen im Dateisystem vorliegenden Raster als (out-db) Raster registrieren.
Es werden nur die Metadaten und der Dateipfad des Rasters abgespeichert (nicht die Rasterzellen).
- l OVERVIEW_FACTOR** Erzeugt eine Übersicht/Overview des Rasters. Mehrere Faktoren sind durch einen Beistrich(,) zu trennen. Die Benennung der Übersichtstabelle erfolgt dem Muster `o_overview_factor_table`, wobei `overview_factor` ein Platzhalter für den numerischen Wert von "overview_factor" ist und `table` für den zugrundeliegenden Tabellennamen. Die erstellte Übersicht wird in der Datenbank gespeichert, auch wenn die Option **-R** gesetzt ist. Anmerkung: die erzeugte SQL-Datei enthält sowohl die Haupttabelle, als auch die Übersichtstabellen.
- N NODATA** Der NODATA-Wert, der für Bänder verwendet wird, die keinen NODATA-Wert definiert haben.

Optionale Parameter zur Manipulation von Datenbankobjekten

- q** Setzt die PostgreSQL-Identifikatoren unter Anführungszeichen
- f COLUMN** Gibt den Spaltennamen des Zielrasters an; standardmäßig wird er 'rast' benannt.
- F** Eine Spalte mit dem Dateinamen hinzufügen
- n COLUMN** Gibt die Bezeichnung für die Spalte mit dem Dateinamen an. Schließt **-F** mit ein.
- q** Setzt die PostgreSQL-Identifikatoren unter Anführungszeichen.
- I** Einen GIST-Index auf die Rasterspalte anlegen.
- M VACUUM ANALYZE** auf die Rastertabelle.
- k** Überspringt die Überprüfung von NODATA-Werten für jedes Rasterband.
- T tablespace** Bestimmt den Tablespace für die neue Tabelle. Beachten Sie bitte, dass Indizes (einschließlich des Primärschlüssels) weiterhin den standardmäßigen Tablespace nutzen, solange nicht die **-X** Flag benutzt wird.
- X tablespace** Bestimmt den Tablespace für den neuen Index der Tabelle. Dieser gilt sowohl für den Primärschlüssel als auch für den räumlichen Index, falls die **-I** Flag gesetzt ist.
- Y** Verwendung von Kopier- anstelle von Eingabe-Anweisungen.

- e Keine Transaktion verwenden, sondern jede Anweisung einzeln ausführen.
- E **ENDIAN** Legt die Byte-Reihenfolge des binär erstellten Rasters fest; geben Sie für XDR 0 und für NDR (Standardwert) 1 an; zurzeit wird nur die Ausgabe von NDR unterstützt.
- V **version** Bestimmt die Version des Ausgabeformats. Voreingestellt ist 0. Zur Zeit wird auch nur 0 unterstützt.

Eine Beispielssitzung, wo mit dem Lader eine Eingabedatei erstellt und stückchenweise als 100x100 Kacheln hochgeladen wird, könnte so aussehen:



Note

Sie können den Schemanamen weglassen z.B. `demelevation` anstatt `public.demelevation`, wodurch die Rastertabelle im Standardschema der Datenbank oder des Anwenders angelegt wird.

```
raster2pgsql -s 4326 -I -C -M *.tif -F -t 100x100 public.demelevation
> elev.sql
psql -d gisdb -f elev.sql
```

Durch die Verwendung von UNIX-Pipes kann die Konvertierung und der Upload in einem Schritt vollzogen werden:

```
raster2pgsql -s 4326 -I -C -M *.tif -F -t 100x100 public.demelevation | psql -d gisdb
```

Luftbildkacheln in "Massachusetts State Plane Meters" in das Schema `aerial` laden. Einen vollständigen View und Übersichtstabellen mit Faktor 2 und 4 erstellen. Verwendet den Modus "copy" für das Insert (keine dazwischengeschaltete Datei, sondern direkt in die Datenbank). Die Option -e bedingt, dass nicht alles innerhalb einer Transaktion abläuft (nützlich, wenn Sie sofort Daten sehen wollen, ohne zu warten). Die Raster werden in 128x128 Pixel große Kacheln zerlegt und Constraints auf die Raster gesetzt. Verwendet den Modus "copy" anstelle eines Tabellen-Inserts. (-F) Erzeugt das Attribut "filename", welches die Bezeichnung der Ausgangsdateien enthält, aus denen die Rasterkacheln ausgeschnitten wurden.

```
raster2pgsql -I -C -e -Y -F -s 26986 -t 128x128 -l 2,4 bostonaerials2008/*.jpg aerials. ↵
boston | psql -U postgres -d gisdb -h localhost -p 5432
```

```
--gibt eine Liste der unterstützten Rasterformate aus:
raster2pgsql -G
```

Der -G Befehl gibt eine ähnliche Liste wie die Folgende aus

```
Available GDAL raster formats:
Virtual Raster
GeoTIFF
National Imagery Transmission Format
Raster Product Format TOC format
ECRG TOC format
Erdas Imagine Images (.img)
CEOS SAR Image
CEOS Image
JAXA PALSAR Product Reader (Level 1.1/1.5)
Ground-based SAR Applications Testbed File Format (.gff)
ELAS
Arc/Info Binary Grid
Arc/Info ASCII Grid
GRASS ASCII Grid
SDTS Raster
DTED Elevation Raster
Portable Network Graphics
JPEG JFIF
In Memory Raster
Japanese DEM (.mem)
```


Graphics Interchange Format (.gif)
Graphics Interchange Format (.gif)
Envisat Image Format
Maptech BSB Nautical Charts
X11 PixMap Format
MS Windows Device Independent Bitmap
SPOT DIMAP
AirSAR Polarimetric Image
RadarSat 2 XML Product
PCIDSK Database File
PCRaster Raster File
ILWIS Raster Map
SGI Image File Format 1.0
SRTMHGT File Format
Leveller heightfield
Terragen heightfield
USGS Astrogeology ISIS cube (Version 3)
USGS Astrogeology ISIS cube (Version 2)
NASA Planetary Data System
EarthWatch .TIL
ERMapper .ers Labelled
NOAA Polar Orbiter Level 1b Data Set
FIT Image
GRIdDED Binary (.grb)
Raster Matrix Format
EUMETSAT Archive native (.nat)
Idrisi Raster A.1
Intergraph Raster
Golden Software ASCII Grid (.grd)
Golden Software Binary Grid (.grd)
Golden Software 7 Binary Grid (.grd)
COSAR Annotated Binary Matrix (TerraSAR-X)
TerraSAR-X Product
DRDC COASP SAR Processor Raster
R Object Data Store
Portable Pixmap Format (netpbm)
USGS DOQ (Old Style)
USGS DOQ (New Style)
ENVI .hdr Labelled
ESRI .hdr Labelled
Generic Binary (.hdr Labelled)
PCI .aux Labelled
Vexcel MFF Raster
Vexcel MFF2 (HKV) Raster
Fuji BAS Scanner Image
GSC Geogrid
EOSAT FAST Format
VTP .bt (Binary Terrain) 1.3 Format
Erdas .LAN/.GIS
Convair PolGASP
Image Data and Analysis
NLAPS Data Format
Erdas Imagine Raw
DIPEx
FARSITE v.4 Landscape File (.lcp)
NOAA Vertical Datum .GTX
NADCON .los/.las Datum Grid Shift
NTv2 Datum Grid Shift
ACE2
Snow Data Assimilation System
Swedish Grid RIK (.rik)
USGS Optional ASCII DEM (and CDED)

```

GeoSoft Grid Exchange Format
Northwood Numeric Grid Format .grd/.tab
Northwood Classified Grid Format .grc/.tab
ARC Digitized Raster Graphics
Standard Raster Product (ASRP/USRP)
Magellan topo (.blx)
SAGA GIS Binary Grid (.sdatt)
Kml Super Overlay
ASCII Gridded XYZ
HF2/HFZ heightfield raster
OziExplorer Image File
USGS LULC Composite Theme Grid
Arc/Info Export E00 GRID
ZMap Plus Grid
NOAA NGS Geoid Height Grids

```

5.1.2 Erzeugung von Rastern mit den PostGIS Rasterfunktionen

Oftmals werden Sie die Raster und die Rastertabellen direkt in der Datenbank erzeugen wollen. Dafür existieren eine Unmenge an Funktionen. Dies verlangt im Allgemeinen die folgende Schritte.

1. Erstellung einer Tabelle mit einer Rasterspalte für die neuen Rasterdatensätze:

```
CREATE TABLE myrasters(rid serial primary key, rast raster);
```

2. Es existieren viele Funktionen die Ihnen helfen dieses Ziel zu erreichen. Wenn Sie einen Raster nicht von anderen Rastern ableiten, sondern selbst erzeugen, können Sie mit **ST_MakeEmptyRaster** beginnen, gefolgt von **ST_AddBand**. Sie können Raster auch aus Geometrien erzeugen. Hierzu können Sie **ST_AsRaster** verwenden, möglicherweise in Verbindung mit anderen Funktionen, wie **ST_Union**, **ST_MapAlgebraFct** oder irgendeiner anderen Map Algebra Funktion.

Es gibt sogar noch viele andere Möglichkeiten, um eine neue Rastertabelle aus bestehenden Tabellen zu erzeugen. Sie können zum Beispiel mit **ST_Transform** einen Raster in eine andere Projektion transformieren und so eine neue Rastertabelle erstellen.

3. Wenn Sie mit der Erstbefüllung der Tabelle fertig sind, werden Sie einen räumlichen Index auf die Rasterspalte setzen wollen:

```
CREATE INDEX myrasters_rast_st_convexhull_idx ON myrasters USING gist( ST_ConvexHull( ↵
rast) );
```

Beachten Sie bitte die Verwendung von **ST_ConvexHull**; der Grund dafür ist, dass die meisten Rasteroperatoren auf der konvexen Hülle des Rasters beruhen.



Note

Vor der Version 2.0 von PostGIS, basierten die Raster auf der Einhüllenden, anstatt auf der konvexen Hülle. Damit die räumlichen Indizes korrekt funktionieren, müssen Sie diese löschen und mit einem auf der konvexen Hülle basierenden Index ersetzen.

4. Mittels **AddRasterConstraints** Bedingungen auf den Raster legen.

5.2 Raster Katalog

Mit PostGIS kommen zwei Views des Rasterkatalogs. Beide Views nutzen 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.

5.2.1 Rasterspalten Katalog

`raster_columns` ist ein Katalog mit allen Rasterspalten Ihrer Datenbanktabellen. Es handelt sich dabei um einen View, der die Constraints auf die Tabellen ausnutzt, um so immer konsistent mit dem aktuellen Stand der Datenbank zu bleiben; sogar dann, wenn Sie den Raster aus einem Backup oder einer anderen Datenbank wiederherstellen. Der `raster_columns` Katalog beinhaltet die folgenden Spalten.

Falls Sie Ihre Tabellen nicht mit dem Loader erstellt haben, oder vergessen haben, die `-C` Option während des Ladens anzugeben, können Sie die Constraints auch anschließend erzwingen, indem Sie `AddRasterConstraints` verwenden, wodurch der `raster_columns` Katalog die Information über Ihre Rasterkacheln, wie üblich abspeichert.

- `r_table_catalog` Die Datenbank, in der sich die Tabelle befindet. Greift immer auf die aktuelle Datenbank zu.
- `r_table_schema` Das Datenbankschema in dem sich die Rastertabelle befindet.
- `r_table_name` Rastertabelle
- `r_raster_column` Die Spalte, in der Tabelle `r_table_name`, die den Datentyp Raster aufweist. In PostGIS gibt es nichts, was Sie daran hindert, mehrere Rasterspalten in einer Tabelle zu haben. Somit ist es möglich auf unterschiedliche Raster(spalten) in einer einzigen Rastertabelle zuzugreifen.
- `srid` Der Identifikator für das Koordinatensystem in dem der Raster vorliegt. Sollte in Section 4.3.1 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. Die selbe Information, die `ST_NumBands` liefert. `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 wird. 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.

5.2.2 Raster Übersicht/Raster Overviews

`raster_overviews` Katalogisiert Information über die Rastertabellenspalten die für die Übersichten/Overviews herangezogen wurden, sowie weitere zusätzliche Information bezüglich Overviews. Die Übersichtstabellen werden sowohl in `raster_columns` als auch in `raster_overviews` registriert, da sie sowohl eigene Raster darstellen, als auch, als niedriger aufgelöstes Zerrbild einer höher aufgelösten Tabelle, einem bestimmten Zweck dienen. Wenn Sie den `-l` Switch beim Laden des Rasters angeben, werden diese gemeinsam mit der Rasterhaupttabelle erstellt; sie können aber auch händisch über [AddOverviewConstraints](#) erstellt werden.

Übersichtstabellen enthalten dieselben Constraints wie andere Rastertabellen und zusätzliche informative Constraints, spezifisch für die Übersichten.



Note

Die Information in `raster_overviews` befindet sich nicht in `raster_columns`. Falls Sie die Information der Übersichtstabelle und der `raster_columns` zusammen benötigen, können Sie einen Join auf `raster_overviews` und `raster_columns` ausführen, um die gesamte Information zu erhalten.

Die zwei Hauptgründe für Übersichtsraaster 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 Überschlagsrechnungen 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 Tabellename 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 Übersichtsdiens 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 Bilddatei enthält die Basistabelle $(5000*5000)/(125*125)$ Datensätze = 1600, Ihre (l=2) `o_2` Tabelle hat dann eine Obergrenze von $(1600/\text{Power}(2,2)) = 400$ Zeilen, Ihre (l=3) `o_3` $(1600/\text{Power}(2,3)) = 200$ Zeilen. Wenn sich die Pixel nicht durch die Größe Ihrer Kacheln teilen lassen, erhalten Sie einige Ausschussskacheln (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 $(\text{Power}(2, \text{overview\_factor}))$ Pixel der Ursprungsdatei repräsentiert.

5.3 Eigene Anwendungen mit PostGIS Raster erstellen

Da PostGIS-Raster SQL-Funktionen für die bekannten Bildformate zur Verfügung stellt, haben Sie bei der Ausgabe von Rastern eine Reihe von Möglichkeiten. Zum Beispiel können Sie OpenOffice/LibreOffice für die Darstellung nutzen, so wie unter [Rendering PostGIS Raster graphics with LibreOffice Base Reports](#) dargestellt. Zusätzlich steht Ihnen eine Vielzahl an Sprachen zur Verfügung, wie in diesem Abschnitt gezeigt wird.

5.3.1 PHP Beispiel: Ausgabe mittels ST_AsPNG in Verbindung mit anderen Rasterfunktionen

In diesem Abschnitt zeigen wir die Anwendung des PHP PostgreSQL Treibers und der Funktion `ST_AsGDALRaster`, um die Bänder 1,2,3 eines Rasters an einen PHP Request-Stream zu übergeben. Dieser kann dann in einen "img src" HTML Tag eingebunden werden.

Dieses Beispiel zeigt, wie Sie ein ganzes Bündel von Rasterfunktionen kombinieren können, um jene Kacheln zu erhalten, die ein bestimmtes WGS84 Umgebungsrechteck schneiden. Anschließend werden alle Bänder dieser Kacheln mit `ST_Union` vereinigt, mit `ST_Transform` in die vom Benutzer vorgegebene Projektion transformiert und das Ergebnis mit `ST_AsPNG` als PNG ausgegeben.

Sie können das unten angeführte Programm über

```
http://mywebserver/test_raster.php?srid=2249
```

aufrufen, um den Raster in "Massachusetts State Plane Feet" zu erhalten.

```
<?php
/** Inahlt von test_raster.php */
$conn_str = 'dbname=mydb host=localhost port=5432 user=myuser password=mypwd';
$dbconn = pg_connect($conn_str);
header('Content-Type: image/png');
/**Wenn eine bestimmte Projektion angefragt ist, wird diese verwendet, ansonsten wird "Massachusetts State Plane Meters" verwendet */
if (!empty( $_REQUEST['srid'] ) && is_numeric( $_REQUEST['srid'] ) ){
    $input_srid = intval($_REQUEST['srid']);
}
else { $input_srid = 26986; }
/** "set bytea_output" für PostgreSQL 9.0+, wird für 8.4 nicht benötigt*/
$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]);
?>
```

5.3.2 ASP.NET C# Beispiel: Ausgabe mittels ST_AsPNG in Verbindung mit anderen Rasterfunktionen

In diesem Abschnitt zeigen wir die Anwendung des Npgsql PostgreSQL .NET Treibers und der Funktion `ST_AsGDALRaster`, um die Bänder 1,2,3 eines Rasters an einen PHP Request-Stream zu übergeben. Dieser kann dann in einen "img src" HTML Tag eingebunden werden.

Für dieses Beispiel benötigen Sie den npgsql .NET PostgreSQL Treiber. Um loslegen zu können, reicht es aus, dass Sie die neueste Version von <http://npgsql.projects.postgresql.org/> in Ihren ASP.NET Ordner laden.

Dieses Beispiel zeigt, wie Sie ein ganzes Bündel von Rasterfunktionen kombinieren können, um jene Kacheln zu erhalten, die ein bestimmtes WGS84 Umgebungsrechteck schneiden. Anschließend werden alle Bänder dieser Kacheln mit **ST_Union** vereinigt, mit **ST_Transform** in die vom Benutzer vorgegebene Projektion transformiert und das Ergebnis mit **ST_AsPNG** als PNG ausgegeben.

Dasselbe Beispiel wie Section 5.3.1 nur in C# implementiert.

Sie können das unten angeführte Programm über

```
http://mywebserver/TestRaster.ashx?srid=2249
```

aufrufen, um den Raster in "Massachusetts State Plane Feet" zu bekommen.

```
-- web.config Verbindungsaufbau --
<connectionStrings>
  <add name="DSN"
    connectionString="server=localhost;database=mydb;Port=5432;User Id=myuser;password= ↵
    mypwd"/>
</connectionStrings>
>
```

```
// Code für TestRaster.ashx
<%@ WebHandler Language="C#" Class="TestRaster" %>
using System;
using System.Data;
using System.Web;
using Npgsql;

public class TestRaster : IHttpHandler
{
    public void ProcessRequest(HttpContext context)
    {
        context.Response.ContentType = "image/png";
        context.Response.BinaryWrite(GetResults(context));
    }

    public bool IsReusable {
        get { return false; }
    }

    public byte[] GetResults(HttpContext context)
    {
        byte[] result = null;
        NpgsqlCommand command;
        string sql = null;
        int input_srid = 26986;

        try {
            using (NpgsqlConnection conn = new NpgsqlConnection(System. ↵
                Configuration.ConfigurationManager.ConnectionStrings["DSN"]. ↵
                ConnectionString)) {
                conn.Open();

                if (context.Request["srid"] != null)
                {
                    input_srid = Convert.ToInt32(context.Request["srid"]);
                }
                sql = @"SELECT ST_AsPNG(
                    ST_Transform(
```

```

        ST_AddBand(
            ST_Union(rast,1), ARRAY[ST_Union(rast,2),ST_Union(rast,3)]
                        ,:input_srid) ) As new_rast
FROM aerials.boston
WHERE
    ST_Intersects(rast,
        ST_Transform(ST_MakeEnvelope(-71.1217, 42.227, ↵
            -71.1210, 42.218,4326),26986) );
command = new NpgsqlCommand(sql, conn);
command.Parameters.Add(new NpgsqlParameter("input_srid", input_srid));

        result = (byte[]) command.ExecuteScalar();
conn.Close();
    }

}
catch (Exception ex)
{
    result = null;
    context.Response.Write(ex.Message.Trim());
}

    return result;
}
}

```

5.3.3 Applikation für die Java-Konsole, welche eine Rasterabfrage als Bilddatei ausgibt

Eine einfache Java Applikation, die eine Abfrage entgegennimmt, ein Bild erzeugt und in eine bestimmte Datei ausgibt.

Sie können die neuesten PostgreSQL JDBC Treiber unter <http://jdbc.postgresql.org/download.html> herunterladen.

Sie können den unten angegebenen Code mit einem Befehl wie folgt kompilieren:

```

set env CLASSPATH ....\postgresql-9.0-801.jdbc4.jar
javac SaveQueryImage.java
jar cfm SaveQueryImage.jar Manifest.txt *.class

```

Und ihn von der Befehlszeile wie folgt aufrufen:

```

java -jar SaveQueryImage.jar "SELECT ST_AsPNG(ST_AsRaster(ST_Buffer(ST_Point(1,5),10, ' ↵
    quad_segs=2'),150, 150, '8BUI',100));" "test.png"

```

```

-- Manifest.txt --
Class-Path: postgresql-9.0-801.jdbc4.jar
Main-Class: SaveQueryImage

```

```

// Code für SaveQueryImage.java
import java.sql.Connection;
import java.sql.SQLException;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.io.*;

public class SaveQueryImage {
    public static void main(String[] argv) {
        System.out.println("Checking if Driver is registered with DriverManager.");

        try {
            //java.sql.DriverManager.registerDriver (new org.postgresql.Driver());

```

```

        Class.forName("org.postgresql.Driver");
    }
    catch (ClassNotFoundException cnfe) {
        System.out.println("Couldn't find the driver!");
        cnfe.printStackTrace();
        System.exit(1);
    }

    Connection conn = null;

    try {
        conn = DriverManager.getConnection("jdbc:postgresql://localhost:5432/mydb","myuser ←
            ", "mypwd");
        conn.setAutoCommit(false);

        PreparedStatement sGetImg = conn.prepareStatement(argv[0]);

        ResultSet rs = sGetImg.executeQuery();

        FileOutputStream fout;
        try
        {
            rs.next();
            /** Output to file name requested by user **/
            fout = new FileOutputStream(new File(argv[1]) );
            fout.write(rs.getBytes(1));
            fout.close();
        }
        catch(Exception e)
        {
            System.out.println("Can't create file");
            e.printStackTrace();
        }

        rs.close();
        sGetImg.close();
        conn.close();
    }
    catch (SQLException se) {
        System.out.println("Couldn't connect: print out a stack trace and exit.");
        se.printStackTrace();
        System.exit(1);
    }
}
}

```

5.3.4 Verwenden Sie PLPython um Bilder via SQL herauszuschreiben

Diese als plpython gespeicherte Prozedur erzeugt eine Datei pro Datensatz im Serververzeichnis. Benötigt die Installation von plpython. Funktioniert sowohl mit plpythonu als auch mit plpython3u.

```

CREATE OR REPLACE FUNCTION write_file (param_bytes bytea, param_filepath text)
RETURNS text
AS $$
f = open(param_filepath, 'wb+')
f.write(param_bytes)
return param_filepath
$$ LANGUAGE plpythonu;

```

```
-- 5 Bilder in verschiedenen Größen nach PostgreSQL schreiben
```



```
-- der Account des PostgreSQL Daemons benötigt Schreibrechte auf den Ordner
-- die Namen der erzeugten Dateien werden ausgegeben;
SELECT write_file(ST_AsPNG(
    ST_AsRaster(ST_Buffer(ST_Point(1,5),j*5, 'quad_segs=2'),150*j, 150*j, '8BUI',100)),
    'C:/temp/slices'|| j || '.png')
FROM generate_series(1,5) As j;

write_file
-----
C:/temp/slices1.png
C:/temp/slices2.png
C:/temp/slices3.png
C:/temp/slices4.png
C:/temp/slices5.png
```

5.3.5 Faster mit PSQL ausgeben

Leider hat PSQL keine einfach zu benützende Funktion für die Ausgabe von Binärdateien eingebaut. Dieser Hack baut auf der etwas veralteten "Large Object" Unterstützung von PostgreSQL auf. Um ihn anzuwenden verbinden Sie sich bitte zuerst über die Befehlszeile "psql" mit Ihrer Datenbank.

Anders als beim Ansatz mit Python, wird die Datei bei diesem Ansatz auf Ihrem lokalen Rechner erzeugt.

```
SELECT oid, lowrite(lo_open(oid, 131072), png) As num_bytes
FROM
( VALUES (lo_create(0),
    ST_AsPNG( (SELECT rast FROM aerials.boston WHERE rid=1) )
) ) As v(oid,png);
-- die Ausgabe sieht ungefähr so aus --
oid | num_bytes
-----+-----
2630819 | 74860

-- beachten Sie die OID und ersetzen Sie c:/test.png mit dem tatsächlichen Pfad
-- auf Ihrem Rechner
\lo_export 2630819 'C:/temp/aerial_samp.png'

-- löscht die Datei aus dem "large object" Speicher der Datenbank
SELECT lo_unlink(2630819);
```

Chapter 6

Anwendung der PostGIS Geometrie: Applikation-entwicklung

6.1 Verwendung von MapServer

Der Minnesota MapServer ist ein Kartendienstserver für das Internet, der die "OpenGIS Web Map Service (WMS) Implementation Specification" erfüllt.

- Die MapServer Homepage finden Sie unter <http://mapserver.org>.
- Die OpenGIS Web Map Spezifikation finden Sie unter <http://www.opengeospatial.org/standards/wms>.

6.1.1 Grundlegende Handhabung

Um PostGIS mit MapServer zu verwenden müssen Sie wissen, wie Sie MapServer konfigurieren, da dies den Rahmens dieser Dokumentation sprengen würde. Dieser Abschnitt deckt PostGIS-spezifische Themen und Konfigurationsdetails ab.

Um PostGIS mit MapServer zu verwenden, benötigen Sie:

- Die PostGIS Version 0.6, oder höher.
- Die MapServer Version 3.5, oder höher.

MapServer greift auf die PostGIS/PostgreSQL-Daten, so wie jeder andere PostgreSQL-Client, über die `libpq` Schnittstelle zu. Dies bedeutet, dass MapServer auf jedem Server, der Netzwerkzugriff auf den PostgreSQL Server hat, installiert werden kann und PostGIS als Datenquelle nutzen kann. Je schneller die Verbindung zwischen den beiden Systemen, desto besser.

1. Es spielt keine Rolle, mit welchen Optionen Sie MapServer kompilieren, solange sie bei der Konfiguration die "--with-postgis"-Option angeben.
2. Fügen Sie einen PostGIS Layer zu der MapServer *.map Datei hinzu. Zum Beispiel:

```
LAYER
  CONNECTIONTYPE postgis
  NAME "widehighways"
  # Verbindung zu einer remote Geodatenbank
  CONNECTION "user=dbuser dbname=gisdatabase host=bigserver"
  PROCESSING "CLOSE_CONNECTION=DEFER"
  # Um die Zeilen der 'geom'-Spalte aus der 'roads'-Tabelle zu erhalten
  DATA "geom from roads using srid=4326 using unique gid"
  STATUS ON
```

```

TYPE LINE
# Von den im Ausschnitt vorhandenen Linien nur die breiten Hauptstraßen/Highways
FILTER "type = 'highway' and numlanes >= 4"
CLASS
# Autobahnen heller und 2Pixel stark machen
EXPRESSION ([numlanes] >= 6)
STYLE
  COLOR 255 22 22
  WIDTH 2
END
END
CLASS
# Der ganze Rest ist dunkler und nur 1 Pixel stark
EXPRESSION ([numlanes] < 6)
STYLE
  COLOR 205 92 82
END
END
END

```

Im oberen Beispiel werden folgende PostGIS-spezifische Anweisungen verwendet:

CONNECTIONTYPE Für PostGIS Layer ist dies immer "postgis".

CONNECTION Die Datenbankverbindung wird durch einen "Connection String" bestimmt, welcher aus einer standardisierten Menge von Schlüsseln und Werten zusammengesetzt ist (Standardwerte zwischen <>):

user=<username> password=<password> dbname=<username> hostname=<server> port=<5432>

Ein leerer "Connection String" ist ebenfalls gültig, sodass jedes Key/Value Paar weggelassen werden kann. Üblicherweise wird man zumindest den Datenbanknamen und den Benutzernamen, mit dem man sich verbinden will, angeben.

DATA Dieser Parameter hat die Form "<geocolumn> from <tablename> using srid=<srid> using unique <primary key>", wobei "geocolumn" dem räumlichen Attribut entspricht, mit dem die Bildsynthese/rendern durchgeführt werden soll. "srid" entspricht der SRID des räumlichen Attributs und "primary key" ist der Primärschlüssel der Tabelle (oder ein anderes eindeutiges Attribut mit einem Index).

Sie können sowohl "using srid" als auch "using unique" weglassen. Wenn möglich, bestimmt MapServer die korrekten Werte dann automatisch, allerdings zu den Kosten einiger zusätzlichen serverseitigen Abfragen, die bei jedem Kartenaufruf ausgeführt werden.

PROCESSING Wenn Sie mehrere Layer darstellen wollen, fügen Sie CLOSE_CONNECTION=DEFER ein, dadurch wird eine bestehende Verbindung wiederverwendet anstatt geschlossen. Dies erhöht die Geschwindigkeit. Unter [MapServer PostGIS Performance Tips](#) findet sich eine detaillierte Erklärung.

FILTER Der Filter muss ein gültiger SQL-Text sein, welcher der Logik, die normalerweise dem "WHERE" Schlüsselwort in der SQL-Abfrage folgt, entspricht. Z.B.: um nur die Straßen mit 6 oder mehr Spuren zu rendern, können Sie den Filter "num_lanes >= 6" verwenden.

3. Stellen Sie bitte sicher, das für alle zu zeichnenden Layer, ein räumlicher Index (GIST) in der Geodatenbank angelegt ist.

```
CREATE INDEX [indexname] ON [tabellenname] USING GIST ( [geometry_spalte] );
```

4. Wenn Sie Ihre Layer über MapServer abfragen wollen, benötigen Sie auch die "using unique" Klausel in Ihrer "DATA" Anweisung.

MapServer benötigt für jeden räumlichen Datensatz, der abgefragt werden soll, eindeutige Identifikatoren. Das PostGIS Modul von MapServer benützt den von Ihnen festgelegten, eindeutigen Wert, um diese eindeutige Identifikatoren zur Verfügung zu stellen. Den Primärschlüssel zu verwenden gilt als Erfolgsrezept.

6.1.2 Häufig gestellte Fragen

1. Wenn Ich *EXPRESSION* in meiner *.map Datei verwende, gibt die *WHERE* Bedingung niemals *TRUE* zurück, obwohl Ich weiß, dass sich die Werte in meiner Tabelle befinden.

Anders als bei Shapefiles müssen die PostGIS-Feldnamen in *EXPRESSION* mit *Kleinbuchstaben* eingetragen werden.

```
EXPRESSION ([numlanes] >= 6)
```

2. *Der FILTER, den ich bei meinen Shapefiles verwende, funktioniert nicht für meine PostGIS Tabelle, obwohl diese die gleichen Daten aufweist.*

Anders als bei Shapefiles, nutzen die Filter bei PostGIS-Layern die SQL Syntax (sie werden an die SQL-Anweisung, die vom PostGIS Konnektor für die Darstellung der Layer im Mapserver erzeugt wird, angehängt).

```
FILTER "type = 'highway' and numlanes >= 4"
```

3. *Mein PostGIS Layer wird viel langsamer dargestellt als mein Shapefile Layer. Ist das normal?*

Je mehr Features eine bestimmte Karte aufweist, umso wahrscheinlicher ist es, dass PostGIS langsamer ist als Shapefiles. Bei Karten mit relativ wenigen Features (100te) ist PostGIS meist schneller. Bei Karten mit einer hohen Feature Dichte (1000e) wird PostGIS immer langsamer sein. Falls erhebliche Probleme mit der Zeichenperformance auftreten, haben Sie eventuell keinen räumlichen Index auf die Tabelle gelegt.

```
postgis# CREATE INDEX geotable_gix ON geotable USING GIST ( geocolumn );
postgis# VACUUM ANALYZE;
```

4. *Mein PostGIS Layer wird ausgezeichnet dargestellt, aber die Abfragen sind sehr langsam. Was läuft falsch?*

Damit Abfragen schnell gehen, müssen Sie einen eindeutigen Schlüssel in Ihrer Tabelle haben und einen Index auf diesen eindeutigen Schlüssel legen. Sie können den von MapServer zu verwendenden eindeutigen Schlüssel mit der USING UNIQUE Klausel in Ihrer DATA Zeile angeben:

```
DATA "geom FROM geotable USING UNIQUE gid"
```

5. *Kann Ich "Geography" Spalten (neu in PostGIS 1.5) als Quelle für MapServer Layer verwenden?*

Ja! Für MapServer sind "Geography" Attribute und "Geometry" Attribute dasselbe. Es kann allerdings nur die SRID 4326 verwendet werden. Stellen Sie bitte sicher, dass sich eine "using srid=4326" Klausel in Ihrer DATA Anweisung befindet. Alles andere funktioniert genau so wie bei "Geometry".

```
DATA "geog FROM geogtable USING SRID=4326 USING UNIQUE gid"
```

6.1.3 Erweiterte Verwendung

Die SQL-Pseudoklausel USING wird verwendet, um MapServer zusätzliche Information über komplexere Abfragen zukommen zu lassen. Genauer gesagt, wenn entweder ein View oder ein Subselect als Ursprungstabelle verwendet wird (der Ausdruck rechts von "FROM" bei einer DATA Definition) ist es für MapServer schwieriger einen eindeutigen Identifikator für jede Zeile und die SRID der Tabelle automatisch zu bestimmen. Die USINGKlausel kann MapServer die Information über diese beiden Teile wie folgt zukommen lassen:

```
DATA "geom FROM (
  SELECT
    table1.geom AS geom,
    table1.gid AS gid,
    table2.data AS data
  FROM table1
  LEFT JOIN table2
  ON table1.id = table2.id
) AS new_table USING UNIQUE gid USING SRID=4326"
```

USING UNIQUE <uniqueid> MapServer benötigt eine eindeutige ID für jede Zeile um die Zeile bei Kartenabfragen identifizieren zu können. Normalerweise wird der Primärschlüssel aus den Systemtabellen ermittelt. Views und Subselects haben jedoch nicht automatisch eine bekannte eindeutige Spalte. Wenn Sie MapServer's Abfragefunktionalität nutzen wollen, müssen Sie sicherstellen, dass Ihr View oder Subselect eine mit eindeutigen Werten versehene Spalte enthält und diese mit USING UNIQUE gekennzeichnet ist. Zum Beispiel können Sie hierfür die Werte des Primärschlüssels verwenden, oder irgendeine andere Spalte bei der sichergestellt ist dass sie eindeutige Werte für die Ergebnismenge aufweist.

**Note**

"eine Karte abfragen" ist jene Aktion, bei der man auf die Karte klickt und nach Information über Kartenfeatures an dieser Stelle fragt. Verwechseln Sie bitte nicht "Kartenabfragen" mit der SQL Abfrage in der DATA Definition.

USING SRID=<srid> PostGIS muss wissen, welches Koordinatenreferenzsystem von der Geometrie verwendet wird, um korrekte Daten an MapServer zurückzugeben. Üblicherweise kann man diese Information in der Tabelle "geometry_columns" in der PostGIS Datenbank finden. Dies ist jedoch nicht möglich bei Tabellen die On-the-fly erzeugt wurden, wo wie bei Subselects oder Views. Hierfür erlaubt die Option `USING SRID=` die Festlegung der richtigen SRID in der DATA Definition.

6.1.4 Beispiele

Beginnen wir mit einem einfachen Beispiel und arbeiten uns dann langsam vor. Betrachten Sie die nachfolgende MapServer Layerdefinition:

```
LAYER
  CONNECTIONTYPE postgis
  NAME "roads"
  CONNECTION "user=theuser password=thepass dbname=thedb host=theserver"
  DATA "geom from roads"
  STATUS ON
  TYPE LINE
  CLASS
    STYLE
      COLOR 0 0 0
    END
  END
END
```

Dieser Layer stellt alle Straßengeometrien der "roads"-Tabelle schwarz dar.

Angenommen, wir wollen bis zu einem Maßstab von 1:100000 nur die Autobahnen anzeigen - die nächsten zwei Layer erreichen diesen Effekt:

```
LAYER
  CONNECTIONTYPE postgis
  CONNECTION "user=theuser password=thepass dbname=thedb host=theserver"
  PROCESSING "CLOSE_CONNECTION=DEFER"
  DATA "geom from roads"
  MINSCALE 100000
  STATUS ON
  TYPE LINE
  FILTER "road_type = 'highway'"
  CLASS
    COLOR 0 0 0
  END
END

LAYER
  CONNECTIONTYPE postgis
  CONNECTION "user=theuser password=thepass dbname=thedb host=theserver"
  PROCESSING "CLOSE_CONNECTION=DEFER"
  DATA "geom from roads"
  MAXSCALE 100000
  STATUS ON
  TYPE LINE
  CLASSITEM road_type
  CLASS
    EXPRESSION "highway"
    STYLE
```

```

        WIDTH 2
        COLOR 255 0 0
    END
END
CLASS
    STYLE
        COLOR 0 0 0
    END
END
END
END

```

Der erste Layer wird verwendet, wenn der Maßstab größer als 1:100000 ist und es werden nur die Straßen vom Typ "highway"/Autobahn als schwarze Linien dargestellt. Die Option `FILTER` bedingt, dass nur Straßen vom Typ "highway" angezeigt werden.

Der zweite Layer wird angezeigt, wenn der Maßstab kleiner als 1:100000 ist. Er zeigt die Autobahnen als doppelt so dicke rote Linien an, die anderen Straßen als normale schwarze Linien.

Wir haben eine Reihe von interessanten Aufgaben lediglich mit der von MapServer zur Verfügung gestellten Funktionalität durchgeführt, und unsere SQL-Anweisung unter `DATA` ist trotzdem einfach geblieben. Angenommen, die Namen der Straßen sind in einer anderen Tabelle gespeichert (wieso auch immer) und wir müssen einen Join ausführen, um sie für die Straßenbeschriftung verwenden zu können.

```

LAYER
    CONNECTIONTYPE postgis
    CONNECTION "user=theuser password=thepass dbname=thedb host=theserver"
    DATA "geom FROM (SELECT roads.gid AS gid, roads.geom AS geom,
        road_names.name as name FROM roads LEFT JOIN road_names ON
        roads.road_name_id = road_names.road_name_id)
        AS named_roads USING UNIQUE gid USING SRID=4326"
    MAXSCALE 20000
    STATUS ON
    TYPE ANNOTATION
    LABELITEM name
    CLASS
        LABEL
            ANGLE auto
            SIZE 8
            COLOR 0 192 0
            TYPE truetype
            FONT arial
        END
    END
END

```

Dieser Beschriftungslayer fügt grüne Beschriftungen zu allen Straßen hinzu, wenn der Maßstab 1:20000 oder weniger wird. Es zeigt auch wie man einen SQL-Join in einer `DATA` Definition verwenden kann.

6.2 Java Clients (JDBC)

Java Clients können auf die PostGIS Geoobjekte in der PostgreSQL Datenbank entweder direkt über die Textdarstellung zugreifen oder über die Objekte der JDBC Erweiterung, die mit PostGIS gebündelt sind. Um die Objekte der Erweiterung zu nutzen, muss sich die Datei "postgis.jar" zusammen mit dem JDBC Treiberpaket "postgresql.jar" in Ihrem `CLASSPATH` befinden.

```

import java.sql.*;
import java.util.*;
import java.lang.*;
import org.postgis.*;

public class JavaGIS {

```

```

public static void main(String[] args) {

    java.sql.Connection conn;

    try {
        /*
         * Den JDBC Treiber laden und eine Verbindung herstellen.
         */
        Class.forName("org.postgresql.Driver");
        String url = "jdbc:postgresql://localhost:5432/database";
        conn = DriverManager.getConnection(url, "postgres", "");
        /*
         * Die geometrischen Datentypen zu der Verbindung hinzufügen. Beachten Sie bitte,
         * dass Sie die Verbindung in eine pgsq- spezifische Verbindung umwandeln
         * bevor Sie die Methode addDataType() aufrufen.
         */
        ((org.postgresql.PGConnection) conn).addDataType("geometry", Class.forName("org.postgis. ←
            PGgeometry"));
        ((org.postgresql.PGConnection) conn).addDataType("box3d", Class.forName("org.postgis. ←
            PGbox3d"));
        /*
         * Eine Anweisung erzeugen und eine Select Abfrage ausführen.
         */
        Statement s = conn.createStatement();
        ResultSet r = s.executeQuery("select geom,id from geomtable");
        while( r.next() ) {
            /*
             * Die Geometrie als Objekt abrufen und es in einen geometrischen Datentyp umwandeln.
             * Ausdrucken.
             */
            PGgeometry geom = (PGgeometry)r.getObject(1);
            int id = r.getInt(2);
            System.out.println("Row " + id + ":");
            System.out.println(geom.toString());
        }
        s.close();
        conn.close();
    }
    catch( Exception e ) {
        e.printStackTrace();
    }
}

```

Das Objekt "PGgeometry" ist ein Adapter, der abhängig vom Datentyp ein bestimmtes topologisches Geoobjekt (Unterlassen der abstrakten Klasse "Geometry") enthält: Point, LineString, Polygon, MultiPoint, MultiLineString, MultiPolygon.

```

PGgeometry geom = (PGgeometry)r.getObject(1);
if( geom.getType() == Geometry.POLYGON ) {
    Polygon pl = (Polygon)geom.getGeometry();
    for( int r = 0; r < pl.numRings(); r++) {
        LinearRing rng = pl.getRing(r);
        System.out.println("Ring: " + r);
        for( int p = 0; p < rng.numPoints(); p++ ) {
            Point pt = rng.getPoint(p);
            System.out.println("Point: " + p);
            System.out.println(pt.toString());
        }
    }
}

```

Das JavaDoc der Erweiterung liefert eine Referenz für die verschiedenen Zugriffsfunktionen auf die Geoobjekte.

6.3 C Clients (libpq)

...

6.3.1 Text Cursor

...

6.3.2 Binäre Cursor

...

Chapter 7

Performance Tipps

7.1 Kleine Tabellen mit großen Geometrien

7.1.1 Problembeschreibung

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

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

Stellen Sie sich nun eine Abfrage vor, die den geometrischen Operator `&&` verwendet, um ein Umgebungsrechteck mit nur wenigen Zeilen zu ermitteln. Der Abfrageoptimierer stellt fest, dass die Tabelle nur 3 Speicherseiten und 80 Zeilen aufweist. Er nimmt an, dass 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.postgresql.org/pgsql-performance/2005-02/msg00030.php>

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

7.1.2 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 dass der Query Planer Fehleinschätzungen in anderen Fällen macht. Daher sollte "SET enable_seqscan TO on;" nach der Abfrage ausgeführt werden.

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

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

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

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

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

7.2 CLUSTER auf die geometrischen Indizes

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

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

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

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

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

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

7.3 Vermeidung von Dimensionsumrechnungen

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

```
UPDATE mytable SET the_geom = ST_Force2D(the_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.

7.4 Tunen der Konfiguration

Die Feinabstimmung von PostGIS ist dem Tunen für jeglichen PostgreSQL workload sehr ähnlich. Die einzige zusätzliche Anmerkung, die Sie im Kopf behalten müssen ist, das Geometrien und Raster groß sind, so das auf den Arbeitsspeicher bezogene Optimierungen im allgemeinen einen größeren Einfluß auf PostGIS haben als andere Arten von PostgreSQL Abfragen.

Für allgemeine Details zur PostgreSQL Optimierung, siehe [Tuning your PostgreSQL Server](#).

Für PostgreSQL 9.4+ kann dies alles auf der Serverebene gesetzt werden, ohne das postgresql.conf oder postgresql.auto.conf angerührt werden müssen, indem man den ALTER SYSTEM. . Befehl nützt.

```
ALTER SYSTEM SET work_mem = '256MB';
-- dies erzwingt die non-startup Konfiguration für neue Verbindungen
SELECT pg_reload_conf();
-- zeige aktuelle Einstellungen
-- benutzen Sie bitte SHOW ALL um alle Einstellungen anzuzeigen
SHOW work_mem;
```

Zusätzlich zu diesen Einstellungen verfügt PostGIS über einige eigene Einstellungen welche unter Section 8.2 aufgeführt sind.

7.4.1 Startup/Inbetriebnahme

Folgende Einstellungen werden in der postgresql.conf konfiguriert:

[constraint_exclusion](#)

- Default: partition
- Dies wird im allgemeinen zur Tabellenpartitionierung verwendet. Die Standardeinstellung ist "partition". Dies ist ideal für PostgreSQL 8.4 oder höher, da es den Query Planer veranlasst nur jene Tabellen, die sich in einer vererbten Hierarchie befinden, in Hinblick auf Bedingungen zu analysieren.

[shared_buffers](#)

- Standard: ~128MB in PostgreSQL 9.6
- Auf ca. 25% bis 40% des verfügbaren RAM setzen. Unter Windows können Sie nicht so hoch ansetzen.

[max_worker_processes](#) Diese Einstellung ist erst ab PostgreSQL 9.4+ verfügbar. Ab PostgreSQL 9.6+ hat diese Einstellung eine zusätzliche Bedeutung, da sie die maximale Anzahl von Prozessen bestimmt, die bei parallel ablaufenden Abfragen verwendet werden können.

- Default: 8
- Bestimmt die maximale Anzahl von Hintergrundprozessen die das System unterstützen kann. Dieser Parameter kann nur beim Serverstart gesetzt werden.

7.4.2 Runtime/Laufzeit

[work_mem](#) (jener Arbeitsspeicher der für Sortieroperationen und komplexe Abfragen benutzt wird)

- Default: 1-4MB
- Nach oben setzen für große Datenbanken, komplexe Abfragen und wenn große Mengen an Arbeitsspeicher zur Verfügung stehen.
- Nach unten setzen bei vielen gleichzeitigen Anwendern oder wenn wenig Arbeitsspeicher zur Verfügung steht

- Falls sie viel Arbeitsspeicher aber wenig Entwickler zur Verfügung haben:

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

maintenance_work_mem (wird für VACUUM, CREATE INDEX, etc. genutzt)

- Default: 16-64MB
- Ist im allgemeinen zu niedrig gesetzt - bindet I/O, sperrt Objekte während es den Speicher auslagert
- Es werden 32MB bis 1GB auf Produktionsservern mit viel Hauptspeicher empfohlen, was allerdings von der Anzahl der Anwender abhängt. Falls Sie eine Menge an Arbeitsspeicher und wenige Entwickler haben:

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

max_parallel_workers_per_gather Diese Einstellung ist erst ab PostgreSQL 9.6+ verfügbar und wirkt sich erst ab PostGIS 2.3+ aus, da nur PostGIS 2.3+ parallel laufende Abfragen unterstützt. Wenn der Parameter auf einen höheren Wert als 0 gestellt wird können manche Abfragen, wie jene die Relationsfunktionen wie `ST_Intersects` verwenden, mehrere Prozesse benutzen und mehr als doppelt so schnell ablaufen. Falls Sie eine große Anzahl an Prozessoren zur Verfügung haben, sollten Sie diesen Wert auf die Anzahl der Prozessoren setzen. Stellen Sie bitte auch sicher die `max_worker_processes` auf zumindest die gleiche Anzahl zu erhöhen.

- Default: 0
- Setzt die maximale Anzahl von Workern, die durch einen einzelnen Gather Knoten aufgerufen werden können. Parallele Worker werden aus einem Pool von Prozessen, welche von `max_worker_processes` aufgespannt sind, entnommen. Beachten Sie bitte, das die angeforderte Anzahl von Workern zur Laufzeit nicht tatsächlich zur Verfügung stehen kann. Falls dies auftritt wird der Plan mit weniger Workern als erwartet ausgeführt, was leistungsschwach sein kann. Setzt man den Wert auf 0, was der Standardeinstellung entspricht, wird die parallele Ausführung von Abfragen unterbunden.

Chapter 8

PostGIS Referenz

Nachfolgend sind jene Funktionen, die ein PostGIS Anwender am ehesten benötigt. Es gibt weitere Funktionen, die jedoch keinen Nutzen für den allgemeinen Anwender haben, da sie zu Unterstützungsaufgaben für PostGIS Objekte benötigt werden.

Note



PostGIS hat begonnen die bestehende Namensgebung in eine SQL-MM-orientierte Namensgebung zu ändern. Daher wurden die meisten Funktionen, die Sie kennen und lieben, unter Verwendung des standard spatial type (ST) Präfix umbenannt. Vorhergegangene Funktionen sind noch verfügbar, werden aber in diesem Dokument nicht aufgeführt, wenn es entsprechende aktualisierte Funktionen gibt. Funktionen ohne das ST_ Präfix, die in dieser Dokumentation nicht aufgeführt sind, gelten als veraltet und werden in zukünftigen Ausgaben entfernt. Benutzen Sie diese daher BITTE NICHT MEHR.

8.1 PostgreSQL PostGIS Geometry/Geography/Box Types

8.1.1 box2d

box2d — Ein Rechteck bestehend aus X min, Y min, X max und Y max. Wird oft verwendet, um das umschliessende Rechteck einer Geometrie in 2D zu erhalten.

Beschreibung

Box2D ist ein räumlicher Datentyp, der verwendet wird, das umschließende Rechteck einer Geometrie, oder einer Menge von Geometrien, darzustellen. In Vorgängerversionen von PostGIS 1.4 würde ST_Extent eine Box2D zurückgeben.

8.1.2 box3d

box3d — Ein Rechteck bestehend aus Xmin, Ymin, Zmin, Xmax, Ymax und Zmax. Wird oft verwendet, um das umschliessende Rechteck einer Geometrie oder einer Geometriekollektion in 3D zu erhalten.

Beschreibung

Box3D ist ein räumlicher Datentyp, der verwendet wird, das umschließende Rechteck einer Geometrie, oder einer Menge von Geometrien, darzustellen. ST_3DExtent gibt ein Box3D-Objekt zurück.

Typumwandlung

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

Typumwandlung nach	Verhaltensweise
box	automatisch
box2d	automatisch
geometry	automatisch

8.1.3 geometry

geometry — Ebener räumlicher Datentyp.

Beschreibung

"Geometry" wird in PostGIS als grundlegender räumlicher Datentyp zur Darstellung eines Features im Euklidischen Koordinatenraum verwendet.

Typumwandlung

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

Typumwandlung nach	Verhaltensweise
box	automatisch
box2d	automatisch
box3d	automatisch
Bytea	automatisch
geography	automatisch
Text	automatisch

Siehe auch

Section [4.1](#)

8.1.4 geometry_dump

geometry_dump — Ein räumlicher Datentyp mit zwei Attributen - geom (enthält ein Geometrieobjekt) und Path[] (ein 1-dimensionales Feld, welches die Position der Geometrie innerhalb des entladenen/dumped Objektes angibt).

Beschreibung

geometry_dump ist ein zusammengesetzter Datentyp, der aus einem geometrischen Objekt - wird durch das Attribut .geom referenziert - und path[] - einem 1-dimensionalen ganzzahligen Feld (beginnt mit 1, z.B.: path[1] gibt das erste Element des Feldes) das den Navigationspfad innerhalb der ausgeladenen/dumped Geometrie zur Auffindung von einzelnen Elementen bestimmt. Es wird von der ST_Dump* Funktionsfamilie als Ausgabetyt verwendet, um komplexere Geometrien in ihre Bestandteile und die Position dieser Bestandteile zu zerlegen.

Siehe auch

Section [14.6](#)

8.1.5 geography

geography — Ein ellipsenförmiger räumlicher Datentyp.

Beschreibung

"Geography" ist ein räumlicher Datentyp, zur Abbildung eines Features im geographischen Kugelkoordinatensystem.

Typumwandlung

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

Typumwandlung nach	Verhaltensweise
geometry	explizit

Siehe auch

Section [14.4](#), Section [4.2](#)

8.2 PostGIS Grand Unified Custom Variables (GUCs)

8.2.1 postgis.backend

postgis.backend — Dieses Backend stellt eine Funktion zur Auswahl zwischen GEOS und SFCGAL zur Verfügung.

Beschreibung

Diese GUC hat nur Bedeutung, wenn Sie PostGIS mit SFCGAL Unterstützung kompiliert haben. Funktionen, welche sowohl bei GEOS als auch bei SFCGAL die gleiche Bezeichnung haben, werden standardmäßig mit dem `geos` Backend ausgeführt. Die Standardeinstellung wird mit dieser Variablen überschrieben und SFCGAL für den Aufruf verwendet.

Verfügbarkeit: 2.1.0

Beispiele

Setzt das Backend für die Dauer der Verbindung

```
set postgis.backend = sfcgal;
```

Setzt das Backend für neue Verbindungen zur Datenbank

```
ALTER DATABASE mygisdb SET postgis.backend = sfcgal;
```

Siehe auch

Section [8.10](#)

8.2.2 postgis.gdal_datapath

postgis.gdal_datapath — Eine Konfigurationsmöglichkeit um den Wert von GDAL's GDAL_DATA Option zu setzen. Wenn sie nicht gesetzt ist, wird die Umgebungsvariable GDAL_DATA verwendet.

Beschreibung

Eine PostgreSQL GUC Variable zum Setzen von GDAL's GDAL_DATA Option. Der `postgis.gdal_datapath` Wert sollte dem gesamten physischen Pfad zu den Datendateien von GDAL entsprechen.

Diese Konfigurationsmöglichkeit ist am nützlichsten auf Windows Plattformen, wo der Dateipfad von "data" nicht fest kodiert ist. Diese Option sollte auch gesetzt werden, wenn sich die Datendateien nicht in dem von GDAL erwarteten Pfad befinden.



Note

Diese Option kann in der Konfigurationsdatei "postgresql.conf" gesetzt werden. Sie kann auch pro Verbindung oder pro Transaktion gesetzt werden.

Verfügbarkeit: 2.2.0



Note

Zusätzliche Informationen über GDAL_DATA ist unter den [Konfigurationsmöglichkeiten](#) für GDAL zu finden.

Beispiele

Den `postgis.gdal_datapath` setzen oder zurücksetzen

```
SET postgis.gdal_datapath TO '/usr/local/share/gdal.hidden';  
SET postgis.gdal_datapath TO default;
```

Auf Windows für eine bestimmte Datenbank setzen

```
ALTER DATABASE gisdb  
SET postgis.gdal_datapath = 'C:/Program Files/PostgreSQL/9.3/gdal-data';
```

Siehe auch

[PostGIS_GDAL_Version](#), [ST_Transform](#)

8.2.3 postgis.gdal_enabled_drivers

`postgis.gdal_enabled_drivers` — Eine Konfigurationsmöglichkeit um einen GDAL Treiber in der PostGIS Umgebung zu aktivieren. Beeinflusst die Konfigurationsvariable GDAL_SKIP von GDAL.

Beschreibung

Eine Konfigurationsmöglichkeit um einen GDAL Treiber in der PostGIS Umgebung zu aktivieren. Beeinflusst die Konfigurationsvariable GDAL_SKIP von GDAL. Diese Option kann in der PostgreSQL Konfigurationsdatei "postgresql.conf" gesetzt werden. Sie kann aber auch pro Verbindung oder pro Transaktion gesetzt werden.

Der Ausgangswert von `postgis.gdal_enabled_drivers` kann auch beim Startprozess von PostgreSQL gesetzt werden, nämlich durch die Übergabe der Umgebungsvariablen `POSTGIS_GDAL_ENABLED_DRIVERS`, welche die Liste der aktivierten Treiber enthält.

Aktivierte GDAL Treiber können auch über die Kurzbezeichnung oder den Code des Treibers bestimmt werden. Kurzbezeichnungen und Codes für die Treiber finden sich unter [GDAL Raster Formate](#). Es können mehrere, durch Leerzeichen getrennte Treiber angegeben werden.

Note

Für `postgis.gdal_enabled_drivers` sind drei spezielle, case-sensitive Codes verfügbar.

- `DISABLE_ALL` deaktiviert alle GDAL-Treiber. Falls vorhanden, überschreibt `DISABLE_ALL` alle anderen Werte in `postgis.gdal_enabled_drivers`.
- `ENABLE_ALL` aktiviert alle GDAL-Treiber.
- `VSICURL` aktiviert GDAL's `/vsicurl/` virtuelles Dateisystem.

Falls `postgis.gdal_enabled_drivers` auf `DISABLE_ALL` gesetzt ist, kommt es bei der Anwendung von `out-db` Rastern, `ST_FromGDALRaster()`, `ST_AsGDALRaster()`, `ST_AsTIFF()`, `ST_AsJPEG()` und `ST_AsPNG()` zu Fehlermeldungen.

**Note**

`postgis.gdal_enabled_drivers` wird bei der Standardinstallation von PostGIS auf `DISABLE_ALL` gesetzt.

**Note**

Weiterführende Informationen über `GDAL_SKIP` ist auf GDAL's [Configuration Options](#) zu finden.

Verfügbarkeit: 2.2.0

Beispiele

`postgis.gdal_enabled_drivers` setzen und zurücksetzen

Bestimmt das Backend, das für alle neuen Verbindungen zur Datenbank verwendet wird

```
ALTER DATABASE mygisdb SET postgis.gdal_enabled_drivers TO 'GTiff PNG JPEG';
```

Setzt die standardmäßig aktivierten Treiber für alle neuen Verbindungen zum Server. Benötigt Administratorrechte und PostgreSQL 9.4+. Beachten Sie aber bitte, dass die Datenbank-, Sitzungs- und Benutzereinstellungen dies überschreiben.

```
ALTER SYSTEM SET postgis.gdal_enabled_drivers TO 'GTiff PNG JPEG';  
SELECT pg_reload_conf();
```

```
SET postgis.gdal_enabled_drivers TO 'GTiff PNG JPEG';  
SET postgis.gdal_enabled_drivers = default;
```

Aktiviert alle GDAL-Treiber

```
SET postgis.gdal_enabled_drivers = 'ENABLE_ALL';
```

Deaktiviert alle GDAL-Treiber

```
SET postgis.gdal_enabled_drivers = 'DISABLE_ALL';
```

Siehe auch

[ST_FromGDALRaster](#), [ST_AsGDALRaster](#), [ST_AsTIFF](#), [ST_AsPNG](#), [ST_AsJPEG](#), [postgis.enable_outdb_rasters](#)

8.2.4 postgis.enable_outdb_rasters

`postgis.enable_outdb_rasters` — Eine boolesche Konfigurationsmöglichkeit um den Zugriff auf out-db Rasterbänder zu ermöglichen

Beschreibung

Eine boolesche Konfigurationsmöglichkeit um den Zugriff auf out-db Rasterbänder zu ermöglichen. Diese Option kann in der PostgreSQL Konfigurationsdatei "postgresql.conf" gesetzt werden. Kann aber auch pro Verbindung oder pro Transaktion gesetzt werden.

Der Ausgangswert von `postgis.enable_outdb_rasters` kann auch beim Startprozess von PostgreSQL gesetzt werden, nämlich durch die Übergabe der Umgebungsvariablen `POSTGIS_ENABLE_OUTDB_RASTERS`, welche ungleich null sein muss.



Note

Auch wenn `postgis.enable_outdb_rasters` `True` ist, bestimmt die GUC `postgis.enable_outdb_rasters` die zugänglichen Rasterformate.



Note

Bei der Standardinstallation von PostGIS ist `postgis.enable_outdb_rasters` auf `False` gesetzt.

Verfügbarkeit: 2.2.0

Beispiele

`postgis.enable_outdb_rasters` setzen oder zurücksetzen

```
SET postgis.enable_outdb_rasters TO True;
SET postgis.enable_outdb_rasters = default;
SET postgis.enable_outdb_rasters = True;
SET postgis.enable_outdb_rasters = False;
```

Siehe auch

[postgis.gdal_enabled_drivers](#)

8.3 Funktionen zur Verwaltung von Geometrien

8.3.1 AddGeometryColumn

`AddGeometryColumn` — Fügt eine Geometriespalte an eine bestehende Tabelle an. Standardmäßig wird Typmodifikation anstelle von Bedingungen/Constraints verwendet. Sie erreichen das alte, auf check constraints basierte Verhalten, wenn Sie `use_typmod` auf `false` setzen.

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

Note



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

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

Note



Änderung: 2.0.0 Views können nicht mehr händisch in "geometry_columns" registriert werden. Views auf eine Geometrie in Typmod-Tabellen, bei denen keine Adapterfunktion verwendet wird, registrieren sich selbst auf korrekte Weise, da sie die Typmod-Verhaltensweise von der Spalte der Stammtabelle erben. Views die eine geometrische Funktion ausführen die eine andere Geometrie ausgibt, benötigen die Umwandlung in eine Typmod-Geometrie, damit die Geometrie des Views korrekt in "geometry_columns" registriert wird. Siehe Section 4.3.4.



This method implements the [OpenGIS Simple Features Implementation Specification for SQL 1.1](#).



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



This method supports Circular Strings and Curves

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

Beispiele

```
-- Ein Schema für die Daten erzeugen
CREATE SCHEMA my_schema;
-- Eine neue einfache PostgreSQL Tabelle erschellen
CREATE TABLE my_schema.my_spatial_table (id serial);

-- Die Beschreibung der Tabelle zeigt eine einfache Tabelle mit einer einzigen "id" Spalte ←
Describing the table shows a simple table with a single "id" column.
postgis=# \d my_schema.my_spatial_table
                                     Table "my_schema.my_spatial_table"
Column | Type      | Modifiers
-----+-----+-----
id      | integer   | not null default nextval('my_schema.my_spatial_table_id_seq'::regclass)

-- Fügt eine Geometriespalte an die Tabelle an
SELECT AddGeometryColumn ('my_schema', 'my_spatial_table', 'geom', 4326, 'POINT', 2);
```

```
-- Hinzufügen einer Punktgeometrie mit dem alten, auf Bedingungen basierten Verhalten/old
constraint behavior
SELECT AddGeometryColumn ('my_schema','my_spatial_table','geom_c',4326,'POINT',2, false);

--Hinzufügen eines Kurvenpolygons/curvepolygon mittels old constraint behavior
SELECT AddGeometryColumn ('my_schema','my_spatial_table','geomcp_c',4326,'CURVEPOLYGON',2,
false);

-- Die neuerliche Beschreibung der Tabelle zeigt die hinzugefügten Geometriespalten an.
\d my_schema.my_spatial_table
          addgeometrycolumn
-----
my_schema.my_spatial_table.geomcp_c SRID:4326 TYPE:CURVEPOLYGON DIMS:2
(1 row)
```

Column	Type	Table "my_schema.my_spatial_table"	Modifiers
id	integer		not null default nextval('my_schema.my_spatial_table_id_seq'::regclass)
geom	geometry(Point,4326)		
geom_c	geometry		
geomcp_c	geometry		

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

-- Der geometry_columns View registriert die neuen Spalten --
SELECT f_geometry_column As col_name, type, srid, coord_dimension As ndims
FROM geometry_columns
WHERE f_table_name = 'my_spatial_table' AND f_table_schema = 'my_schema';
```

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

Siehe auch

[DropGeometryColumn](#), [DropGeometryTable](#), [Section 4.3.2](#), [Section 4.3.4](#)

8.3.2 DropGeometryColumn

DropGeometryColumn — Entfernt eine Geometriespalte aus einer räumlichen Tabelle.

Synopsis

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

Beschreibung

Entfernt eine geometrische Spalte aus der Geometrietabelle. Der "schema_name" muss mit dem Feld "f_table_schema" in der Tabelle "geometry_columns" übereinstimmen.



This method implements the [OpenGIS Simple Features Implementation Specification for SQL 1.1](#).



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



This method supports Circular Strings and Curves



Note

Änderung: 2.0.0 Diese Funktion wurde zwecks Abwärtskompatibilität eingeführt. Seit geometry_columns ein View auf den Systemkatalog ist, können Sie die Geometriespalte, so wie jede andere Tabellenspalte, mit ALTER TABLE löschen.

Beispiele

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

-- In PostGIS 2.0+ entspricht das oben angeführte Aufruf ebenfalls dem Standard
-- Der standardmäßige ALTER TABLE Aufruf. Beide Aufrufe entfernen die Tabelle aus dem
  geometry_columns Register.
ALTER TABLE my_schema.my_spatial_table DROP column geom;
```

Siehe auch

[AddGeometryColumn](#), [DropGeometryTable](#), [Section 4.3.2](#)

8.3.3 DropGeometryTable

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

Synopsis

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

Beschreibung

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



Note

Änderung: 2.0.0 Diese Funktion wurde zwecks Abwärtskompatibilität eingeführt. Seit geometry_columns ein View auf den Systemkatalog ist, können Sie eine Tabelle mit einer Geometriespalte, so wie jede andere Tabelle, mit DROP TABLE löschen.

Beispiele

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

-- Obiges ist nun gleichbedeutend mit --
DROP TABLE my_schema.my_spatial_table;
```

Siehe auch

[AddGeometryColumn](#), [DropGeometryColumn](#), [Section 4.3.2](#)

8.3.4 PostGIS_Full_Version

PostGIS_Full_Version — Gibt einen vollständigen Bericht über die Version und Information über die Kompilationskonfiguration aus.

Synopsis

text **PostGIS_Full_Version**();

Beschreibung

Gibt einen vollständigen Versionsbericht und Information zur Kompilationskonfiguration aus. Informiert auch über die Synchronisation von Bibliotheken und Skripts und schlägt falls notwendig ein Upgrade vor.

Beispiele

```
SELECT PostGIS_Full_Version();

                                     postgis_full_version
-----
POSTGIS="2.2.0dev r12699" GEOS="3.5.0dev-CAPI-1.9.0 r3989" SFCGAL="1.0.4" PROJ="Rel. 4.8.0, ↵
6 March 2012"
GDAL="GDAL 1.11.0, released 2014/04/16" LIBXML="2.7.8" LIBJSON="0.12" RASTER
(1 row)
```

Siehe auch

[Section 2.10](#), [PostGIS_GEOS_Version](#), [PostGIS_Lib_Version](#), [PostGIS_LibXML_Version](#), [PostGIS_PROJ_Version](#), [PostGIS_Version](#)

8.3.5 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 GEOS nicht unterstützt wird.

Beispiele

```
SELECT PostGIS_GEOS_Version();
       postgis_geos_version
-----
3.1.0-CAPI-1.5.0
(1 row)
```

Siehe auch

[PostGIS_Full_Version](#), [PostGIS_Lib_Version](#), [PostGIS_LibXML_Version](#), [PostGIS_PROJ_Version](#), [PostGIS_Version](#)

8.3.6 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

Gibt die Versionsnummer der liblwgeom Bibliothek zurück/

Beispiele

```
SELECT PostGIS_Liblwgeom_Version();
       postgis_liblwgeom_version
-----
2.3.3 r15473
(1 row)
```

Siehe auch

[PostGIS_Full_Version](#), [PostGIS_Lib_Version](#), [PostGIS_LibXML_Version](#), [PostGIS_PROJ_Version](#), [PostGIS_Version](#)

8.3.7 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

```
SELECT PostGIS_LibXML_Version();
       postgis_libxml_version
-----
2.7.6
(1 row)
```

Siehe auch

[PostGIS_Full_Version](#), [PostGIS_Lib_Version](#), [PostGIS_PROJ_Version](#), [PostGIS_GEOS_Version](#), [PostGIS_Version](#)

8.3.8 PostGIS_Lib_Build_Date

PostGIS_Lib_Build_Date — Gibt das Kompilations-Datum der PostGIS Bibliothek zurück.

Synopsis

text **PostGIS_Lib_Build_Date**();

Beschreibung

Gibt das Kompilations-Datum der PostGIS Bibliothek zurück.

Beispiele

```
SELECT PostGIS_Lib_Build_Date();
       postgis_lib_build_date
-----
2008-06-21 17:53:21
(1 row)
```

8.3.9 PostGIS_Lib_Version

PostGIS_Lib_Version — Gibt die Versionsnummer der PostGIS Bibliothek aus.

Synopsis

text **PostGIS_Lib_Version**();

Beschreibung

Gibt die Versionsnummer der PostGIS Bibliothek aus.

Beispiele

```
SELECT PostGIS_Lib_Version();
 postgis_lib_version
-----
1.3.3
(1 row)
```

Siehe auch

[PostGIS_Full_Version](#), [PostGIS_GEOS_Version](#), [PostGIS_LibXML_Version](#), [PostGIS_PROJ_Version](#), [PostGIS_Version](#)

8.3.10 PostGIS_PROJ_Version

PostGIS_PROJ_Version — Gibt die Versionsnummer der Proj4 Bibliothek zurück.

Synopsis

text **PostGIS_PROJ_Version()**;

Beschreibung

Gibt die Versionsnummer der Proj4 Bibliothek aus, oder NULL wenn die PROJ4 Unterstützung nicht aktiviert ist.

Beispiele

```
SELECT PostGIS_PROJ_Version();
 postgis_proj_version
-----
Rel. 4.4.9, 29 Oct 2004
(1 row)
```

Siehe auch

[PostGIS_Full_Version](#), [PostGIS_GEOS_Version](#), [PostGIS_Lib_Version](#), [PostGIS_LibXML_Version](#), [PostGIS_Version](#)

8.3.11 PostGIS_Scripts_Build_Date

PostGIS_Scripts_Build_Date — Gibt das Kompilationsdatum der PostGIS Skripts aus.

Synopsis

text **PostGIS_Scripts_Build_Date()**;

Beschreibung

Gibt das Kompilationsdatum der PostGIS Skripts aus.

Verfügbarkeit: 1.0.0RC1

Beispiele

```
SELECT PostGIS_Scripts_Build_Date();
 postgis_scripts_build_date
-----
2007-08-18 09:09:26
(1 row)
```

Siehe auch

[PostGIS_Full_Version](#), [PostGIS_GEOS_Version](#), [PostGIS_Lib_Version](#), [PostGIS_LibXML_Version](#), [PostGIS_Version](#)

8.3.12 PostGIS_Scripts_Installed

PostGIS_Scripts_Installed — Gibt die Version der in der Datenbank installierten PostGIS Skripts aus.

Synopsis

text **PostGIS_Scripts_Installed()**;

Beschreibung

Gibt die Version der in der Datenbank installierten PostGIS Skripts aus.



Note

Wenn die Ausgabe dieser Funktion nicht mit der Ausgabe von [PostGIS_Scripts_Released](#) übereinstimmt, haben Sie vermutlich ein Upgrade auf eine bestehende Datenbank nicht richtig ausgeführt. Siehe Abschnitt [Upgrading](#) für weiterführende Information.

Verfügbarkeit: 0.9.0

Beispiele

```
SELECT PostGIS_Scripts_Installed();
 postgis_scripts_installed
-----
1.5.0SVN
(1 row)
```

Siehe auch

[PostGIS_Full_Version](#), [PostGIS_Scripts_Released](#), [PostGIS_Version](#)

8.3.13 PostGIS_Scripts_Released

PostGIS_Scripts_Released — Gibt die Versionsnummer des Skript "postgis.sql" aus, das mit der installierten PostGIS Bibliothek veröffentlicht wurde.

Synopsis

text **PostGIS_Scripts_Released()**;

Beschreibung

Gibt die Versionsnummer des Skript "postgis.sql" aus, das mit der installierten PostGIS Bibliothek veröffentlicht wurde.



Note

Ab der Version 1.1.0 gibt diese Funktion den selben Wert aus wie **PostGIS_Lib_Version**. Zwecks Abwärtskompatibilität beibehalten.

Verfügbarkeit: 0.9.0

Beispiele

```
SELECT PostGIS_Scripts_Released();
       postgis_scripts_released
-----
1.3.4SVN
(1 row)
```

Siehe auch

PostGIS_Full_Version, **PostGIS_Scripts_Installed**, **PostGIS_Lib_Version**

8.3.14 PostGIS_Version

PostGIS_Version — Gibt die Versionsnummer von PostGIS und die Optionen zur Kompilationzeit aus.

Synopsis

text **PostGIS_Version()**;

Beschreibung

Gibt die Versionsnummer von PostGIS und die Optionen zur Kompilationzeit aus.

Beispiele

```
SELECT PostGIS_Version();
               postgis_version
-----
1.3 USE_GEOS=1 USE_PROJ=1 USE_STATS=1
(1 row)
```

Siehe auch

PostGIS_Full_Version, **PostGIS_GEOS_Version**, **PostGIS_Lib_Version**, **PostGIS_LibXML_Version**, **PostGIS_PROJ_Version**

8.3.15 Populate_Geometry_Columns

Populate_Geometry_Columns — 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`

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 in der Tabelle `geometry_columns` sichergestellt.

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

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

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

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

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

Verfügbarkeit: 1.4.0

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

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

Beispiele

```
CREATE TABLE public.myspatial_table(gid serial, geom geometry);
INSERT INTO myspatial_table(geom) VALUES(ST_GeomFromText('LINESTRING(1 2, 3 4)',4326) );
-- Hier werden nun Typmodifikatoren verwendet. Damit dies funktioniert, müssen Daten ↵
    vorhanden sein
SELECT Populate_Geometry_Columns('public.myspatial_table'::regclass);

populate_geometry_columns
-----
1

\d myspatial_table

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

-- Dies stellt die Geometriespalten auf die Verwendung von Constraints um. Allerdings nur, ↵
    wenn sie sich nicht in typmod befinden oder nicht bereits Constraints aufweisen.
-- Damit dies funktioniert müssen Daten vorhanden sein
CREATE TABLE public.myspatial_table_cs(gid serial, geom geometry);
INSERT INTO myspatial_table_cs(geom) VALUES(ST_GeomFromText('LINESTRING(1 2, 3 4)',4326) );
SELECT Populate_Geometry_Columns('public.myspatial_table_cs'::regclass, false);
populate_geometry_columns
-----
1

\d myspatial_table_cs

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

8.3.16 UpdateGeometrySRID

UpdateGeometrySRID — Erneuert die SRID aller Features einer Geometriespalte, die Metadaten und die SRID in "geometry_columns". Wenn Constraints erzwungen wurden, werden die Constraints mit der neuen SRID versehen. Falls die Definition über den Datentyp erfolgte, wird die Datentypdefinition geändert.

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.



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



This method supports Circular Strings and Curves

Beispiele

Ändert die SRID der Straßentabelle auf 4326

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

Das vorhergegangene Beispiel ist gleichbedeutend mit dieser DDL Anweisung

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

Falls Sie sich in der Projektion geirrt haben (oder sie als "unknown" importiert haben) und sie in einem Aufwaschen in die Web Mercator Projektion transformieren wollen, so können Sie dies mit DDL bewerkstelligen. Es gibt jedoch keine äquivalente PostGIS Managementfunktion, die dies in einem Schritt bewerkstelligen könnte.

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

Siehe auch

[UpdateRasterSRID](#), [ST_SetSRID](#), [ST_Transform](#)

8.4 Geometrische Konstruktoren

8.4.1 ST_BdPolyFromText

ST_BdPolyFromText — Konstruiert ein Polygon aus einer beliebigen Ansammlung von geschlossenen Linienzügen, welche als MultiLineString in der Well-Known Text Darstellung vorliegen müssen.

Synopsis

geometry **ST_BdPolyFromText**(text WKT, integer srid);

Beschreibung

Konstruiert ein Polygon aus einer beliebigen Ansammlung von geschlossenen Linienzügen, welche als MultiLineString in der Well-Known Text Darstellung vorliegen müssen.

**Note**

Meldet einen Fehler, wenn es sich bei dem WKT nicht um einen MULTILINESTRING handelt. Meldet einen Fehler, wenn die Ausgabe ein MULTIPOLYGON ist; in diesem Fall verwenden Sie bitte `ST_BdMPolyFromText`, oder vergleichen `ST_BuildArea()` für einen PostGIS orientierten Ansatz.



This method implements the [OpenGIS Simple Features Implementation Specification for SQL 1.1](#). s3.2.6.2

Verfügbarkeit: 1.1.0 - benötigt GEOS >= 2.1.0.

Beispiele

Kommt noch

Siehe auch

[ST_BuildArea](#), [ST_BdMPolyFromText](#)

8.4.2 ST_BdMPolyFromText

`ST_BdMPolyFromText` — Konstruiert ein MultiPolygon aus einer beliebigen Ansammlung von geschlossenen Linienzügen, welche als MultiLineString in der Well-Known Text Darstellung vorliegen müssen.

Synopsis

geometry **ST_BdMPolyFromText**(text WKT, integer srid);

Beschreibung

Konstruiert ein Polygon aus einer beliebigen Ansammlung von geschlossenen Linienzügen, Polygonen und MultiLineStrings, welche in der Well-Known Text Darstellung vorliegen müssen.

**Note**

Meldet einen Fehler wenn der WKT kein MULTILINESTRING ist. Erzwingt die MULTIPOLYGON Ausgabe sogar dann, wenn das Ergebnis nur aus einem einzelnen POLYGON besteht; verwenden Sie bitte [ST_BdPolyFromText](#) , wenn Sie sicher sind, daß nur ein einzelnes POLYGON entsteht, oder vergleichen Sie [ST_BuildArea\(\)](#) für einen PostGIS orientierten Ansatz.



This method implements the [OpenGIS Simple Features Implementation Specification for SQL 1.1](#). s3.2.6.2

Verfügbarkeit: 1.1.0 - benötigt GEOS >= 2.1.0.

Beispiele

Kommt noch

Siehe auch

[ST_BuildArea](#), [ST_BdPolyFromText](#)

8.4.3 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 `precision` angegeben wird, dann gibt ST_Box2dFromGeoHash eine BOX2D zurück, die auf der vollständigen Genauigkeit der GeoHash Zeichenfolge beruht.

Wenn `precision` angegeben wird, verwendet ST_Box2dFromGeoHash entsprechend viele Zeichen des GeoHash um die BOX2D zu erzeugen. Niedrigere Werte erzeugen eine größere BOX2D und höhere Werte erhöhen die Genauigkeit.

Verfügbarkeit: 2.1.0

Beispiele

```
SELECT ST_Box2dFromGeoHash('9qqj7nmxnccgyy4d0dbxqz0');

          st_geomfromgeohash
-----
BOX(-115.172816 36.114646,-115.172816 36.114646)

SELECT ST_Box2dFromGeoHash('9qqj7nmxnccgyy4d0dbxqz0', 0);

          st_box2dfromgeohash
-----
BOX(-180 -90,180 90)

SELECT ST_Box2dFromGeoHash('9qqj7nmxnccgyy4d0dbxqz0', 10);
          st_box2dfromgeohash
-----
BOX(-115.17282128334 36.1146408319473,-115.172810554504 36.1146461963654)
```

Siehe auch

[ST_GeoHash](#), [ST_GeomFromGeoHash](#), [ST_PointFromGeoHash](#)

8.4.4 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

```
--- Umwandlung von Länge Breite Koordinaten in Geographie
ALTER TABLE sometable ADD COLUMN geog geography(POINT,4326);
UPDATE sometable SET geog = ST_GeogFromText('SRID=4326;POINT(' || lon || ' ' || lat || ')') ←
;

--- Definition eines geographischen Punktes mit EPSG:4267, NAD27
SELECT ST_AsEWKT(ST_GeogFromText('SRID=4267;POINT(-77.0092 38.889588)'));
```

Siehe auch

[ST_AsText](#), [ST_GeographyFromText](#)

8.4.5 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](#)

8.4.6 ST_GeogFromWKB

`ST_GeogFromWKB` — Erzeugt ein geographisches Objekt aus der Well-known-Binary (WKB) oder der erweiterten Well-known-Binary (EWKB) Darstellung.

Synopsis

`geography ST_GeogFromWKB(bytea wkb);`

Beschreibung

Die Funktion `ST_GeogFromWKB` empfängt eine Well-known-Binary (WKB) oder eine erweiterte PostGIS WKB (EWKB) Darstellung einer Geometrie und erzeugt eine Instanz des entsprechenden geographischen Datentyps. Diese Funktion übernimmt die Rolle der Geometrie-Factory/Fabrik in SQL.

Wenn die SRID nicht festgelegt ist, wird 4326 (WGS 84) angenommen.



This method supports Circular Strings and Curves

Beispiele

```
--Obwohl die BYTEA Darstellung einzelne "\"" enthält, müssen diese beim Einfügen in eine
Tabelle maskiert werden
SELECT ST_AsText(
ST_GeogFromWKB(E'\\001\\002\\000\\000\\000\\002\\000\\000\\000\\037\\205\\353Q
\\270~\\\\\\\\300\\323Mb\\020X\\231C@\\020X9\\264\\310~\\\\\\\\300)\\\\\\\\217\\302\\365\\230
C@')
);
-----st_astext-----
LINESTRING(-113.98 39.198,-113.981 39.195)
(1 row)
```

Siehe auch

[ST_GeogFromText](#), [ST_AsBinary](#)

8.4.7 ST_GeomFromTWKB

`ST_GeomFromTWKB` — Erzeugt eine Geometrie aus einer TWKB ("[Tiny Well-Known Binary](#)") Darstellung.

Synopsis

geometry `ST_GeomFromTWKB`(bytea twkb);

Beschreibung

Die Funktion `ST_GeomFromTWKB` nimmt eine TWKB ("[Tiny Well-Known Binary](#)") Darstellung und erzeugt ein Objekt mit dem entsprechenden geometrischen Datentyp.

Beispiele

```
SELECT ST_AsText(ST_GeomFromTWKB(ST_AsTWKB('LINESTRING(126 34, 127 35)::geometry')));
-----st_astext-----
LINESTRING(126 34, 127 35)
(1 row)

SELECT ST_AsEWKT(
ST_GeomFromTWKB(E'\\x620002f7f40dbce4040105')
```

```
);
----- st_asewkt -----
LINESTRING(-113.98 39.198,-113.981 39.195)
(1 row)
```

Siehe auch

[ST_AsTWKB](#)

8.4.8 ST_GeomCollFromText

ST_GeomCollFromText — Erzeugt eine Sammelgeometrie mit der gegebenen SRID aus einer WKT-Kollektion. Wenn keine SRID angegeben ist, wird diese standardmäßig auf 0 gesetzt.

Synopsis

```
geometry ST_GeomCollFromText(text WKT, integer srid);
geometry ST_GeomCollFromText(text WKT);
```

Beschreibung

Erzeugt eine Sammelgeometrie mit der gegebenen SRID aus einer Well-known-Text (WKT) Darstellung. Wenn keine SRID angegeben ist, wird diese standardmäßig auf 0 gesetzt.

OGC SPEC 3.2.6.2 - Die Option SRID stammt aus dem Konformitätstest

Gibt NULL zurück, wenn der WKT keine GEOMETRYCOLLECTION ist



Note

Verwenden Sie diese Funktion nicht, wenn Sie sich vollkommen sicher sind, dass ihre WKT Geometrie eine Sammelgeometrie ist. Sie ist langsamer als ST_GeomFromText, da sie einen zusätzlichen Validierungsschritt ausführt.



This method implements the [OpenGIS Simple Features Implementation Specification for SQL 1.1](#). s3.2.6.2



This method implements the SQL/MM specification.

Beispiele

```
SELECT ST_GeomCollFromText('GEOMETRYCOLLECTION(POINT(1 2),LINESTRING(1 2, 3 4))');
```

Siehe auch

[ST_GeomFromText](#), [ST_SRID](#)

8.4.9 ST_GeomFromEWKB

ST_GeomFromEWKB — Gibt einen geometrischen Datentyp (ST_Geometry) aus einer Well-known-Binary (WKB) Darstellung zurück.

Synopsis

geometry **ST_GeomFromEWKB**(bytea EWKB);

Beschreibung

Erzeugt ein PostGIS ST_Geometry Objekt aus der erweiterten OGC Well-known-Text (EWKT) Darstellung.



Note

EWKB ist kein Format des OGC Standards, sondern ein PostGIS eigenes Format, welches den Identifikator (SRID) des räumlichen Koordinatenreferenzsystem mit einbindet

Erweiterung: Mit 2.0.0 wurde die Unterstützung für polyedrische Oberflächen und TIN eingeführt.



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



This method supports Circular Strings and Curves



This function supports Polyhedral surfaces.



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

Beispiele

Binärdarstellung des Linienzuges `LINESTRING(-71.160281 42.258729,-71.160837 42.259113,-71.161144 42.25932)` in NAD 83 (4269).



Note

ANMERKUNG: Obwohl die Bytefelder durch \ getrennt sind und auch ' aufweisen können, müssen wir beides mit \ maskieren; wenn "standard_conforming_strings" ausgeschaltet ist mit ". Somit sieht diese Darstellung nicht genauso wie die AsEWKB Darstellung aus.

```
SELECT ST_GeomFromEWKB(E'\001\002\000\000 \255\020\000\000\003\000\000\000\344 ←
J=
\013B\312Q\300n\303(\010\036!E@'\277E'K
\312Q\300\366{b\235*!E@\225|\354.P\312Q
\300p\231\323e1!E@');
```



Note

Ab PostgreSQL 9.1 - ist standard_conforming_strings standardmäßig auf "on" gesetzt. Bei Vorgängerversionen war es "off". Sie können die Standardvorgaben für eine einzelne Abfrage ändern oder auf Datenbank- oder Serverebene setzen. Unterhalb steht, wie Sie dies mit standard_conforming_strings = on umsetzen können. In diesem Fall maskieren wir das ' mit dem ANSI Zeichen ', aber Schrägstriche werden nicht maskiert

```
set standard_conforming_strings = on;
SELECT ST_GeomFromEWKB('001\002\000\000 \255\020\000\000\003\000\000\000\344J=\012\013B
\312Q\300n\303(\010\036!E@'\277E'K\012\312Q\300\366{b\235*!E@\225|\354.P\312Q\012\300 ←
p\231\323e1')
```

Siehe auch

[ST_AsBinary](#), [ST_AsEWKB](#), [ST_GeomFromWKB](#)

8.4.10 ST_GeomFromEWKT

ST_GeomFromEWKT — Gibt einen spezifizierten ST_Geometry-Wert von einer erweiterten Well-known-Text Darstellung (EWKT) zurück.

Synopsis

geometry **ST_GeomFromEWKT**(text EWKT);

Beschreibung

Erzeugt ein PostGIS ST_Geometry Objekt aus der erweiterten OGC Well-known-Text (EWKT) Darstellung.

**Note**

EWKT ist kein Format des OGC Standards, sondern ein PostGIS eigenes Format, welches den Identifikator (SRID) des räumlichen Koordinatenreferenzsystem mit einbindet

Erweiterung: Mit 2.0.0 wurde die Unterstützung für polyedrische Oberflächen und TIN eingeführt.



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



This method supports Circular Strings and Curves



This function supports Polyhedral surfaces.



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

Beispiele

```
SELECT ST_GeomFromEWKT('SRID=4269;LINESTRING(-71.160281 42.258729,-71.160837 ↵
  42.259113,-71.161144 42.25932)');
SELECT ST_GeomFromEWKT('SRID=4269;MULTILINESTRING((-71.160281 42.258729,-71.160837 ↵
  42.259113,-71.161144 42.25932))');

SELECT ST_GeomFromEWKT('SRID=4269;POINT(-71.064544 42.28787)');

SELECT ST_GeomFromEWKT('SRID=4269;POLYGON((-71.1776585052917 ↵
  42.3902909739571,-71.1776820268866 42.3903701743239,
-71.1776063012595 42.3903825660754,-71.1775826583081 42.3903033653531,-71.1776585052917 ↵
  42.3902909739571))');

SELECT ST_GeomFromEWKT('SRID=4269;MULTIPOLYGON((( -71.1031880899493 42.3152774590236,
-71.1031627617667 42.3152960829043,-71.102923838298 42.3149156848307,
-71.1023097974109 42.3151969047397,-71.1019285062273 42.3147384934248,
-71.102505233663 42.3144722937587,-71.10277487471 42.3141658254797,
-71.103113945163 42.3142739188902,-71.10324876416 42.31402489987,
-71.1033002961013 42.3140393340215,-71.1033488797549 42.3139495090772,
```

```
-71.103396240451 42.3138632439557,-71.1041521907712 42.3141153348029,
-71.1041411411543 42.3141545014533,-71.1041287795912 42.3142114839058,
-71.1041188134329 42.3142693656241,-71.1041112482575 42.3143272556118,
-71.1041072845732 42.3143851580048,-71.1041057218871 42.3144430686681,
-71.1041065602059 42.3145009876017,-71.1041097995362 42.3145589148055,
-71.1041166403905 42.3146168544148,-71.1041258822717 42.3146748022936,
-71.1041375307579 42.3147318674446,-71.1041492906949 42.3147711126569,
-71.1041598612795 42.314808571739,-71.1042515013869 42.3151287620809,
-71.1041173835118 42.3150739481917,-71.1040809891419 42.3151344119048,
-71.1040438678912 42.3151191367447,-71.1040194562988 42.3151832057859,
-71.1038734225584 42.3151140942995,-71.1038446938243 42.3151006300338,
-71.1038315271889 42.315094347535,-71.1037393329282 42.315054824985,
-71.1035447555574 42.3152608696313,-71.1033436658644 42.3151648370544,
-71.1032580383161 42.3152269126061,-71.103223066939 42.3152517403219,
-71.1031880899493 42.3152774590236)),
((-71.1043632495873 42.315113108546,-71.1043583974082 42.3151211109857,
-71.1043443253471 42.3150676015829,-71.1043850704575 42.3150793250568,-71.1043632495873 ↔
42.315113108546)))');
```

```
--3D Kreisbogen
SELECT ST_GeomFromEWKT('CIRCULARSTRING(220268 150415 1,220227 150505 2,220227 150406 3)');
```

```
--Beispiel für eine polyedrische Oberfläche
SELECT ST_GeomFromEWKT('POLYHEDRALSURFACE(
    ((0 0 0, 0 0 1, 0 1 1, 0 1 0, 0 0 0)),
    ((0 0 0, 0 1 0, 1 1 0, 1 0 0, 0 0 0)),
    ((0 0 0, 1 0 0, 1 0 1, 0 0 1, 0 0 0)),
    ((1 1 0, 1 1 1, 1 0 1, 1 0 0, 1 1 0)),
    ((0 1 0, 0 1 1, 1 1 1, 1 1 0, 0 1 0)),
    ((0 0 1, 1 0 1, 1 1 1, 0 1 1, 0 0 1))
)');
```

Siehe auch

[ST_AsEWKT](#), [ST_GeomFromText](#), [ST_GeomFromEWKT](#)

8.4.11 ST_GeometryFromText

ST_GeometryFromText — Gibt einen spezifizierten ST_Geometry-Wert von einer Well-known-Text Darstellung (WKT) zurück. Die Bezeichnung ist ein Alias für [ST_GeomFromText](#)

Synopsis

```
geometry ST_GeometryFromText(text WKT);
geometry ST_GeometryFromText(text WKT, integer srid);
```

Beschreibung



This method implements the [OpenGIS Simple Features Implementation Specification for SQL 1.1](#).



This method implements the SQL/MM specification. SQL-MM 3: 5.1.40

Siehe auch[ST_GeomFromText](#)**8.4.12 ST_GeomFromGeoHash**

ST_GeomFromGeoHash — Gibt die Geometrie einer GeoHash Zeichenfolge zurück.

Synopsis

```
geometry ST_GeomFromGeoHash(text geohash, integer precision=full_precision_of_geohash);
```

Beschreibung

Gibt die Geometrie einer GeoHash Zeichenfolge zurück. Der geometrische Datentyp ist ein Polygon, das den GeoHash begrenzt.

Wenn keine `precision` angegeben wird, dann gibt ST_GeomFromGeoHash ein Polygon zurück, das auf der vollständigen Genauigkeit der GeoHash Zeichenfolge beruht.

Wenn `precision` angegeben wird, verwendet ST_GeomFromGeoHash entsprechend viele Zeichen des GeoHash, um das Polygon zu erzeugen.

Verfügbarkeit: 2.1.0

Beispiele

```
SELECT ST_AsText(ST_GeomFromGeoHash('9qqj7nmxnccgyy4d0dbxqz0'));
           st_astext
-----
POLYGON((-115.172816 36.114646,-115.172816 36.114646,-115.172816 36.114646,-115.172816 36.114646,-115.172816 36.114646))

SELECT ST_AsText(ST_GeomFromGeoHash('9qqj7nmxnccgyy4d0dbxqz0', 4));
           st_astext
-----
POLYGON((-115.3125 36.03515625,-115.3125 36.2109375,-114.9609375 36.2109375,-114.9609375 36.03515625,-115.3125 36.03515625))

SELECT ST_AsText(ST_GeomFromGeoHash('9qqj7nmxnccgyy4d0dbxqz0', 10));
           st_astext
-----
POLYGON((-115.17282128334 36.1146408319473,-115.17282128334 36.1146461963654,-115.172810554504 36.1146461963654,-115.172810554504 36.1146408319473,-115.17282128334 36.1146408319473))
```

Siehe auch

[ST_GeoHash](#), [ST_Box2dFromGeoHash](#), [ST_PointFromGeoHash](#)

8.4.13 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 Rückwärtskompatibilität)
- GML 2.1.2

OGC GML Standards, vgl.: <http://www.opengeospatial.org/standards/gml>:

Verfügbarkeit: 1.5, benötigt libxml2 1.6+

Erweiterung: Mit 2.0.0 wurde die Unterstützung für polyedrische Oberflächen und TIN eingeführt.

Erweiterung: 2.0.0 Standardwert für den optionalen Parameter SRID eingefügt.



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



This function supports Polyhedral surfaces.



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

GML erlaubt das Mischen von Dimensionen (z.B. 2D und 3D innerhalb der selben MultiGeometry). Da PostGIS Geometrien dies nicht zulassen, wandelt ST_GeomFromGML die gesamte Geometrie in 2D um, sobald eine fehlende Z-Dimension existiert.

GML unterstützt uneinheitliche Koordinatenreferenzsysteme innerhalb derselben Mehrfachgeometrie. Da dies der geometrische Datentyp von PostGIS nicht unterstützt, wird in diesem Fall die Subgeometrie in das Referenzsystem des Knotens, der die Wurzel darstellt, umprojiziert. Wenn kein Attribut "srsName" für den Knoten der GML-Wurzel vorhanden ist, gibt die Funktion eine Fehlermeldung aus.

Die Funktion ST_GeomFromGML ist nicht kleinlich, was die explizite Vergabe eines GML-Namensraums betrifft. Bei üblichen Anwendungen können Sie die explizite Vergabe weglassen. Wenn Sie aber das XLink Feature von GML verwenden wollen, müssen Sie den Namensraum explizit angeben.



Note

SQL/MM Kurvengeometrien werden von der Funktion ST_GeomFromGML nicht unterstützt

Beispiele - Eine Einzelgeometrie mit einem srsName

```
SELECT ST_GeomFromGML('
    <gml:LineString srsName="EPSG:4269">
      <gml:coordinates>
        -71.16028,42.258729 -71.160837,42.259112 ↔
        -71.161143,42.25932
      </gml:coordinates>
    </gml:LineString
>');
```

Beispiele - Verwendung von XLink

```
SELECT ST_GeomFromGML('
    <gml:LineString xmlns:gml="http://www.opengis.net/gml"
      xmlns:xlink="http://www.w3.org/1999/xlink"
      srsName="urn:ogc:def:crs:EPSG::4269">
      <gml:pointProperty>
        <gml:Point gml:id="p1"
><gml:pos
>42.258729 -71.16028</gml:pos
></gml:Point>
        </gml:pointProperty>
        <gml:pos
>42.259112 -71.160837</gml:pos>
        <gml:pointProperty>
          <gml:Point xlink:type="simple" xlink:href="#p1"/>
        </gml:pointProperty>
      </gml:LineString
>'););
```

Beispiele - polyedrische Oberfläche

```
SELECT ST_AsEWKT(ST_GeomFromGML('
<gml:PolyhedralSurface>
<gml:polygonPatches>
  <gml:PolygonPatch>
    <gml:exterior>
      <gml:LinearRing
><gml:posList srsDimension="3"
>0 0 0 0 0 1 0 1 1 0 1 0 0 0 0</gml:posList
></gml:LinearRing>
      </gml:exterior>
    </gml:PolygonPatch>
    <gml:PolygonPatch>
      <gml:exterior>
        <gml:LinearRing
><gml:posList srsDimension="3"
>0 0 0 0 1 0 1 1 0 1 0 0 0 0 0</gml:posList
></gml:LinearRing>
        </gml:exterior>
      </gml:PolygonPatch>
    <gml:PolygonPatch>
      <gml:exterior>
        <gml:LinearRing
><gml:posList srsDimension="3"
>0 0 0 1 0 0 1 0 1 0 0 1 0 0 0</gml:posList
></gml:LinearRing>
```

```

    </gml:exterior>
  </gml:PolygonPatch>
<gml:PolygonPatch>
  <gml:exterior>
    <gml:LinearRing
><gml:posList srsDimension="3"
>1 1 0 1 1 1 1 0 1 1 0 0 1 1 0</gml:posList
></gml:LinearRing>
    </gml:exterior>
  </gml:PolygonPatch>
<gml:PolygonPatch>
  <gml:exterior>
    <gml:LinearRing
><gml:posList srsDimension="3"
>0 1 0 0 1 1 1 1 1 1 0 0 1 0</gml:posList
></gml:LinearRing>
    </gml:exterior>
  </gml:PolygonPatch>
<gml:PolygonPatch>
  <gml:exterior>
    <gml:LinearRing
><gml:posList srsDimension="3"
>0 0 1 1 0 1 1 1 1 0 1 1 0 0 1</gml:posList
></gml:LinearRing>
    </gml:exterior>
  </gml:PolygonPatch>
</gml:polygons>
</gml:PolyhedralSurface
>'))';

-- result --
POLYHEDRALSURFACE(((0 0 0,0 0 1,0 1 1,0 1 0,0 0 0)),
((0 0 0,0 1 0,1 1 0,1 0 0,0 0 0)),
((0 0 0,1 0 0,1 0 1,0 0 1,0 0 0)),
((1 1 0,1 1 1,1 0 1,1 0 0,1 1 0)),
((0 1 0,0 1 1,1 1 1,1 1 0,0 1 0)),
((0 0 1,1 0 1,1 1 1,0 1 1,0 0 1)))

```

Siehe auch

Section [2.4.1](#), [ST_AsGML](#), [ST_GMLToSQL](#)

8.4.14 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);

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.

Verfügbarkeit: 2.0.0 benötigt - JSON-C >= 0.9

**Note**

Wenn Sie die JSON-C Unterstützung nicht aktiviert haben, sehen Sie eine Fehlermeldung anstatt einer Ausgabe. Um JSON-C zu aktivieren, führen Sie bitte `configure --with-jsondir=/path/to/json-c` aus. Für Einzelheiten siehe Section [2.4.1](#).



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

Beispiele

```
SELECT ST_AsText(ST_GeomFromGeoJSON('{"type":"Point","coordinates":[-48.23456,20.12345]}')) ←
      As wkt;
wkt
-----
POINT(-48.23456 20.12345)
```

```
-- ein 3D Polygonzug
SELECT ST_AsText(ST_GeomFromGeoJSON('{"type":"LineString","coordinates ←
      ":[1,2,3],[4,5,6],[7,8,9]}')) As wkt;
wkt
-----
LINESTRING(1 2,4 5,7 8)
```

Siehe auch

[ST_AsText](#), [ST_AsGeoJSON](#), Section [2.4.1](#)

8.4.15 ST_GeomFromKML

ST_GeomFromKML — Nimmt als Eingabe eine KML-Darstellung der Geometrie und gibt ein geometrisches PostGIS-Objekt aus.

Synopsis

geometry **ST_GeomFromKML**(text geomkml);

Beschreibung

Erzeugt ein PostGIS ST_Geometry Objekt aus der OGC KML Darstellung.

T_GeomFromKML funktioniert nur bei Fragmenten von KML-Geometrien. Auf das ganze KML-Dokument angewendet führt zu einer Fehlermeldung.

Unterstützte OGC KML Versionen:

- KML 2.2.0 Namespace

OGC KML Standards, vgl.: <http://www.opengeospatial.org/standards/kml>:

Verfügbarkeit: 1.5, benötigt libxml2 2.6+



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

**Note**

SQL/MM Kurvengeometrien werden von der Funktion ST_GeomFromKML nicht unterstützt

Beispiele - Eine Einzelgeometrie mit einem srsName

```
SELECT ST_GeomFromKML('
    <LineString>
      <coordinates>
>-71.1663,42.2614
                                -71.1667,42.2616</coordinates>
      </LineString>
>');
```

Siehe auch

Section [2.4.1](#), [ST_AsKML](#)

8.4.16 ST_GMLToSQL

ST_GMLToSQL — Gibt einen spezifizierten ST_Geometry Wert aus einer GML-Darstellung zurück. Dies ist ein Aliasname für ST_GeomFromGML

Synopsis

```
geometry ST_GMLToSQL(text geomgml);
geometry ST_GMLToSQL(text geomgml, integer srid);
```

Beschreibung

This method implements the SQL/MM specification. SQL-MM 3: 5.1.50 (ausgenommen Unterstützung von Kurven).

Verfügbarkeit: 1.5, benötigt libxml2 1.6+

Erweiterung: Mit 2.0.0 wurde die Unterstützung für polyedrische Oberflächen und TIN eingeführt.

Erweiterung: 2.0.0 Standardwert für den optionalen Parameter SRID eingefügt.

Siehe auch

Section [2.4.1](#), [ST_GeomFromGML](#), [ST_AsGML](#)

8.4.17 ST_GeomFromText

ST_GeomFromText — Gibt einen spezifizierten ST_Geometry Wert aus einer Well-known-Text Darstellung (WKT) zurück.

Synopsis

```
geometry ST_GeomFromText(text WKT);
geometry ST_GeomFromText(text WKT, integer srid);
```

Beschreibung

Erzeugt ein PostGIS ST_Geometry Objekt aus der OGC Well-known-Text Darstellung.



Note

Die Funktion ST_GeomFromText hat zwei Varianten. Die erste Variante nimmt keine SRID entgegen und gibt eine Geometrie ohne ein bestimmtes Koordinatenreferenzsystem aus (SRID=0) . Die zweite Variante nimmt eine SRID als zweiten Übergabewert entgegen und gibt eine Geometrie zurück, die diese SRID als Teil ihrer Metadaten beinhaltet.



This method implements the [OpenGIS Simple Features Implementation Specification for SQL 1.1](#). s3.2.6.2 - die Option SRID ist vom Konformitätstest.



This method implements the SQL/MM specification. SQL-MM 3: 5.1.40



This method supports Circular Strings and Curves



Warning

Änderung: 2.0.0 - In Vorgängerversionen von PostGIS war ST_GeomFromText('GEOMETRYCOLLECTION(EMPTY)') erlaubt. Um eine bessere Übereinstimmung mit der SQL/MM Norm zu erreichen, ist dies in PostGIS 2.0.0 nun nicht mehr gestattet. Hier sollte nun ST_GeomFromText('GEOMETRYCOLLECTION EMPTY') geschrieben werden.

Beispiele

```
SELECT ST_GeomFromText('LINESTRING(-71.160281 42.258729,-71.160837 42.259113,-71.161144 42.25932)');
SELECT ST_GeomFromText('LINESTRING(-71.160281 42.258729,-71.160837 42.259113,-71.161144 42.25932)',4269);

SELECT ST_GeomFromText('MULTILINESTRING((-71.160281 42.258729,-71.160837 42.259113,-71.161144 42.25932))');

SELECT ST_GeomFromText('POINT(-71.064544 42.28787)');

SELECT ST_GeomFromText('POLYGON((-71.1776585052917 42.3902909739571,-71.1776820268866 42.3903701743239,
-71.1776063012595 42.3903825660754,-71.1775826583081 42.3903033653531,-71.1776585052917 42.3902909739571))');

SELECT ST_GeomFromText('MULTIPOLYGON((( -71.1031880899493 42.3152774590236,
-71.1031627617667 42.3152960829043,-71.102923838298 42.3149156848307,
-71.1023097974109 42.3151969047397,-71.1019285062273 42.3147384934248,
-71.102505233663 42.3144722937587,-71.10277487471 42.3141658254797,
-71.103113945163 42.3142739188902,-71.10324876416 42.31402489987,
-71.1033002961013 42.3140393340215,-71.1033488797549 42.3139495090772,
-71.103396240451 42.3138632439557,-71.1041521907712 42.3141153348029,
-71.1041411411543 42.3141545014533,-71.1041287795912 42.3142114839058,
-71.1041188134329 42.3142693656241,-71.1041112482575 42.3143272556118,
-71.1041072845732 42.3143851580048,-71.1041057218871 42.3144430686681,
-71.1041065602059 42.3145009876017,-71.1041097995362 42.3145589148055,
-71.1041166403905 42.3146168544148,-71.1041258822717 42.3146748022936,
-71.1041375307579 42.3147318674446,-71.1041492906949 42.3147711126569,
-71.1041598612795 42.314808571739,-71.1042515013869 42.3151287620809,
-71.1041173835118 42.3150739481917,-71.1040809891419 42.3151344119048,
-71.1040438678912 42.3151191367447,-71.1040194562988 42.3151832057859,
```

```
-71.1038734225584 42.3151140942995,-71.1038446938243 42.3151006300338,
-71.1038315271889 42.315094347535,-71.1037393329282 42.315054824985,
-71.1035447555574 42.3152608696313,-71.1033436658644 42.3151648370544,
-71.1032580383161 42.3152269126061,-71.103223066939 42.3152517403219,
-71.1031880899493 42.3152774590236)),
((-71.1043632495873 42.315113108546,-71.1043583974082 42.3151211109857,
-71.1043443253471 42.3150676015829,-71.1043850704575 42.3150793250568,-71.1043632495873  ←
  42.315113108546))) ',4326);

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

Siehe auch

[ST_GeomFromEWKT](#), [ST_GeomFromWKB](#), [ST_SRID](#)

8.4.18 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_GeomFromWKB** nimmt eine Well-known-Binary (WKB) Darstellung und eine Id für das Koordinatenreferenzsystem (SRID) entgegen und erzeugt eine Instanz des entsprechenden geometrischen Datentyps. Diese Funktion übernimmt die Rolle der Geometrie-Factory in SQL. Ist eine alternative Bezeichnung für **ST_WKBToSQL**.

Wenn die SRID nicht festgelegt ist, wird sie standardmäßig auf 0 (Unknown) gesetzt.



This method implements the [OpenGIS Simple Features Implementation Specification for SQL 1.1](#). s3.2.7.2 - die optionale SRID kommt vom Konformitätstest.



This method implements the SQL/MM specification. SQL-MM 3: 5.1.41



This method supports Circular Strings and Curves

Beispiele

```
--Obwohl die BYTEA Darstellung einzelne "\" enthält, müssen diese beim Einfügen in eine  ←
  Tabelle maskiert werden
      -- ausgenommen standard_conforming_strings ist auf "on" gesetzt.
SELECT ST_AsEWKT(
ST_GeomFromWKB(E'\001\002\000\000\000\002\000\000\000\037\205\353Q  ←
  \270~\000\300\323Mb\020X\231C@020X9\264\310~\000\300)\000\217\302\365\230  ←
  C@',4326)
);

                                st_asewkt
-----
SRID=4326;LINESTRING(-113.98 39.198,-113.981 39.195)
```

```
(1 row)

SELECT
  ST_AsText (
    ST_GeomFromWKB (
      ST_AsEWKB ('POINT(2 5) '::geometry)
    )
  );
 st_astext
-----
POINT(2 5)
(1 row)
```

Siehe auch

[ST_WKBToSQL](#), [ST_AsBinary](#), [ST_GeomFromEWKB](#)

8.4.19 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.

Siehe <http://developers.google.com/maps/documentation/utilities/polylinealgorithm>

Verfügbarkeit: 2.2.0

Beispiele

```
--Erzeugung eines Linienzuges aus einer Polyline
SELECT ST_AsEWKT(ST_LineFromEncodedPolyline('_p~iF~ps|U_ulLnnqC_mqNvxq`@'));
--result--
LINESTRING(-120.2 38.5,-120.95 40.7,-126.453 43.252)
```

Siehe auch

[ST_AsEncodedPolyline](#)

8.4.20 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.



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

Beispiele

```
--ERzeugt die Zeichenkette einer 3D-Linie aus einem 3D-MultiPoint
SELECT ST_AsEWKT(ST_LineFromMultiPoint(ST_GeomFromEWKT('MULTIPOINT(1 2 3, 4 5 6, 7 8 9)')))) ←
;
--result--
LINESTRING(1 2 3,4 5 6,7 8 9)
```

Siehe auch

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

8.4.21 ST_LineFromText

ST_LineFromText — Erzeugt eine Geometrie aus einer WKT Darstellung mit der angegebenen SRID. Wenn keine SRID angegeben wird, wird diese standardmäßig auf 0 gesetzt.

Synopsis

```
geometry ST_LineFromText(text WKT);
geometry ST_LineFromText(text WKT, integer srid);
```

Beschreibung

Erzeugt eine Geometrie aus einer WKT Darstellung mit der angegebenen SRID. Wenn keine SRID angegeben wird, wird diese standardmäßig auf 0 gesetzt. Wenn das übergebene WKT kein LineString ist, wird NULL zurückgegeben.



Note

OGC SPEC 3.2.6.2 - die Option SRID ist vom Konformitätstest.



Note

Wenn Sie wissen, dass die Geometrie nur aus LINESTRINGs besteht, ist es effizienter einfach ST_GeomFromText zu verwenden. Diese Funktion ruft auch nur ST_GeomFromText auf und fügt die Information hinzu, dass es sich um einen Linienzug handelt.



This method implements the [OpenGIS Simple Features Implementation Specification for SQL 1.1](#). s3.2.6.2



This method implements the SQL/MM specification. SQL-MM 3: 7.2.8

Beispiele

```
SELECT ST_LineFromText('LINESTRING(1 2, 3 4)') AS aline, ST_LineFromText('POINT(1 2)') AS ↵
      null_return;
aline                                     | null_return
-----
01020000000020000000000000000000F ... | t
```

Siehe auch

[ST_GeomFromText](#)

8.4.22 ST_LineFromWKB

ST_LineFromWKB — Erzeugt einen `LINESTRING` mit gegebener `SRID` aus einer `WKB`-Darstellung

Synopsis

```
geometry ST_LineFromWKB(bytea WKB);
geometry ST_LineFromWKB(bytea WKB, integer srid);
```

Beschreibung

Die Funktion `ST_GeogFromWKB` nimmt eine Well-known-Binary Darstellung der Geometrie und eine Id für das Koordinatenreferenzsystem (`SRID`) entgegen und erzeugt eine Instanz des entsprechenden geometrischen Datentyps - in diesem Fall eine Geometrie vom Typ `LineString`. Diese Funktion übernimmt die Rolle der Geometrie-Factory in SQL.

Wenn keine `SRID` angegeben ist, wird diese auf 0 gesetzt. `NULL` wird zurückgegeben, wenn die Eingabe `bytea` keinen `LINESTRING` darstellt.



Note

OGC SPEC 3.2.6.2 - die Option `SRID` ist vom Konformitätstest.



Note

Wenn Sie wissen, dass Ihre Geometrie nur aus `LINESTRINGs` besteht, ist es effizienter einfach `ST_GeomFromWKB` zu verwenden. Diese Funktion ruft auch nur `ST_GeomFromWKB` auf und fügt die Information hinzu, dass es sich um einen Linienzug handelt.



This method implements the [OpenGIS Simple Features Implementation Specification for SQL 1.1](#). s3.2.6.2



This method implements the SQL/MM specification. SQL-MM 3: 7.2.9

Beispiele

```
SELECT ST_LineFromWKB(ST_AsBinary(ST_GeomFromText('LINESTRING(1 2, 3 4)'))) AS aline,
      ST_LineFromWKB(ST_AsBinary(ST_GeomFromText('POINT(1 2)'))) IS NULL AS ↵
      null_return;
aline                                     | null_return
-----
01020000000020000000000000000000F ... | t
```

Siehe auch

[ST_GeomFromWKB](#), [ST_LinestringFromWKB](#)

8.4.23 ST_LinestringFromWKB

`ST_LinestringFromWKB` — Erzeugt eine Geometrie mit gegebener SRID aus einer WKB-Darstellung.

Synopsis

```
geometry ST_LinestringFromWKB(bytea WKB);
geometry ST_LinestringFromWKB(bytea WKB, integer srid);
```

Beschreibung

Die Funktion `ST_LinestringFromWKB` nimmt eine Well-known-Binary Darstellung der Geometrie und eine Id für das Koordinatenreferenzsystem (SRID) entgegen und erzeugt eine Instanz des entsprechenden geometrischen Datentyps - in diesem Fall eine Geometrie vom Typ `LineString`. Diese Funktion übernimmt die Rolle der Geometrie-Factory in SQL.

Wenn keine SRID angegeben ist, wird diese standardmäßig auf 0 gesetzt. NULL wird zurückgegeben, wenn die Eingabe `bytea` keinen `LINSTRING` darstellt. Ist ein Alias für [ST_LineFromWKB](#).

**Note**

OGC SPEC 3.2.6.2 - optionale SRID ist vom Konformitätstest.

**Note**

Wenn Sie wissen, dass Ihre Geometrie nur aus `LINSTRINGS` besteht, ist es effizienter einfach [ST_GeomFromWKB](#) zu verwenden. Diese Funktion ruft auch nur [ST_GeomFromWKB](#) auf und fügt die Information hinzu, dass es sich um einen `LINSTRING` handelt.



This method implements the [OpenGIS Simple Features Implementation Specification for SQL 1.1](#). s3.2.6.2



This method implements the SQL/MM specification. SQL-MM 3: 7.2.9

Beispiele

```
SELECT
  ST_LineStringFromWKB(
    ST_AsBinary(ST_GeomFromText('LINestring(1 2, 3 4)'))
  ) AS aline,
  ST_LinestringFromWKB(
    ST_AsBinary(ST_GeomFromText('POINT(1 2)'))
  ) IS NULL AS null_return;
  aline                                | null_return
-----
01020000000200000000000000000000F ... | t
```

Siehe auch

[ST_GeomFromWKB](#), [ST_LineFromWKB](#)

8.4.24 ST_MakeBox2D

ST_MakeBox2D — Erzeugt eine BOX2D, die durch die gegebene Punktgeometrie definiert ist.

Synopsis

box2d **ST_MakeBox2D**(geometry pointLowLeft, geometry pointUpRight);

Beschreibung

Erzeugt eine BOX2D, die durch die gegebene Punktgeometrie definiert ist. Nützlich bei Bereichsanfragen

Beispiele

```
--Gibt alle Geoobjekte zurück, die sich teilweise oder zur Gänze in den ↔  
  Koordinatenausschnitt des US National Atlas fallen  
--Es wird vorausgesetzt, dass die Geometrie in SRID = 2163 (US National atlas equal area) ↔  
  abgespeichert ist  
SELECT feature_id, feature_name, the_geom  
FROM features  
WHERE the_geom && ST_SetSRID(ST_MakeBox2D(ST_Point(-989502.1875, 528439.5625),  
      ST_Point(-987121.375 , 529933.1875)), 2163)
```

Siehe auch

[ST_MakePoint](#), [ST_Point](#), [ST_SetSRID](#), [ST_SRID](#)

8.4.25 ST_3DMakeBox

ST_3DMakeBox — Erzeugt eine BOX3D, die durch die gegebene 3D-Punktgeometrie definiert ist.

Synopsis

box3d **ST_3DMakeBox**(geometry point3DLowLeftBottom, geometry point3DUpRightTop);

Beschreibung

Erzeugt eine BOX3D, die durch 2 geometrische 3D-Punkte definiert wird.



Dieser Funktion unterstützt 3D und löscht den Z-Index nicht.

Änderung: 2.0.0 In Vorgängerversionen als **ST_MakeBox3D** bezeichnet.

Beispiele

```
SELECT ST_3DMakeBox(ST_MakePoint(-989502.1875, 528439.5625, 10),
                    ST_MakePoint(-987121.375, 529933.1875, 10)) As abb3d

--abb3d--
-----
BOX3D(-989502.1875 528439.5625 10,-987121.375 529933.1875 10)
```

Siehe auch

[ST_MakePoint](#), [ST_SetSRID](#), [ST_SRID](#)

8.4.26 ST_MakeLine

ST_MakeLine — Erzeugt einen Linienzug aus einer Punkt-, Mehrfachpunkt- oder Liniengeometrie.

Synopsis

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

Beschreibung

ST_MakeLine hat 3 Versionen: eine räumliche Aggregatversion die Zeilen von Punkt-, Mehrfachpunkt-, oder Liniengeometrie entgegennimmt und einen Linienzug zurückgibt, eine Funktion die ein Feld mit Punkten, Mehrfachpunkten oder Linien entgegennimmt, und eine normale Funktion welche die Geometrie in Form von zwei Punkten, Mehrfachpunkten oder Linien entgegennimmt. Bevor Sie die Punkte an die Aggregatversion dieser Funktion übergeben, können Sie eine Unterabfrage ausführen um die Punkte zu sortieren.

Eingaben außer Punkt, Mehrfachpunkt oder Linien werden ignoriert.

Wenn Sie Linienstücke hinzufügen, so werden gemeinsame Knoten am Anfang der Linien bei der Ausgabe entfernt. Gemeinsame Knoten von Punkt- und Mehrfachpunkteingaben werden nicht entfernt.



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

Verfügbarkeit: Mit 2.3.0 wurde die Unterstützung zur Eingabe von MultiPoint Elementen eingeführt

Verfügbarkeit: 2.0.0 - Unterstützung zur Eingabe von LineString Elementen eingeführt

Verfügbarkeit: 1.4.0 - **ST_MakeLine**(geomarray) wurde eingeführt. **ST_MakeLine** Aggregatfunktion wurde verbessert, um mehr Punkte schneller handhaben zu können.

Beispiele: Spatiale Aggregatversion

Diese Beispiel nimmt eine Abfolge von GPS Punkten entgegen und erzeugt einen Datensatz für jeden GPS Pfad, wobei das Geometriefeld ein Linienzug ist, welcher in der Reihenfolge der Aufnahmeroute aus den GPS Punkten zusammengesetzt wird.

```
-- Bei Vorgängerversionen von PostgreSQL 9.0 funktioniert dies üblicherweise,
-- allerdings kann es der Anfrageoptimierer gelegentlich vorziehen, die Reihenfolge der
  Unterabfrage zu missachten.
SELECT gps.gps_track, ST_MakeLine(gps.the_geom) As newgeom
      FROM (SELECT gps_track,gps_time, the_geom
              FROM gps_points ORDER BY gps_track, gps_time) As gps
      GROUP BY gps.gps_track;
```

```
-- Wenn Sie PostgreSQL 9.0+ verwenden
-- (Sie können die neue ORDER BY Unterstützung für Aggregate nutzen)
-- ist die richtige Anordnung des Linienzuges garantiert
-- Falls nötig, kann die Reihenfolge auch auf mehreren Attributen basieren
SELECT gps.gps_track, ST_MakeLine(gps.the_geom ORDER BY gps_time) As newgeom
      FROM gps_points As gps
      GROUP BY gps.gps_track;
```

Beispiele: Nicht-Spatiale Aggregatversion

Das erste Beispiel ist einfach eine Linie, die aus 2 Punkten gebildet wird. Das Zweite bildet Linienzüge aus 2 Punkten die von einem Anwender eingegeben werden. Das Dritte ist einmalig, indem es 2 3D-Punkte verbindet, um eine Linie im 3D-Raum zu erzeugen.

```
SELECT ST_AsText(ST_MakeLine(ST_MakePoint(1,2), ST_MakePoint(3,4)));
      st_astext
-----
LINESTRING(1 2,3 4)

SELECT userpoints.id, ST_MakeLine(startpoint, endpoint) As drawn_line
      FROM userpoints ;

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

Beispiele: Verwendung der Feld-Version

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

--Making a 3d line with 3 3-d points
SELECT ST_AsEWKT(ST_MakeLine(ARRAY[ST_MakePoint(1,2,3),
      ST_MakePoint(3,4,5), ST_MakePoint(6,6,6)]));
      st_asewkt
-----
LINESTRING(1 2 3,3 4 5,6 6 6)
```

Siehe auch

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

8.4.27 ST_MakeEnvelope

ST_MakeEnvelope — Erzeugt ein rechteckiges Polygon aus den gegebenen Minimum- und Maximumwerten. Die Eingabewerte müssen in dem Koordinatenreferenzsystem sein, welches durch die SRID angegeben wird.

Synopsis

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

Beschreibung

Erzeugt ein rechteckiges Polygon das durch die Minima und Maxima angegeben wird, durch die gegebene Hülle. Die Eingabewerte müssen in dem Koordinatenreferenzsystem sein, welches durch die SRID angegeben wird. Wenn keine SRID angegeben ist, so wird das Koordinatenreferenzsystem "unknown" angenommen

Verfügbarkeit: 1.5

Erweiterung: 2.0: es wurde die Möglichkeit eingeführt, eine Einhüllende/Envelope festzulegen, ohne dass die SRID spezifiziert ist.

Beispiel: Ein Umgebungsrechteck Polygon erzeugen

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

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

Siehe auch

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

8.4.28 ST_MakePolygon

ST_MakePolygon — Erzeugt ein Polygon, das durch die gegebene Hülle gebildet wird. Die Eingabegeometrie muss aus geschlossenen Linienzügen bestehen.

Synopsis

geometry **ST_MakePolygon**(geometry linestring);

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

Beschreibung

Erzeugt ein Polygon, das durch die gegebene Hülle gebildet wird. Die Eingabegeometrie muss aus geschlossenen Linienzügen bestehen. Hat 2 Varianten.

Variante 1: Nimmt einen geschlossenen Linienzug entgegen.

Variante 2: Erzeugt ein Polygon, das durch die gegebene Hülle und einem Feld mit den Lücken gebildet wird. Sie können ein geometrisches Feld mit **ST_Accum** oder mit den PostgreSQL Konstrukten **ARRAY[]** und **ARRAY()** erzeugen. Die Eingabegeometrie muss aus geschlossenen Linienzügen bestehen.



Note

Diese Funktion akzeptiert keine MULTILINESTRINGS. Verwenden Sie bitte [ST_LineMerge](#) oder [ST_Dump](#) um Linienzüge zu erzeugen.



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

Beispiele: Einzelner geschlossener LINESTRING

```
--2D Linie
SELECT ST_MakePolygon(ST_GeomFromText('LINESTRING(75.15 29.53,77 29,77.6 29.5, 75.15 29.53) ←
'));
--falls der Linienzug nicht geschlossen ist
--können sie ihn schließen, indem Sie den Anfangspunkt hinzufügen
SELECT ST_MakePolygon(ST_AddPoint(foo.open_line, ST_StartPoint(foo.open_line)))
FROM (
SELECT ST_GeomFromText('LINESTRING(75.15 29.53,77 29,77.6 29.5)') As open_line) As foo;

--3D geschlossene Linie --
SELECT ST_MakePolygon(ST_GeomFromText('LINESTRING(75.15 29.53 1,77 29 1,77.6 29.5 1, 75.15 ←
29.53 1)'));

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

-- Meßstrecke --
SELECT ST_MakePolygon(ST_GeomFromText('LINESTRINGM(75.15 29.53 1,77 29 1,77.6 29.5 2, 75.15 ←
29.53 2)'));

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

Beispiele: Außenhülle mit inneren Ringen

Erzeugung eines Donuts mit einem Ameisenloch

```
SELECT ST_MakePolygon(
    ST_ExteriorRing(ST_Buffer(foo.line,10)),
    ARRAY[ST_Translate(foo.line,1,1),
    ST_ExteriorRing(ST_Buffer(ST_MakePoint(20,20),1)) ]
)
FROM
    (SELECT ST_ExteriorRing(ST_Buffer(ST_MakePoint(10,10),10,10))
    As line )
    As foo;
```

Ausgehend von Polygonen/Mehrfachpolygonen der Provinz und Linienzügen für die Gewässer werden die Landesgrenzen erzeugt, wobei Lücken die Seen des Gebietes darstellen. Dies ist ein Beispiel für die Verwendung von ST_Accum in PostGIS.



Note

Das Konstrukt mit CASE wird verwendet, da die Übergabe eines NULL-Feldes an ST_MakePolygon NULL ergibt.



Note

Der Left Join (linker äußerer Verbund) wird verwendet um sicherzustellen dass wir alle Provinzen zurückbekommen, auch wenn diese keine Seen haben.

```
SELECT p.gid, p.province_name,
       CASE WHEN
```



```

        ST_Accum(w.the_geom) IS NULL THEN p.the_geom
    ELSE ST_MakePolygon(ST_LineMerge(ST_Boundary(p.the_geom)), ST_Accum(w. ↵
        the_geom)) END
FROM
    provinces p LEFT JOIN waterlines w
        ON (ST_Within(w.the_geom, p.the_geom) AND ST_IsClosed(w.the_geom))
GROUP BY p.gid, p.province_name, p.the_geom;

--Gleiches Beispiel wie oben, nur mit einer korrespondierenden Unterabfrage
--und der PostgreSQL internen ARRAY() Funktion, welche eine Menge von Zeilen in ein ↵
    Feld/Array umwandelt

SELECT p.gid, p.province_name, CASE WHEN
    EXISTS(SELECT w.the_geom
        FROM waterlines w
        WHERE ST_Within(w.the_geom, p.the_geom)
        AND ST_IsClosed(w.the_geom))
    THEN
        ST_MakePolygon(ST_LineMerge(ST_Boundary(p.the_geom)),
            ARRAY(SELECT w.the_geom
                FROM waterlines w
                WHERE ST_Within(w.the_geom, p.the_geom)
                AND ST_IsClosed(w.the_geom)))
    ELSE p.the_geom END As the_geom
FROM
    provinces p;
```

Siehe auch

[ST_Boundary](#), [ST_Accum](#), [ST_AddPoint](#), [ST_GeometryType](#), [ST_IsClosed](#), [ST_LineMerge](#), [ST_BuildArea](#)

8.4.29 ST_MakePoint

ST_MakePoint — Erzeugt eine 2D-, 3DZ- oder 4D-Punktgeometrie.

Synopsis

geometry **ST_MakePoint**(double precision x, double precision y);

geometry **ST_MakePoint**(double precision x, double precision y, double precision z);

geometry **ST_MakePoint**(double precision x, double precision y, double precision z, double precision m);

Beschreibung

Erzeugt eine 2D-, 3DZ- oder 4D-Punktgeometrie (Geometrie mit Kilometrierung). **ST_MakePoint** ist zwar nicht OGC-konform, ist aber im Allgemeinen schneller und genauer als [ST_GeomFromText](#) oder [ST_PointFromText](#) und auch leichter anzuwenden wenn Sie mit rohen Koordinaten anstatt mit WKT arbeiten.



Note

Merke: X steht für die geographische Länge, Y für die geographische Breite.

**Note**

Für Punkt mit X-, Y- und M-Koordinaten verwenden Sie bitte **ST_MakePointM**.



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

Beispiele

```
--Gibt einen Punkt mit unbekannter SRID aus
SELECT ST_MakePoint(-71.1043443253471, 42.3150676015829);

--Gibt einen Punkt in geographischer Länge und Breite im WGS84 aus
SELECT ST_SetSRID(ST_MakePoint(-71.1043443253471, 42.3150676015829),4326);

--Gibt einen 3D-Punkt zurück (z.B.: wenn der Punkt eine Höhe aufweist)
SELECT ST_MakePoint(1, 2,1.5);

--Gibt die Z-Koordinate des Punktes zurück
SELECT ST_Z(ST_MakePoint(1, 2,1.5));
result
-----
1.5
```

Siehe auch

ST_GeomFromText, **ST_PointFromText**, **ST_SetSRID**, **ST_MakePointM**

8.4.30 ST_MakePointM

ST_MakePointM — Erzeugt eine Punktgeometrie mit einer X-, einer Y- und einer M-Koordinate.

Synopsis

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

Beschreibung

Erzeugt einen Punkt mit x, y und measure/Kilometrierungs Koordinaten.

**Note**

Merke: X ist die geographische Länge, Y ist die Breite.

Beispiele

In diesen Beispielen verwenden wir **ST_AsEWKT** um die Textdarstellung zu zeigen und nicht **ST_AsText**, da **ST_AsText** die Ausgabe von M nicht unterstützt.

```
--Gibt die EWKT Darstellung des Punktes mit SRID "unknown" zurück
SELECT ST_AsEWKT(ST_MakePointM(-71.1043443253471, 42.3150676015829, 10));

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

--Return EWKT representation of point with measure marked as WGS 84 long lat
--Ausgabe der EWKT Darstellung des Punktes mit "measure" und gekennzeichnet als WGS 84 ↔
  Länge/Breite
SELECT ST_AsEWKT(ST_SetSRID(ST_MakePointM(-71.1043443253471, 42.3150676015829,10),4326));

                                     st_asewkt
-----
SRID=4326;POINTM(-71.1043443253471 42.3150676015829 10)

--Gibt einen 3D-Punkt zurück (z.B. Seehöhe)
SELECT ST_MakePoint(1, 2,1.5);

--Gibt m des Punktes
SELECT ST_M(ST_MakePointM(-71.1043443253471, 42.3150676015829,10));
result
-----
10
```

Siehe auch

[ST_AsEWKT](#), [ST_MakePoint](#), [ST_SetSRID](#)

8.4.31 ST_MLineFromText

ST_MLineFromText — Liest einen festgelegten ST_MultiLineString Wert von einer WKT-Darstellung aus.

Synopsis

```
geometry ST_MLineFromText(text WKT, integer srid);
geometry ST_MLineFromText(text WKT);
```

Beschreibung

Erzeugt eine Geometrie aus einer Well-known-Text (WKT) Darstellung mit der angegebenen SRID. Wenn keine SRID angegeben wird, wird diese standardmäßig auf 0 gesetzt.

OGC SPEC 3.2.6.2 - Die Option SRID stammt aus dem Konformitätstest

Gibt NULL zurück wenn der WKT kein MULTILINESTRING ist.

**Note**

Verwenden Sie diese Funktion nicht, wenn Sie sich vollkommen sicher sind, dass ihre WKT Geometrie nur aus Punkten besteht. Sie ist langsamer als ST_GeomFromText, da sie einen zusätzlichen Validierungsschritt hinzufügt.



This method implements the [OpenGIS Simple Features Implementation Specification for SQL 1.1](#). s3.2.6.2



This method implements the SQL/MM specification. SQL-MM 3: 9.4.4

Beispiele

```
SELECT ST_MLineFromText('MULTILINESTRING((1 2, 3 4), (4 5, 6 7))');
```

Siehe auch

[ST_GeomFromText](#)

8.4.32 ST_MPointFromText

ST_MPointFromText — Erzeugt eine Geometrie aus WKT mit der angegebenen SRID. Wenn keine SRID angegeben wird, wird diese standardmäßig auf 0 gesetzt.

Synopsis

geometry **ST_MPointFromText**(text WKT, integer srid);

geometry **ST_MPointFromText**(text WKT);

Beschreibung

Erzeugt eine Geometrie aus WKT mit der angegebenen SRID. Wenn SRID nicht angegeben ist, wird sie standardmäßig auf 0 gesetzt.

OGC SPEC 3.2.6.2 - Die Option SRID stammt aus dem Konformitätstest

Gibt NULL zurück, wenn der WKT kein MULTIPOINT ist.



Note

Verwenden Sie diese Funktion nicht, wenn Sie sich vollkommen sicher sind, dass ihre WKT Geometrie nur aus Punkten besteht. Sie ist langsamer als [ST_GeomFromText](#), da sie einen zusätzlichen Validierungsschritt hinzufügt.



This method implements the [OpenGIS Simple Features Implementation Specification for SQL 1.1](#). 3.2.6.2



This method implements the SQL/MM specification. SQL-MM 3: 9.2.4

Beispiele

```
SELECT ST_MPointFromText('MULTIPOINT(1 2, 3 4)');  
SELECT ST_MPointFromText('MULTIPOINT(-70.9590 42.1180, -70.9611 42.1223)', 4326);
```

Siehe auch

[ST_GeomFromText](#)

8.4.33 ST_MPolyFromText

ST_MPolyFromText — Erzeugt eine MultiPolygon Geometrie aus WKT mit der angegebenen SRID. Wenn SRID nicht angegeben ist, wird sie standardmäßig auf 0 gesetzt.

Synopsis

geometry **ST_MPolyFromText**(text WKT, integer srid);
 geometry **ST_MPolyFromText**(text WKT);

Beschreibung

Erzeugt ein MultiPolygon von WKT mit der gegebenen SRID. Wenn SRID nicht angegeben ist, wird sie standardmäßig auf 0 gesetzt.

OGC SPEC 3.2.6.2 - Die Option SRID stammt aus dem Konformitätstest

Meldet einen Fehler, wenn der WKT kein MULTIPOLYGON ist.



Note

Verwenden Sie diese Funktion nicht, wenn Sie sich vollkommen sicher sind, dass ihre WKT Geometrie nur aus Multi-Polygonen besteht. Sie ist langsamer als ST_GeomFromText, da sie einen zusätzlichen Validierungsschritt hinzufügt.



This method implements the [OpenGIS Simple Features Implementation Specification for SQL 1.1](#). s3.2.6.2



This method implements the SQL/MM specification. SQL-MM 3: 9.6.4

Beispiele

```
SELECT ST_MPolyFromText('MULTIPOLYGON(((0 0 1,20 0 1,20 20 1,0 20 1,0 0 1),(5 5 3,5 7 3,7 7 3,7 5 3,5 5 3)))');
SELECT ST_MPolyFromText('MULTIPOLYGON((-70.916 42.1002,-70.9468 42.0946,-70.9765 42.0872,-70.9754 42.0875,-70.9749 42.0879,-70.9752 42.0881,-70.9754 42.0891,-70.9758 42.0894,-70.9759 42.0897,-70.9759 42.0899,-70.9754 42.0902,-70.9756 42.0906,-70.9753 42.0907,-70.9753 42.0917,-70.9757 42.0924,-70.9755 42.0928,-70.9755 42.0942,-70.9751 42.0948,-70.9755 42.0953,-70.9751 42.0958,-70.9751 42.0962,-70.9759 42.0983,-70.9767 42.0987,-70.9768 42.0991,-70.9771 42.0997,-70.9771 42.1003,-70.9768 42.1005,-70.977 42.1011,-70.9766 42.1019,-70.9768 42.1026,-70.9769 42.1033,-70.9775 42.1042,-70.9773 42.1043,-70.9776 42.1043,-70.9778 42.1048,-70.9773 42.1058,-70.9774 42.1061,-70.9779 42.1065,-70.9782 42.1078,-70.9788 42.1085,-70.9798 42.1087,-70.9806 42.109,-70.9807 42.1093,-70.9806 42.1099,-70.9809 42.1109,-70.9808 42.1112,-70.9798 42.1116,-70.9792 42.1127,-70.979 42.1129,-70.9787 42.1134,-70.979 42.1139,-70.9791 42.1141,-70.9987 42.1116,-71.0022 42.1273,-70.9408 42.1513,-70.9315 42.1165,-70.916 42.1002)))',4326);
```

Siehe auch

[ST_GeomFromText](#), [ST_SRID](#)

8.4.34 ST_Point

ST_Point — Gibt einen ST_Point mit den gegebenen Koordinatenwerten aus. Ein OGC-Alias für ST_MakePoint.

Synopsis

geometry **ST_Point**(float x_lon, float y_lat);

Beschreibung

Gibt einen `ST_Point` mit den gegebenen Koordinatenwerten aus. Ein SQL/MM kompatibler Alias für `ST_MakePoint`, der nur ein X und ein Y entgegennimmt.



This method implements the SQL/MM specification. SQL-MM 3: 6.1.2

Beispiele: Geometrie

```
SELECT ST_SetSRID(ST_Point(-71.1043443253471, 42.3150676015829), 4326)
```

Beispiele: Geographie

```
SELECT CAST(ST_SetSRID(ST_Point(-71.1043443253471, 42.3150676015829), 4326) As geography);
```

```
-- Der :: ist die Kurzform von PostgreSQL für die Typumwandlung.
```

```
SELECT ST_SetSRID(ST_Point(-71.1043443253471, 42.3150676015829), 4326)::geography;
```

```
--Wenn Ihre Punktkoordinaten in einem anderen Koordinatenreferenzsystem als WGS-84 Länge/ ←  
    Breite vorliegen, müssen Sie vor der Typumwandlung eine Koordinatentransformation ←  
    durchführen.
```

```
-- In diesem Beispiel transformieren wir einen Punkt von "Pennsylvania State Plane Feet" ←  
    nach WGS 84 and wandeln ihn anschließend in einen geographischen Datentyp um
```

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

Siehe auch

Section [4.2.1](#), [ST_MakePoint](#), [ST_SetSRID](#), [ST_Transform](#)

8.4.35 ST_PointFromGeoHash

`ST_PointFromGeoHash` — Gibt einen Punkt von einer GeoHash Zeichenfolge zurück.

Synopsis

point **ST_PointFromGeoHash**(text geohash, integer precision=full_precision_of_geohash);

Beschreibung

Gibt die Geometrie einer GeoHash Zeichenfolge zurück. Der Punkt entspricht dem Mittelpunkt des GeoHas.

Wenn keine `precision` angegeben wird, dann gibt `ST_PointFromGeoHash` einen Punkt zurück, der auf der vollständigen Genauigkeit der gegebenen GeoHash Zeichenfolge beruht.

Wenn `precision` angegeben wird, verwendet `ST_PointFromGeoHash` entsprechend viele Zeichen des GeoHash, um den Punkt zu erzeugen.

Verfügbarkeit: 2.1.0

Beispiele

```
SELECT ST_AsText(ST_PointFromGeoHash('9qqj7nmxncgyy4d0dbxqz0'));
          st_astext
-----
POINT(-115.172816 36.114646)

SELECT ST_AsText(ST_PointFromGeoHash('9qqj7nmxncgyy4d0dbxqz0', 4));
          st_astext
-----
POINT(-115.13671875 36.123046875)

SELECT ST_AsText(ST_PointFromGeoHash('9qqj7nmxncgyy4d0dbxqz0', 10));
          st_astext
-----
POINT(-115.172815918922 36.1146435141563)
```

Siehe auch

[ST_GeoHash](#), [ST_Box2dFromGeoHash](#), [ST_GeomFromGeoHash](#)

8.4.36 ST_PointFromText

ST_PointFromText — Erzeugt eine Punktgeometrie mit gegebener SRID von WKT. Wenn SRID nicht angegeben ist, wird sie standardmäßig auf 0 gesetzt.

Synopsis

geometry **ST_PointFromText**(text WKT);
geometry **ST_PointFromText**(text WKT, integer srid);

Beschreibung

Erzeugt ein PostGIS ST_Geometrie Punktobjekt von der OGC Well-known-Text Darstellung. Wenn die SRID nicht angegeben ist, wird sie standardmäßig auf "unknown" (zurzeit 0) gesetzt. Falls die Geometrie nicht in der WKT Punktdarstellung vorliegt, wird NULL zurückgegeben. Bei einer invaliden WKT Darstellung wird eine Fehlermeldung angezeigt.

Note



Die Funktion **ST_PointFromText** hat zwei Varianten. Die erste Variante nimmt keine SRID entgegen und gibt eine Geometrie ohne ein bestimmtes Koordinatenreferenzsystem aus. Die zweite Variante nimmt eine SRID als zweiten Übergabewert entgegen und gibt eine Geometrie zurück, die diese SRID als Teil ihrer Metadaten beinhaltet. Die SRID muss in der Tabelle "spatial_ref_sys" definiert sein.

Note



Verwenden Sie diese Funktion nicht, wenn Sie sich vollkommen sicher sind, dass ihre WKT Geometrie nur aus Punkten besteht. Sie ist langsamer als **ST_GeomFromText**, da sie einen zusätzlichen Validierungsschritt hinzufügt. Wenn Sie Punkte aus Koordinaten in Länge und Breite erstellen und mehr auf Rechenleistung und Genauigkeit wertlegen als auf OGC-Konformität, so verwenden Sie bitte **ST_MakePoint** oder den OGC-konformen Alias **ST_Point**.



This method implements the [OpenGIS Simple Features Implementation Specification for SQL 1.1](#). s3.2.6.2 - die Option SRID ist vom Konformitätstest.



This method implements the SQL/MM specification. SQL-MM 3: 6.1.8

Beispiele

```
SELECT ST_PointFromText('POINT(-71.064544 42.28787)');
SELECT ST_PointFromText('POINT(-71.064544 42.28787)', 4326);
```

Siehe auch

[ST_GeomFromText](#), [ST_MakePoint](#), [ST_Point](#), [ST_SRID](#)

8.4.37 ST_PointFromWKB

ST_PointFromWKB — Erzeugt eine Geometrie mit gegebener SRID von WKB.

Synopsis

```
geometry ST_GeomFromWKB(bytea geom);
geometry ST_GeomFromWKB(bytea geom, integer srid);
```

Beschreibung

Die Funktion ST_PointFromWKB nimmt eine Well-known-Binary Darstellung der Geometrie und eine Id für das Koordinatenreferenzsystem (SRID) entgegen und erzeugt eine Instanz des entsprechenden geometrischen Datentyps - in diesem Fall eine Geometrie vom Typ POINT. Diese Funktion übernimmt die Rolle der Geometrie-Factory in SQL.

Wenn keine SRID angegeben ist, wird diese standardmäßig auf 0 gesetzt. NULL wird zurückgegeben, wenn die Eingabe bytea keinen POINT darstellt.



This method implements the [OpenGIS Simple Features Implementation Specification for SQL 1.1](#). s3.2.7.2



This method implements the SQL/MM specification. SQL-MM 3: 6.1.9



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



This method supports Circular Strings and Curves

Beispiele

```
SELECT
  ST_AsText (
    ST_PointFromWKB (
      ST_AsEWKB('POINT(2 5)::geometry)
    )
  );
st_astext
-----
```



```

POINT(2 5)
(1 row)

SELECT
  ST_AsText(
    ST_PointFromWKB(
      ST_AseWKB('LINESTRING(2 5, 2 6)::geometry')
    )
  );
 st_astext
-----
(1 row)

```

Siehe auch

[ST_GeomFromWKB](#), [ST_LineFromWKB](#)

8.4.38 ST_Polygon

ST_Polygon — Gibt ein Polygon zurück, das aus vorgegebenen Linienzug und SRID erzeugt wurde.

Synopsis

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

Beschreibung

Gibt ein Polygon zurück, das aus vorgegebenen Linienzug und SRID erzeugt wurde.



Note

ST_Polygon ist ähnlich der ersten Version von ST_MakePolygon, außer dass auch das Koordinatenreferenzsystem (SRID) des Polygons festgelegt wird. Da die Funktion nicht mit MULTILINESTRINGS arbeitet, verwenden Sie bitte ST_LineMerge um Mehrfachlinien zu vereinigen. Kann auch keine Polygone mit Lücken erzeugen; benutzen Sie ST_MakePolygon für diesen Fall.



This method implements the [OpenGIS Simple Features Implementation Specification for SQL 1.1](#).



This method implements the SQL/MM specification. SQL-MM 3: 8.3.2



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

Beispiele

```

-- ein 2D Polygon
SELECT ST_Polygon(ST_GeomFromText('LINESTRING(75.15 29.53,77 29,77.6 29.5, 75.15 29.53)'), ←
  4326);

--result--
POLYGON((75.15 29.53,77 29,77.6 29.5,75.15 29.53))
--a 3d polygon

```

```
SELECT ST_AsEWKT(ST_Polygon(ST_GeomFromEWKT('LINESTRING(75.15 29.53 1,77 29 1,77.6 29.5 1, ←
75.15 29.53 1)')), 4326));

result
-----
SRID=4326;POLYGON((75.15 29.53 1,77 29 1,77.6 29.5 1,75.15 29.53 1))
```

Siehe auch

[ST_AsEWKT](#), [ST_AsText](#), [ST_GeomFromEWKT](#), [ST_GeomFromText](#), [ST_LineMerge](#), [ST_MakePolygon](#)

8.4.39 ST_PolygonFromText

ST_PolygonFromText — Erzeugt eine Geometrie aus WKT mit der angegebenen SRID. Wenn keine SRID angegeben wird, wird diese standardmäßig auf 0 gesetzt.

Synopsis

```
geometry ST_PolygonFromText(text WKT);
geometry ST_PolygonFromText(text WKT, integer srid);
```

Beschreibung

Erzeugt eine Geometrie mit gegebener SRID von WKT. Wenn SRID nicht angegeben ist, wird sie standardmäßig auf 0 gesetzt. Gibt NULL zurück, wenn WKT kein Polygon ist.

OGC SPEC 3.2.6.2 - Die Option SRID stammt aus dem Konformitätstest

**Note**

Verwenden Sie diese Funktion nicht, wenn Sie sich vollkommen sicher sind, dass ihre WKT Geometrie nur aus Polygonen besteht. Sie ist langsamer als [ST_GeomFromText](#), da sie einen zusätzlichen Validierungsschritt hinzufügt.



This method implements the [OpenGIS Simple Features Implementation Specification for SQL 1.1](#). s3.2.6.2



This method implements the SQL/MM specification. SQL-MM 3: 8.3.6

Beispiele

```
SELECT ST_PolygonFromText('POLYGON((-71.1776585052917 42.3902909739571,-71.1776820268866 ←
42.3903701743239,
-71.1776063012595 42.3903825660754,-71.1775826583081 42.3903033653531,-71.1776585052917 ←
42.3902909739571))');
st_polygonfromtext
-----
0103000000010000000500000006...
```

```
SELECT ST_PolygonFromText('POINT(1 2)') IS NULL as point_is_notpoly;

point_is_notpoly
-----
t
```

Siehe auch

[ST_GeomFromText](#)

8.4.40 ST_WKBToSQL

ST_WKBToSQL — Gibt einen geometrischen Datentyp (**ST_Geometry**) aus einer Well-known-Binary (WKB) Darstellung zurück. Ein Synonym für **ST_GeomFromWKB**, welches jedoch keine SRID annimmt

Synopsis

```
geometry ST_WKBToSQL(bytea WKB);
```

Beschreibung



This method implements the SQL/MM specification. SQL-MM 3: 5.1.36

Siehe auch

[ST_GeomFromWKB](#)

8.4.41 ST_WKTToSQL

ST_WKTToSQL — Gibt einen spezifizierten **ST_Geometry**-Wert von einer Well-known-Text Darstellung (WKT) zurück. Die Bezeichnung ist ein Alias für **ST_GeomFromText**

Synopsis

```
geometry ST_WKTToSQL(text WKT);
```

Beschreibung



This method implements the SQL/MM specification. SQL-MM 3: 5.1.34

Siehe auch

[ST_GeomFromText](#)

8.5 Zugriffsfunktionen auf Geometrien

8.5.1 GeometryType

GeometryType — Gibt den Geometrietyp als Zeichenkette zurück. z.B.: 'LINESTRING', 'POLYGON', 'MULTIPOINT', etc.

Synopsis

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

Beschreibung

Gibt den Geometriertyp als Zeichenkette zurück. z.B.: 'LINESTRING', 'POLYGON', 'MULTIPOINT', etc.

OGC SPEC s2.1.1.1 - Gibt den Namen des instanziierten Subtyps der Geometrie zurück, von dem die geometrische Instanz ein Mitglied ist. Der Name des instanziierten Subtyps der Geometrie wird als Zeichenkette ausgegeben.



Note

Die Funktion zeigt auch an ob die Geometrie eine Maßzahl aufweist, indem eine Zeichenkette wie 'POINTM' zurückgegeben wird.

Erweiterung: Mit 2.0.0 wurde die Unterstützung für polyedrische Oberflächen, Dreiecke und TIN eingeführt.



This method implements the [OpenGIS Simple Features Implementation Specification for SQL 1.1](#).



This method supports Circular Strings and Curves



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



This function supports Polyhedral surfaces.



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

Beispiele

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

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

```
SELECT GeometryType(geom) as result
FROM
  (SELECT
    ST_GeomFromEWKT('TIN (((
      0 0 0,
      0 0 1,
      0 1 0,
      0 0 0
    )), ((
      0 0 0,
      0 1 0,
      1 1 0,
      0 0 0
```

```

        ))
    ) ' ' AS geom
) AS g;
result
-----
TIN

```

Siehe auch

[ST_GeometryType](#)

8.5.2 ST_Boundary

ST_Boundary — Gibt die abgeschlossene Hülle aus der kombinierten Begrenzung der Geometrie zurück.

Synopsis

geometry **ST_Boundary**(geometry geomA);

Beschreibung

Gibt die abgeschlossene Hülle aus der kombinierten Begrenzung der Geometrie zurück. Die Definition der kombinierte Begrenzung ist in Abschnitt 3.12.3.2 der OGC SPEC beschrieben. Da das Ergebnis dieser Funktion eine abgeschlossene Hülle und daher topologisch geschlossen ist, kann die resultierende Begrenzung durch geometrische Primitive, wie in Abschnitt 3.12.2. der OGC SPEC erörtert, dargestellt werden.

Wird durch das GEOS Modul ausgeführt



Note

Vor 2.0.0 meldete diese Funktion einen Fehler, falls sie auf eine `GEOMETRYCOLLECTION` angewandt wurde. Ab 2.0.0 wird stattdessen `NULL` (nicht unterstützte Eingabe) zurückgegeben.



This method implements the [OpenGIS Simple Features Implementation Specification for SQL 1.1](#). OGC SPEC s2.1.1.1



This method implements the SQL/MM specification. SQL-MM 3: 5.1.14



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

Erweiterung: mit 2.1.0 wurde die Unterstützung von Dreiecken eingeführt

Beispiele



Linienzug mit überlagerten Begrenzungspunkten

```
SELECT ST_Boundary(geom)
FROM (SELECT 'LINESTRING(100 150,50 60, ↵
      70 80, 160 170)>:::geometry As geom) As f;

-- Ausgabe als ST_AsText
MULTIPOINT(100 150,160 170)
```



Polygon mit Lücke und der Abgrenzung/Boundary als Multilinestring

```
SELECT ST_Boundary(geom)
FROM (SELECT
'POLYGON (( 10 130, 50 190, 110 190, 140 ↵
      150, 150 80, 100 10, 20 40, 10 130 ),
      ( 70 40, 100 50, 120 80, 80 110, ↵
      50 90, 70 40 ))>:::geometry As geom) As f;

-- Ausgabe als ST_AsText
MULTILINESTRING((10 130,50 190,110 ↵
      190,140 150,150 80,100 10,20 40,10 130),
      (70 40,100 50,120 80,80 110,50 ↵
      90,70 40))
```

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

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

--Verwendung eines 3D Polygons
SELECT ST_AsEWKT(ST_Boundary(ST_GeomFromEWKT('POLYGON((1 1 1,0 0 1, -1 1 1, 1 1 1))')));

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

--Verwendung eines 3D Multilinestrings
SELECT ST_AsEWKT(ST_Boundary(ST_GeomFromEWKT('MULTILINESTRING((1 1 1,0 0 0.5, -1 1 1),(1 1 ↵
      0.5,0 0 0.5, -1 1 0.5, 1 1 0.5) )')));

st_asewkt
-----
```

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

Siehe auch

[ST_AsText](#), [ST_ExteriorRing](#), [ST_MakePolygon](#)

8.5.3 ST_CoordDim

ST_CoordDim — Gibt die Dimension der Koordinaten von ST_Geometry zurück.

Synopsis

```
integer ST_CoordDim(geometry geomA);
```

Beschreibung

Gibt die Dimension der Koordinaten für den Wert von ST_Geometry zurück.

Dies ist der MM konforme Alias für [ST_NDims](#)



This method implements the [OpenGIS Simple Features Implementation Specification for SQL 1.1](#).



This method implements the SQL/MM specification. SQL-MM 3: 5.1.3



This method supports Circular Strings and Curves



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



This function supports Polyhedral surfaces.



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

Beispiele

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

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

Siehe auch

[ST_NDims](#)

8.5.4 ST_Dimension

ST_Dimension — Die inhärente Dimension des geometrischen Objekts muss niedriger oder gleich der Dimension der Koordinaten sein.

Synopsis

integer **ST_Dimension**(geometry g);

Beschreibung

Die inhärente geometrische Dimension eines Objektes muss kleiner gleich der Dimension der Koordinaten sein. Nach OGC SPEC s2.1.1.1 - gibt 0 für POINT, 1 für LINESTRING, 2 for POLYGON, und die größte Dimension der Teile einer GEOMETRY COLLECTION. Wenn die Dimension nicht bekannt ist (empty geometry), so wird NULL zurückgegeben.



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

Erweiterung: 2.0.0 - Unterstützung für polyedrische Oberflächen und TIN eingeführt.



Note

Vor 2.0.0 meldete diese Funktion einen Fehler, falls sie auf eine leere Geometrie angewandt wurde.



This function supports Polyhedral surfaces.



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

Beispiele

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

Siehe auch

[ST_NDims](#)

8.5.5 ST_EndPoint

ST_EndPoint — Gibt den Endpunkt einer LINESTRING oder CIRCULARLINESTRING Geometrie als POINT zurück.

Synopsis

boolean **ST_EndPoint**(geometry g);

Beschreibung

Gibt den Endpunkt einer `LINESTRING` Geometrie als `POINT` oder `NULL` zurück, falls der Eingabewert nicht ein `LINESTRING` ist.



This method implements the SQL/MM specification. SQL-MM 3: 7.1.4



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



This method supports Circular Strings and Curves

Note



Änderung: 2.0.0 unterstützt die Verarbeitung von `MultiLineString`'s die nur aus einer einzelnen Geometrie bestehen, nicht mehr. In früheren Versionen von PostGIS gab die Funktion bei einem aus einer einzelnen Linie bestehender `MultiLineString` den Anfangspunkt zurück. Ab 2.0.0 gibt sie nur `NULL` zurück, so wie bei jedem anderen `MultiLineString`. Die alte Verhaltensweise war undokumentiert, aber Anwender, die annahmen, dass Sie Ihre Daten als `LINESTRING` vorliegen haben, könnten in 2.0 dieses zurückgegebene `NULL` bemerken.

Beispiele

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

postgis=# SELECT ST_EndPoint('POINT(1 1)::geometry') IS NULL AS is_null;
is_null
-----
t
(1 row)

--3D Endpunkt
SELECT ST_AsEWKT(ST_EndPoint('LINESTRING(1 1 2, 1 2 3, 0 0 5)'));
st_asewkt
-----
POINT(0 0 5)
(1 row)
```

Siehe auch

[ST_PointN](#), [ST_StartPoint](#)

8.5.6 ST_Envelope

`ST_Envelope` — Gibt eine Geometrie in doppelter Genauigkeit (`float8`) zurück, welche das Umgebungsrechteck der beigestellten Geometrie darstellt.

Synopsis

geometry **ST_Envelope**(geometry g1);

Beschreibung

Gibt das kleinstmögliche Umgebungsrechteck der bereitgestellten Geometrie als Geometrie im Float8-Format zurück. Das Polygon wird durch die Eckpunkte des Umgebungsrechteckes beschrieben ((MINX, MINY), (MINX, MAXY), (MAXX, MAXY), (MAXX, MINY), (MINX, MINY)). (PostGIS fügt auch die ZMIN/ZMAX Koordinaten hinzu).

Spezialfälle (vertikale Linien, Punkte) geben eine Geometrie geringerer Dimension zurück als POLYGON, insbesondere POINT oder LINESTRING.

Verfügbarkeit: 1.5.0 Änderung der Verhaltensweise insofern, das die Ausgabe in Double Precision anstelle von Float4 erfolgt



This method implements the [OpenGIS Simple Features Implementation Specification for SQL 1.1](#). s2.1.1.1



This method implements the SQL/MM specification. SQL-MM 3: 5.1.15

Beispiele

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

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

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

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

Siehe auch

[Box2D](#), [Box3D](#)

8.5.7 ST_BoundingDiagonal

ST_BoundingDiagonal — Gibt die Diagonale des Umgebungsrechtecks 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.

Die zurückgegebene Linienzug-Geometrie beinhaltet immer die SRID und die Dimensionalität (Anwesenheit von Z und M) der eingegebenen Geometrie.

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.



Note

Bei Spezialfällen (ein einzelner Knoten als Eingabewert) ist der zurückgegebene Linienzug topologisch ungültig (kein Inneres/Interior). Das Ergebnis ist dadurch jedoch nicht semantisch ungültig.

Verfügbarkeit: 2.2.0



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



This function supports M coordinates.

Beispiele

```
-- Gibt die kleinste X-Koordinate eines Buffers um einen Punkt aus
SELECT ST_X(ST_StartPoint(ST_BoundingDiagonal(
  ST_Buffer(ST_MakePoint(0,0),10)
)));
 st_x
-----
  -10
```

Siehe auch

[ST_StartPoint](#), [ST_EndPoint](#), [ST_X](#), [ST_Y](#), [ST_Z](#), [ST_M](#), &&&

8.5.8 ST_ExteriorRing

ST_ExteriorRing — 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. Funktioniert nicht mit `MULTIPOLYGON`

Synopsis

```
geometry ST_ExteriorRing(geometry a_polygon);
```

Beschreibung

Gibt einen Linienzug zurück, welcher den äußeren Ring der POLYGON Geometrie darstellt. Gibt NULL zurück wenn es sich bei der Geometrie um kein Polygon handelt.



Note

Funktioniert nur mit dem Geometrietyp POLYGON



This method implements the [OpenGIS Simple Features Implementation Specification for SQL 1.1](#). 2.1.5.1



This method implements the SQL/MM specification. SQL-MM 3: 8.2.3, 8.3.3



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

Beispiele

```
--Wenn Sie eine Tabelle mit Polygonen haben
SELECT gid, ST_ExteriorRing(the_geom) AS ering
FROM sometable;

--Wenn Sie eine Tabelle mit MULTIPOLYGONen haben
--und Sie wollen als Ergebnis einen MULTILINESTRING der aus Außenringen der Polygone ↔
    zusammengesetzt ist
SELECT gid, ST_Collect(ST_ExteriorRing(the_geom)) AS erings
      FROM (SELECT gid, (ST_Dump(the_geom)).geom As the_geom
            FROM sometable) As foo
GROUP BY gid;

--3D Beispiel
SELECT ST_AsEWKT(
  ST_ExteriorRing(
    ST_GeomFromEWKT('POLYGON((0 0 1, 1 1 1, 1 2 1, 1 1 1, 0 0 1))')
  )
);

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

Siehe auch

[ST_InteriorRingN](#), [ST_Boundary](#), [ST_NumInteriorRings](#)

8.5.9 ST_GeometryN

ST_GeometryN — Gibt die auf 1-basierende n-te Geometrie zurück, wenn es sich bei der Geometrie um eine GEOMETRYCOLLECTION, (MULTI)POINT, (MULTI)LINestring, MULTICURVE oder (MULTI)POLYGON, POLYHEDRALSURFACE handelt. Anderenfalls wird NULL zurückgegeben.

Synopsis

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

Beschreibung

Gibt die auf 1-basierende n-te Geometrie zurück, wenn es sich bei der Geometrie um eine GEOMETRYCOLLECTION, (MULTI)POINT, (MULTI)LINestring, MULTICURVE oder (MULTI)POLYGON, POLYHEDRALSURFACE handelt. Anderenfalls wird NULL zurückgegeben.



Note

Seit Version 0.8.0 basiert der Index auf 1, so wie in der OGC Spezifikation. Vorhergegangene Versionen waren 0-basiert.



Note

Falls Sie alle Geometrien einer Geometrie entnehmen wollen, so ist ST_Dump wesentlich leistungsfähiger und es funktioniert auch mit Einzelgeometrien.

Erweiterung: Mit 2.0.0 wurde die Unterstützung für polyedrische Oberflächen, Dreiecke und TIN eingeführt.

Änderung: 2.0.0 Vorangegangene Versionen geben bei Einzelgeometrien NULL zurück. Dies wurde geändert um die Geometrie für den ST_GeometryN(..,1) Fall zurückzugeben.



This method implements the [OpenGIS Simple Features Implementation Specification for SQL 1.1](#).



This method implements the SQL/MM specification. SQL-MM 3: 9.1.5



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



This method supports Circular Strings and Curves



This function supports Polyhedral surfaces.



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

Standard Beispiele

```
--Entnahme einer Teilmenge von Punkten aus einem 3D Multipoint
SELECT n, ST_AsEWKT(ST_GeometryN(the_geom, n)) As geomewkt
FROM (
VALUES (ST_GeomFromEWKT('MULTIPOINT(1 2 7, 3 4 7, 5 6 7, 8 9 10)') ),
( ST_GeomFromEWKT('MULTICURVE(CIRCULARSTRING(2.5 2.5,4.5 2.5, 3.5 3.5), (10 11, 12 11))') )
)As foo(the_geom)
CROSS JOIN generate_series(1,100) n
WHERE n <= ST_NumGeometries(the_geom);
```

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

```
--Entnahme aller Geometrien (sinnvoll, wenn Sie einen Schlüssel/ID zuweisen wollen)
SELECT gid, n, ST_GeometryN(the_geom, n)
FROM sometable CROSS JOIN generate_series(1,100) n
WHERE n <= ST_NumGeometries(the_geom);
```

Beispiele für polyedrische Oberflächen, TIN und Dreieck

```
-- Beispiel für eine polyedrische Oberfläche
-- Auftrennung einer polyedrischen Oberfläche in Teilflächen/Faces
SELECT ST_AsEWKT(ST_GeometryN(p_geom,3)) AS geom_ewkt
FROM (SELECT ST_GeomFromEWKT('POLYHEDRALSURFACE(
((0 0 0, 0 0 1, 0 1 1, 0 1 0, 0 0 0)),
((0 0 0, 0 1 0, 1 1 0, 1 0 0, 0 0 0)),
((0 0 0, 1 0 0, 1 0 1, 0 0 1, 0 0 0)),
((1 1 0, 1 1 1, 1 0 1, 1 0 0, 1 1 0)),
((0 1 0, 0 1 1, 1 1 1, 1 1 0, 0 1 0)),
((0 0 1, 1 0 1, 1 1 1, 0 1 1, 0 0 1))
)') AS p_geom ) AS a;

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

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

          wkt
-----
TRIANGLE((0 0 0,0 1 0,1 1 0,0 0 0))
```

Siehe auch

[ST_Dump](#), [ST_NumGeometries](#)

8.5.10 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, dass nicht angezeigt wird, ob die Geometrie eine Maßzahl besitzt.

Erweiterung: Mit 2.0.0 wurde die Unterstützung für polyedrische Oberflächen eingeführt.



This method implements the SQL/MM specification. SQL-MM 3: 5.1.4



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



This function supports Polyhedral surfaces.

Beispiele

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

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

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

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

```
result
-----
ST_Tin
```

Siehe auch

[GeometryType](#)

8.5.11 ST_InteriorRingN

ST_InteriorRingN — Gibt den Nten innenliegenden Linienzug des Ringes der Polygeometrie 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.

Synopsis

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

Beschreibung

Gibt den Nten innenliegenden Linienzug des Ringes der Polygeometrie 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. Der Zeiger beginnt mit der Zahl 1.



Note

Dies funktioniert nicht mit MULTIPOLYGONen. Verwenden Sie die Funktion bitte in Zusammenhang mit ST_Dump um sie auf MULTIPOLYGONE anzuwenden.



This method implements the [OpenGIS Simple Features Implementation Specification for SQL 1.1](#).



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



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

Beispiele

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

Siehe auch

[ST_ExteriorRing](#) [ST_BuildArea](#), [ST_Collect](#), [ST_Dump](#), [ST_NumInteriorRing](#), [ST_NumInteriorRings](#)

8.5.12 ST_IsPolygonCCW

ST_IsPolygonCCW — Gibt TRUE zurück, wenn alle äußeren Ringe gegen den Uhrzeigersinn orientiert sind und alle inneren Ringe im Uhrzeigersinn ausgerichtet sind.

Synopsis

boolean **ST_IsPolygonCCW** (geometry geom);

Beschreibung

Gibt TRUE zurück, wenn für alle Bestandteile der angegebenen Geometrie gilt: die äußeren Ringe sind gegen den Uhrzeigersinn und die inneren Ringe im Uhrzeigersinn ausgerichtet.

Gibt TRUE zurück, wenn die Geometrie keine Polygonbestandteile aufweist.



Note

Da geschlossene Linienzüge nicht als Polygonbestandteile betrachtet werden, erhalten Sie auch dann TRUE, wenn Sie einen einzelnen geschlossenen Linienzug eingeben und zwar unabhängig von dessen Ausrichtung.



Note

Wenn bei einer Polygoneometrie 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.



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



This function supports M coordinates.

Siehe auch

[ST_ForcePolygonCW](#) , [ST_ForcePolygonCCW](#) , [ST_IsPolygonCW](#)

8.5.13 ST_IsPolygonCW

ST_IsPolygonCW — Gibt den Wert TRUE zurück, wenn alle äußeren Ringe im Uhrzeigersinn und alle inneren Ringe gegen den Uhrzeigersinn ausgerichtet sind.

Synopsis

boolean **ST_IsPolygonCW** (geometry geom);

Beschreibung

Gibt den Wert `TRUE` zurück, wenn für alle Polygonbestandteile der eingegebenen Geometrie gilt: die äußeren Ringe sind im Uhrzeigersinn orientiert, die inneren Ringe entgegen dem Uhrzeigersinn.

Gibt `TRUE` zurück, wenn die Geometrie keine Polygonbestandteile aufweist.



Note

Da geschlossene Linienzüge nicht als Polygonbestandteile betrachtet werden, erhalten Sie auch dann `TRUE`, wenn Sie einen einzelnen geschlossenen Linienzug eingeben und zwar unabhängig von dessen Ausrichtung.



Note

Wenn bei einer Polygoneometrie 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.



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



This function supports M coordinates.

Siehe auch

[ST_ForcePolygonCW](#) , [ST_ForcePolygonCCW](#) , [ST_IsPolygonCW](#)

8.5.14 ST_IsClosed

`ST_IsClosed` — Gibt den Wert `TRUE` zurück, wenn die Anfangs- und Endpunkte des `LINESTRING`'s zusammenfallen. Bei polyedrischen Oberflächen, wenn sie geschlossen (volumetrisch) sind.

Synopsis

```
boolean ST_IsClosed(geometry g);
```

Beschreibung

Gibt den Wert `TRUE` zurück, wenn die Anfangs- und Endpunkte des `LINESTRING`'s zusammenfallen. Bei polyedrischen Oberflächen wird angezeigt, ob die Oberfläche eine Fläche (offen) oder ein Volumen (geschlossen) beschreibt.



This method implements the [OpenGIS Simple Features Implementation Specification for SQL 1.1](#).



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



Note

SQL-MM gibt vor, daß das Ergebnis von `ST_IsClosed(NULL)` 0 ergeben soll, während PostGIS `NULL` zurückgibt.



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



This method supports Circular Strings and Curves

Erweiterung: Mit 2.0.0 wurde die Unterstützung für polyedrische Oberflächen eingeführt.



This function supports Polyhedral surfaces.

Beispiele für Linienzüge und Punkte

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

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

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

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

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

Beispiel für eine polyedrische Oberfläche

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

st_isclosed
-----
t

-- Ein Würfel, bei dem eine Seite fehlt --
```

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

stisclosed
-----
f
```

Siehe auch[ST_IsRing](#)**8.5.15 ST_IsCollection**

ST_IsCollection — Gibt den Wert `TRUE` zurück, wenn der Parameter eine Ansammlung/Kollektion von Geometrien ist (`MULTI*`, `GEOMETRYCOLLECTION`, ...)

Synopsis

```
boolean ST_IsCollection(geometry g);
```

Beschreibung

Gibt den Wert `TRUE` zurück, wenn der Geometrietyp einer der folgenden Geometrietypen entspricht:

- `GEOMETRYCOLLECTION`
- `MULTI{POINT,POLYGON,LINestring,CURVE,SURFACE}`
- `COMPOUNDCURVE`

**Note**

Diese Funktion wertet den Geometrietyp aus. D.h.: sie gibt den Wert `TRUE` für Geometriekollektionen zurück, wenn diese leer sind, oder nur ein einziges Element aufweisen.



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



This method supports Circular Strings and Curves

Beispiele

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

```

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

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

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

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

```

Siehe auch[ST_NumGeometries](#)**8.5.16 ST_IsEmpty**

ST_IsEmpty — Gibt den Wert TRUE zurück, falls es sich bei der Geometrie um eine leere GeometryCollection, Polygon, Point etc. handelt.

Synopsis

boolean **ST_IsEmpty**(geometry geomA);

Beschreibung

Gibt den Wert TRUE zurück, wenn es sich um eine leere Geometrie handelt. Falls TRUE, dann repräsentiert diese Geometrie eine leere GeometryCollection, Polygon, Point etc.

**Note**

SQL-MM gibt vor, daß das Ergebnis von ST_IsEmpty(NULL) der Wert 0 ist, während PostGIS den Wert NULL zurückgibt.



This method implements the [OpenGIS Simple Features Implementation Specification for SQL 1.1](#). s2.1.1.1



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



This method supports Circular Strings and Curves

**Warning**

Änderung: 2.0.0 - In Vorgängerversionen von PostGIS war `ST_GeomFromText('GEOMETRYCOLLECTION(EMPTY)')` erlaubt. Um eine bessere Übereinstimmung mit der SQL/MM Norm zu erreichen, ist dies nun nicht mehr gestattet.

Beispiele

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

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

SELECT ST_IsEmpty(ST_GeomFromText('POLYGON((1 2, 3 4, 5 6, 1 2))'));
st_isempty
-----
f
(1 row)

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

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

8.5.17 ST_IsRing

ST_IsRing — Gibt den Wert `TRUE` zurück, wenn der `LINESTRING` geschlossen ist und der Simple Feature Spezifikation entspricht.

Synopsis

boolean **ST_IsRing**(geometry g);

Beschreibung

Gibt den Wert `TRUE` zurück, wenn der `LINESTRING` sowohl **ST_IsClosed** (`ST_StartPoint((g)) ~= ST_Endpoint((g))`) als auch **ST_IsSimple** (sich nicht selbst überschneidet) ist.



This method implements the [OpenGIS Simple Features Implementation Specification for SQL 1.1](#). 2.1.5.1



This method implements the SQL/MM specification. SQL-MM 3: 7.1.6

**Note**

SQL-MM gibt vor, daß das Ergebnis von `ST_IsRing(NULL)` der Wert 0 sein soll, während PostGIS den Wert `NULL` zurückgibt.

Beispiele

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

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

Siehe auch

[ST_IsClosed](#), [ST_IsSimple](#), [ST_StartPoint](#), [ST_EndPoint](#)

8.5.18 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 Simplität und Gültigkeit von Geometrien, siehe "[Ensuring OpenGIS compliancy of geometries](#)"

**Note**

SQL-MM definiert das Ergebnis von `ST_IsSimple(NULL)` als 0, während PostGIS `NULL` zurückgibt.



This method implements the [OpenGIS Simple Features Implementation Specification for SQL 1.1](#). s2.1.1.1



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



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

Beispiele

```
SELECT ST_IsSimple(ST_GeomFromText('POLYGON((1 2, 3 4, 5 6, 1 2))'));
st_issimple
-----
t
(1 row)

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

Siehe auch

[ST_IsValid](#)

8.5.19 ST_IsValid

`ST_IsValid` — Gibt `true` zurück, wenn `ST_Geometry` wohlgeformt ist.

Synopsis

```
boolean ST_IsValid(geometry g);
boolean ST_IsValid(geometry g, integer flags);
```

Beschreibung

Überprüft, ob ein `ST_Geometry` Wert wohlgeformt ist. Bei ungültigen Geometrien liefert PostgreSQL NOTICE die Einzelheiten darüber, warum die Geometrie ungültig ist. Weiterführende Information über die OGC-Definition von Simplität und Gültigkeit von Geometrien finden Sie unter "[Ensuring OpenGIS compliancy of geometries](#)"



Note

SQL-MM definiert das Ergebnis von `ST_IsValid(NULL)` als 0, während PostGIS NULL zurückgibt.

Ab Version 2.0.0 und GEOS $\geq 3.3.0$ werden Flags akzeptiert. Diese Variante gibt keine NOTICE mit der Erklärung der Ungültigkeit aus. Die erlaubten `flags` sind unter [ST_IsValidDetail](#) dokumentiert.



This method implements the [OpenGIS Simple Features Implementation Specification for SQL 1.1](#).



This method implements the SQL/MM specification. SQL-MM 3: 5.1.9



Note

Weder die OGC-SFS Spezifikation, noch die SQL-MM Norm beinhalten ein Argument für `ST_IsValid`. Die Flag ist eine PostGIS-Erweiterung.

Beispiele

```
SELECT ST_IsValid(ST_GeomFromText('LINESTRING(0 0, 1 1)')) As good_line,
       ST_IsValid(ST_GeomFromText('POLYGON((0 0, 1 1, 1 2, 1 1, 0 0))')) As bad_poly
--results
NOTICE: Self-intersection at or near point 0 0
good_line | bad_poly
-----+-----
t         | f
```

Siehe auch

[ST_IsSimple](#), [ST_IsValidReason](#), [ST_IsValidDetail](#), [ST_Summary](#)

8.5.20 ST_IsValidReason

ST_IsValidReason — Gibt an, ob die Geometrie zulässig ist oder nicht; gibt auch die Ursache an falls nicht.

Synopsis

```
text ST_IsValidReason(geometry geomA);
text ST_IsValidReason(geometry geomA, integer flags);
```

Beschreibung

Gibt an, ob die Geometrie zulässig ist oder nicht; gibt auch die Ursache an falls sie nicht zulässig ist.

Sinnvoll in Zusammenhang mit **ST_IsValid**, um einen ausführlichen Bericht, der unzulässigen Geometrien und der Gründe dafür, zu erstellen.

Zugelassene `flags` sind unter [ST_IsValidDetail](#) beschrieben.

Verfügbarkeit: 1.4 - benötigt GEOS >= 3.1.0.

Verfügbarkeit: 2.0 - benötigt GEOS >= 3.3.0 damit die Version Flags annimmt.

Beispiele

```
--Die ersten 3 Ausgaben eines erfolgreichen "Versuchs mit "Fünflingen"/quintuplet ↔
experiment
SELECT gid, ST_IsValidReason(the_geom) as validity_info
FROM
  (SELECT ST_MakePolygon(ST_ExteriorRing(e.buff), ST_Accum(f.line)) As the_geom, gid
  FROM (SELECT ST_Buffer(ST_MakePoint(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_MakePoint(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(the_geom) = false
```

```
ORDER BY gid
LIMIT 3;
```

gid	validity_info
5330	Self-intersection [32 5]
5340	Self-intersection [42 5]
5350	Self-intersection [52 5]

```
--Einfaches Beispiel
SELECT ST_IsValidReason('LINESTRING(220227 150406,2220227 150407,222020 150410)');

st_isvalidreason
-----
Valid Geometry
```

Siehe auch

[ST_IsValid](#), [ST_Summary](#)

8.5.21 ST_IsValidDetail

ST_IsValidDetail — Gibt eine `valid_detail` (Gültigkeit, Ursache, Lage) Zeile aus, die anzeigt, ob die Geometrie gültig ist oder nicht, und falls nicht, die Ursache warum und die Lage wo dies auftritt.

Synopsis

```
valid_detail ST_IsValidDetail(geometry geom);
valid_detail ST_IsValidDetail(geometry geom, integer flags);
```

Beschreibung

Gibt eine `valid_detail` Zeile aus, die aus folgenden Attributen besteht: einer Booleschen Variablen (`valid`), die angibt ob die Geometrie gültig ist; einem Textfeld variabler Zeichenlänge (`reason`), das den Grund angibt weshalb die Geometrie ungültig ist; eine Geometrie (`location`) die anzeigt, wo diese ungültig ist.

Nützlich, um die Kombination von `ST_IsValid` und `ST_IsValidReason` zu ersetzen und zu verbessern. Erstellt einen ausführlichen Bericht über die unzulässigen Geometrien.

Der 'flags' Parameter ist ein Bitfeld und kann folgende Werte annehmen:

- 1: Erachtet sich selbst-überschneidende, Lücken bildende als gültig. self-intersecting rings forming holes as valid. Ist auch als "ESRI flag" bekannt. Beachten Sie bitte, dass dies im Widerspruch zum OGC Modell steht.

Verfügbarkeit: 2.0.0 - benötigt GEOS >= 3.3.0.

Beispiele

```
--Die ersten 3 Ausgaben eines erfolgreichen "Versuchs mit "Fünflingen"/quintuplet ↔
  experiment
SELECT gid, reason(ST_IsValidDetail(the_geom)), ST_AsText(location(ST_IsValidDetail( ↔
  the_geom))) as location
FROM
(SELECT ST_MakePolygon(ST_ExteriorRing(e.buff), ST_Accum(f.line)) As the_geom, gid
```

```

FROM (SELECT ST_Buffer(ST_MakePoint(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_MakePoint(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(the_geom) = false
ORDER BY gid
LIMIT 3;

```

gid	reason	location
5330	Self-intersection	POINT(32 5)
5340	Self-intersection	POINT(42 5)
5350	Self-intersection	POINT(52 5)

```

--Einfaches Beispiel
SELECT * FROM ST_IsValidDetail('LINESTRING(220227 150406,220227 150407,22020 150410)');

valid | reason | location
-----+-----+-----
t      |         |

```

Siehe auch

[ST_IsValid](#), [ST_IsValidReason](#)

8.5.22 ST_M

ST_M — Gibt die M-Koordinate eines Punktes, oder NULL, wenn diese nicht existiert, zurück. Bei der Eingabe muss es sich um eine Punktgeometrie handeln.

Synopsis

```
float ST_M(geometry a_point);
```

Beschreibung

Gibt die M-Koordinate des Punktes zurück, oder NULL wenn keine vorhanden ist. Der Einabewert muss ein Punkt sein.



Note

Dies ist (noch) kein Teil der OGC Spezifikation, wird aber hier aufgeführt um die Liste von Funktionen zum Auslesen von Punktkoordinaten zu vervollständigen.



This method implements the [OpenGIS Simple Features Implementation Specification for SQL 1.1](#).



This method implements the SQL/MM specification.



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

Beispiele

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

Siehe auch

[ST_GeomFromEWKT](#), [ST_X](#), [ST_Y](#), [ST_Z](#)

8.5.23 ST_NDims

ST_NDims — Gibt die Koordinatendimension der Geometrie als Small Int zurück. Der Wertebereich ist 2, 3 oder 4.

Synopsis

integer **ST_NDims**(geometry g1);

Beschreibung

Gibt die Koordinatendimension der Geometrie zurück. PostGIS unterstützt 2- (x,y), 3- (x,y,z) oder 2D mit Kilometrierung - x,y,m, und 4- dimensionalen Raum - 3D mit Kilometrierung x,y,z,m .



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

Beispiele

```
SELECT ST_NDims(ST_GeomFromText('POINT(1 1)')) As d2point,
       ST_NDims(ST_GeomFromEWKT('POINT(1 1 2)')) As d3point,
       ST_NDims(ST_GeomFromEWKT('POINTM(1 1 0.5)')) As d2pointm;

 d2point | d3point | d2pointm
-----+-----+-----
      2 |       3 |       3
```

Siehe auch

[ST_CoordDim](#), [ST_Dimension](#), [ST_GeomFromEWKT](#)

8.5.24 ST_NPoints

ST_NPoints — Gibt die Anzahl der Punkte (Knoten) einer Geometrie zurück.

Synopsis

integer **ST_NPoints**(geometry g1);

Beschreibung

Gibt die Anzahl der Punkte einer Geometrie zurück. Funktioniert für alle Geometrien.

Erweiterung: Mit 2.0.0 wurde die Unterstützung für polyedrische Oberflächen eingeführt.



Note

Vor 1.3.4 ist diese Funktion abgestürzt, wenn die Geometrien CURVES enthalten. Dies wurde mit 1.3.4+ behoben



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



This method supports Circular Strings and Curves



This function supports Polyhedral surfaces.

Beispiele

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

--Polygon im 3D Raum
SELECT ST_NPoints(ST_GeomFromEWKT('LINESTRING(77.29 29.07 1,77.42 29.26 0,77.27 29.31
-1,77.29 29.07 3)'));
--result
4
```

Siehe auch

[ST_NumPoints](#)

8.5.25 ST_NRings

ST_NRings — Wenn es sich bei der Geometrie um ein Polygon oder um ein MultiPolygon handelt, wird die Anzahl der Ringe zurückgegeben.

Synopsis

integer **ST_NRings**(geometry geomA);

Beschreibung

Wenn es sich bei der Geometrie um ein Polygon oder um ein MultiPolygon handelt, wird die Anzahl der Ringe zurückgegeben. Anders als NumInteriorRings werden auch die äußeren Ringe gezählt.



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



This method supports Circular Strings and Curves

Beispiele

```
SELECT ST_NRings(the_geom) As Nrings, ST_NumInteriorRings(the_geom) As ninterrings
      FROM (SELECT ST_GeomFromText('POLYGON((1 2, 3 4, 5 6, 1 2))') As the_geom) As foo;
```

nrings	ninterrings
1	0

(1 row)

Siehe auch

[ST_NumInteriorRings](#)

8.5.26 ST_NumGeometries

ST_NumGeometries — Wenn es sich bei der Geometrie um eine GEOMETRYCOLLECTION (oder MULTI*) handelt, wird die Anzahl der Geometrien zurückgegeben, bei Einzelgeometrien wird 1, ansonsten NULL zurückgegeben.

Synopsis

integer **ST_NumGeometries**(geometry geom);

Beschreibung

Gibt die Anzahl an Geometrien aus. Wenn es sich bei der Geometrie um eine GEOMETRYCOLLECTION (oder MULTI*) handelt, wird die Anzahl der Geometrien zurückgegeben, bei Einzelgeometrien wird 1, ansonsten NULL zurückgegeben.

Erweiterung: Mit 2.0.0 wurde die Unterstützung für polyedrische Oberflächen, Dreiecke und TIN eingeführt.

Änderung: 2.0.0 Bei früheren Versionen wurde NULL zurückgegeben, wenn die Geometrie nicht vom Typ GEOMETRYCOLLECTION/MULTI war. 2.0.0+ gibt nun 1 für Einzelgeometrien, wie POLYGON, LINESTRING, POINT zurück.



This method implements the SQL/MM specification. SQL-MM 3: 9.1.4



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



This function supports Polyhedral surfaces.



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

Beispiele

```
--Frühere Versionen gaben hier den Wert NULL zurück -- ab 2.0.0 wird der Wert 1 ←
zurückgegeben
SELECT ST_NumGeometries(ST_GeomFromText('LINESTRING(77.29 29.07,77.42 29.26,77.27 ←
29.31,77.29 29.07)'));
--result
1

--Beispiel einer Geometriekollektion - Multis zählen als eine Geometrie in einer Kollektion
SELECT ST_NumGeometries(ST_GeomFromEWKT('GEOMETRYCOLLECTION(MULTIPOINT(-2 3 , -2 2),
LINESTRING(5 5 ,10 10),
POLYGON((-7 4.2,-7.1 5,-7.1 4.3,-7 4.2)))'));
--result
3
```

Siehe auch

[ST_GeometryN](#), [ST_Multi](#)

8.5.27 ST_NumInteriorRings

ST_NumInteriorRings — Gibt die Anzahl der inneren Ringe einer Polygoneometrie aus.

Synopsis

integer **ST_NumInteriorRings**(geometry a_polygon);

Beschreibung

Gibt die Anzahl der inneren Ringe einer Polygoneometrie aus. Gibt NULL zurück, wenn die Geometrie kein Polygon ist.



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

Änderung: 2.0.0 - In früheren Versionen war ein MULTIPOLYGON als Eingabe erlaubt, wobei die Anzahl der inneren Ringe des ersten Polygons ausgegeben wurde.

Beispiele

```
--Falls Sie ein normales Polygon haben
SELECT gid, field1, field2, ST_NumInteriorRings(the_geom) AS numholes
FROM sometable;

--Falls Sie Multipolygone haben
--und die Gesamtzahl der inneren Ringe im MULTIPOLYGON wissen wollen
SELECT gid, field1, field2, SUM(ST_NumInteriorRings(the_geom)) AS numholes
FROM (SELECT gid, field1, field2, (ST_Dump(the_geom)).geom As the_geom
      FROM sometable) As foo
GROUP BY gid, field1,field2;
```

Siehe auch

[ST_NumInteriorRing](#)

8.5.28 ST_NumInteriorRing

ST_NumInteriorRing — Gibt die Anzahl der inneren Ringe eines Polygons in der Geometrie aus. Ist ein Synonym für ST_NumInteriorRings.

Synopsis

```
integer ST_NumInteriorRing(geometry a_polygon);
```

Siehe auch

[ST_NumInteriorRings](#)

8.5.29 ST_NumPatches

ST_NumPatches — Gibt die Anzahl der Maschen einer polyedrischen Oberfläche aus. Gibt NULL zurück, wenn es sich nicht um polyedrische Geometrien handelt.

Synopsis

```
integer ST_NumPatches(geometry g1);
```

Beschreibung

Gibt die Anzahl der Maschen einer polyedrischen Oberfläche aus. Gibt NULL zurück, wenn es sich um keine polyedrische Geometrie handelt. Ist ein Synonym für ST_NumGeometries zur Unterstützung der MM Namensgebung. Wenn Ihnen die MM-Konvention egal ist, so ist die Verwendung von ST_NumGeometries schneller.

Verfügbarkeit: 2.0.0



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



This method implements the [OpenGIS Simple Features Implementation Specification for SQL 1.1](#).



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



This function supports Polyhedral surfaces.

Beispiele

```
SELECT ST_NumPatches(ST_GeomFromEWKT('POLYHEDRALSURFACE( ((0 0 0, 0 0 1, 0 1 1, 0 1 0, 0 0 0) ←
0)),
((0 0 0, 0 1 0, 1 1 0, 1 0 0, 0 0 0)), ((0 0 0, 1 0 0, 1 0 1, 0 0 1, 0 0 0) ←
),
((1 1 0, 1 1 1, 1 0 1, 1 0 0, 1 1 0)),
((0 1 0, 0 1 1, 1 1 1, 1 1 0, 0 1 0)), ((0 0 1, 1 0 1, 1 1 1, 0 1 1, 0 0 1) ←
) )''));
--result
6
```

Siehe auch

[ST_GeomFromEWKT](#), [ST_NumGeometries](#)

8.5.30 ST_NumPoints

ST_NumPoints — Gibt die Anzahl der Stützpunkte eines ST_LineString oder eines ST_CircularString zurück.

Synopsis

integer **ST_NumPoints**(geometry g1);

Beschreibung

Gibt die Anzahl der Stützpunkte eines ST_LineString oder eines ST_CircularString zurück. Vor 1.4 funktionierte dies nur mit ST_LineString, wie von der Spezifikation festgelegt. Ab 1.4 aufwärts handelt es sich um einen Alias für ST_NPoints, das die Anzahl der Knoten nicht nur für Linienzüge ausgibt. Erwägen Sie stattdessen die Verwendung von ST_NPoints, das vielseitig ist und mit vielen Geometrietypen funktioniert.



This method implements the [OpenGIS Simple Features Implementation Specification for SQL 1.1](#).



This method implements the SQL/MM specification. SQL-MM 3: 7.2.4

Beispiele

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

Siehe auch

[ST_NPoints](#)

8.5.31 ST_PatchN

ST_PatchN — Gibt die auf 1-basierende n-te Geometrie (Masche) zurück, wenn es sich bei der Geometrie um ein POLYHEDRALSURFACE, oder ein POLYHEDRALSURFACEM handelt. Anderenfalls wird NULL zurückgegeben.

Synopsis

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

Beschreibung

>Gibt die auf 1-basierende n-te Geometrie (Masche) zurück, wenn es sich bei der Geometrie um ein POLYHEDRALSURFACE, oder ein POLYHEDRALSURFACEM handelt. Anderenfalls wird NULL zurückgegeben. Gibt bei polyedrischen Oberflächen das selbe Ergebnis wie ST_GeometryN. Die Verwendung von ST_GeometryN ist schneller.



Note

Der Index ist auf 1 basiert.

**Note**

Falls Sie alle Geometrien einer Geometrie entnehmen wollen, so ist `ST_Dump` wesentlich leistungsfähiger.

Verfügbarkeit: 2.0.0



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



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



This function supports Polyhedral surfaces.

Beispiele

```
--Entnimmt die 2te Fläche einer polyedrischen Oberfläche
SELECT ST_AsEWKT(ST_PatchN(geom, 2)) As geomewkt
FROM (
VALUES (ST_GeomFromEWKT('POLYHEDRALSURFACE( ((0 0 0, 0 0 1, 0 1 1, 0 1 0, 0 0 0)),
        ((0 0 0, 0 1 0, 1 1 0, 1 0 0, 0 0 0)), ((0 0 0, 1 0 0, 1 0 1, 0 0 1, 0 0 0)),
        ((1 1 0, 1 1 1, 1 0 1, 1 0 0, 1 1 0)),
        ((0 1 0, 0 1 1, 1 1 1, 1 1 0, 0 1 0)), ((0 0 1, 1 0 1, 1 1 1, 0 1 1, 0 0 1)) )' ) ) ) ←
        As foo(geom);

          geomewkt
-----+-----
POLYGON((0 0 0,0 1 0,1 1 0,1 0 0,0 0 0))
```

Siehe auch

[ST_AsEWKT](#), [ST_GeomFromEWKT](#), [ST_Dump](#), [ST_GeometryN](#), [ST_NumGeometries](#)

8.5.32 ST_PointN

`ST_PointN` — Gibt den n-ten Punkt des ersten LineString's oder des kreisförmigen LineStrings's in der Geometrie zurück. Negative Werte werden rückwärts vom Ende des LineString's gezählt. Gibt NULL aus, wenn es sich bei der Geometrie nicht um einen LineString handelt.

Synopsis

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

Beschreibung

Gibt den n-ten Punkt des ersten LineString's oder des kreisförmigen LineStrings's einer Geometrie zurück. Negative Werte werden rückwärts, vom Ende des LineString's her gezählt, sodass -1 der Endpunkt ist. Gibt NULL aus, wenn die Geometrie keinen LineString enthält.

**Note**

Seit Version 0.8.0 ist der Index 1-basiert, so wie in der OGC Spezifikation. Rückwärtiges Indizieren (negativer Index) findet sich nicht in der OGC Spezifikation. Vorhergegangene Versionen waren 0-basiert.

**Note**

Falls Sie den n-ten Punkt eines jeden LineString's in einem MultiLinestring wollen, nutzen Sie diese Funktion gemeinsam mit ST_Dump.



This method implements the [OpenGIS Simple Features Implementation Specification for SQL 1.1](#).



This method implements the SQL/MM specification. SQL-MM 3: 7.2.5, 7.3.5



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



This method supports Circular Strings and Curves

**Note**

Änderung: 2.0.0 arbeitet nicht mehr mit MultiLinestring's, die nur eine einzelne Geometrie enthalten. In früheren Versionen von PostGIS gab die Funktion bei einem, aus einer einzelnen Linie bestehender MultiLinestring, den Anfangspunkt zurück. Ab 2.0.0 wird, so wie bei jedem anderen MultiLinestring auch, NULL zurückgegeben.

Änderung: 2.3.0 : negatives Indizieren verfügbar (-1 entspricht dem Endpunkt)

Beispiele

```
-- Entnimmt alle POINTs eines LINESTRINGs
SELECT ST_AsText(
  ST_PointN(
    column1,
    generate_series(1, ST_NPoints(column1))
  ))
FROM ( VALUES ('LINESTRING(0 0, 1 1, 2 2)::geometry') AS foo;

st_astext
-----
POINT(0 0)
POINT(1 1)
POINT(2 2)
(3 rows)

--Beispiel für einen Kreisbogen
SELECT ST_AsText(ST_PointN(ST_GeomFromText('CIRCULARSTRING(1 2, 3 2, 1 2)'),2));

st_astext
-----
POINT(3 2)

SELECT st_astext(f)
FROM ST_GeometryFromText('LINESTRING(0 0 0, 1 1 1, 2 2 2)') as g
    ,ST_PointN(g, -2) AS f -- 1 based index

st_astext
-----
"POINT Z (1 1 1)"
```

Siehe auch[ST_NPoints](#)**8.5.33 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, welcher alle Koordinaten einer Geometrie enthält. Duplizierte Punkte in der Eingabegeometrie, einschließlich der Anfangs- und Endpunkte von Ringgeometrien, werden nicht entfernt. (Wenn diese Verhaltensweise unerwünscht ist, können die Duplikate mit [ST_RemoveRepeatedPoints](#) entfernt werden).

Vorhandene M- und Z-Ordinaten werden erhalten.



This method supports Circular Strings and Curves



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

Verfügbarkeit: 2.3.0

Beispiele

```
SELECT ST_AsText(ST_Points('POLYGON Z ((30 10 4,10 30 5,40 40 6, 30 10))'));  
  
--result  
MULTIPOINT Z (30 10 4,10 30 5,40 40 6, 30 10 4)
```

Siehe auch[ST_RemoveRepeatedPoints](#)**8.5.34 ST_SRID**

ST_SRID — Gibt den Identifikator des Koordinatenreferenzsystems, wie in der spatial_ref_sys Tabelle definiert, für die ST_Geometry aus.

Synopsis

```
integer ST_SRID(geometry g1);
```

Beschreibung

Gibt den Identifikator des Koordinatenreferenzsystems, wie in der `spatial_ref_sys` Tabelle definiert, für die `ST_Geometry` aus. Section 4.3.1



Note

In der `spatial_ref_sys` Tabelle werden alle Koordinatenreferenzsysteme, die PostGIS bekannt sind, katalogisiert und für Koordinatentransformationen herangezogen. Falls Sie jemals Koordinatentransformationen mit Ihren Geometrien durchführen wollen, sollten Sie sich vergewissern, dass die richtigen Identifikatoren für die Koordinatenreferenzsysteme dieser Geometrien eingetragen sind.



This method implements the [OpenGIS Simple Features Implementation Specification for SQL 1.1](#). s2.1.1.1



This method implements the SQL/MM specification. SQL-MM 3: 5.1.5



This method supports Circular Strings and Curves

Beispiele

```
SELECT ST_SRID(ST_GeomFromText('POINT(-71.1043 42.315)', 4326));
--result
4326
```

Siehe auch

Section 4.3.1, [ST_GeomFromText](#), [ST_SetSRID](#), [ST_Transform](#)

8.5.35 ST_StartPoint

`ST_StartPoint` — Gibt den den Anfangspunkt einer `LINESTRING` Geometrie als `POINT` zurück.

Synopsis

geometry **ST_StartPoint**(geometry geomA);

Beschreibung

Gibt den Anfangspunkt einer `LINESTRING` oder `CIRCULARLINESTRING` Geometrie als `POINT` oder `NULL` zurück, falls es sich beim Eingabewert nicht um einen `LINESTRING` oder `CIRCULARLINESTRING` handelt.



This method implements the SQL/MM specification. SQL-MM 3: 7.1.3



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



This method supports Circular Strings and Curves

**Note**

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

```
SELECT ST_AsText(ST_StartPoint('LINESTRING(0 1, 0 2)::geometry'));
st_astext
-----
POINT(0 1)
(1 row)

SELECT ST_StartPoint('POINT(0 1)::geometry') IS NULL AS is_null;
is_null
-----
t
(1 row)

--3D Linie
SELECT ST_AsEWKT(ST_StartPoint('LINESTRING(0 1 1, 0 2 2)::geometry'));
st_asewkt
-----
POINT(0 1 1)
(1 row)

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

Siehe auch

[ST_EndPoint](#), [ST_PointN](#)

8.5.36 ST_Summary

ST_Summary — Gibt eine Zusammenfassung des Inhalts einer Geometrie wieder.

Synopsis

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

Beschreibung

Gibt eine Zusammenfassung des Inhalts einer Geometrie wieder.

Die Bedeutung der Flags, welche in eckigen Klammern hinter dem Geometrietyp angezeigt werden, ist wie folgt:

- M: besitzt eine M-Ordinate
- Z: besitzt eine Z-Ordinate
- B: besitzt ein zwischengespeichertes Umgebungsrechteck
- G: ist geodätisch (Geographie)
- S: besitzt ein räumliches Koordinatenreferenzsystem



This method supports Circular Strings and Curves



This function supports Polyhedral surfaces.



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

Verfügbarkeit: 1.2.2

Erweiterung: 2.0.0 Unterstützung für geographische Koordinaten hinzugefügt

Erweiterung: 2.1.0 S-Flag, diese zeigt an ob das Koordinatenreferenzsystem bekannt ist

Erweiterung: 2.2.0 Unterstützung für TIN und Kurven

Beispiele

```
=# SELECT ST_Summary(ST_GeomFromText('LINESTRING(0 0, 1 1)')) as geom,
          ST_Summary(ST_GeogFromText('POLYGON((0 0, 1 1, 1 2, 1 1, 0 0))')) geog;
-----+-----
LineString[B] with 2 points | Polygon[BGS] with 1 rings
                           | ring 0 has 5 points
                           :
(1 row)

=# SELECT ST_Summary(ST_GeogFromText('LINESTRING(0 0 1, 1 1 1)')) As geog_line,
          ST_Summary(ST_GeomFromText('SRID=4326;POLYGON((0 0 1, 1 1 2, 1 2 3, 1 1 1, 0 0 1)) ←
          ')) As geom_poly;
;
-----+-----
LineString[ZBGS] with 2 points | Polygon[ZBS] with 1 rings
                           :   ring 0 has 5 points
                           :
(1 row)
```

Siehe auch

[PostGIS_DropBBBox](#), [PostGIS_AddBBBox](#), [ST_Force3DM](#), [ST_Force3DZ](#), [ST_Force2D](#), [geography](#)
[ST_IsValid](#), [ST_IsValidReason](#), [ST_IsValidDetail](#)

8.5.37 ST_X

ST_X — Gibt die X-Koordinate eines Punktes, oder NULL wenn diese nicht vorhanden ist, zurück. Die Eingabe muss ein Punkt sein.

Synopsis

```
float ST_X(geometry a_point);
```

Beschreibung

Gibt die X-Koordinate eines Punktes, oder NULL wenn diese nicht vorhanden ist, zurück. Die Eingabe muss ein Punkt sein.

**Note**

Für die maximalen bzw. minimalen Werte einer beliebigen Geometrie benötigen, sh. die Funktionen ST_XMin und ST_XMax.



This method implements the SQL/MM specification. SQL-MM 3: 6.1.3



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

Beispiele

```
SELECT ST_X(ST_GeomFromEWKT('POINT(1 2 3 4)'));
 st_x
-----
      1
(1 row)

SELECT ST_Y(ST_Centroid(ST_GeomFromEWKT('LINESTRING(1 2 3 4, 1 1 1 1)')));
 st_y
-----
    1.5
(1 row)
```

Siehe auch

[ST_Centroid](#), [ST_GeomFromEWKT](#), [ST_M](#), [ST_XMax](#), [ST_XMin](#), [ST_Y](#), [ST_Z](#)

8.5.38 ST_XMax

ST_XMax — Gibt das größte X des Umgebungsrechtecks in 2D, 3D oder als Geometrie zurück.

Synopsis

```
float ST_XMax(box3d aGeomorBox2DorBox3D);
```

Beschreibung

Gibt das größte X des Umgebungsrechtecks in 2D, 3D oder als Geometrie zurück.

**Note**

Obwohl diese Funktion nur für Box3D definiert ist, funktioniert sie auch für Box2D und für Geometrie, da eine automatische Typumwandlung für Geometrie und Box2D festgelegt ist. Allerdings kann eine Geometrie oder Box2D nicht in der Textdarstellung eingegeben werden, da es dann zu keiner automatischen Typumwandlung kommt.



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



This method supports Circular Strings and Curves

Beispiele

```
SELECT ST_XMax('BOX3D(1 2 3, 4 5 6)');
st_xmax
-----
4

SELECT ST_XMax(ST_GeomFromText('LINESTRING(1 3 4, 5 6 7)'));
st_xmax
-----
5

SELECT ST_XMax(CAST('BOX(-3 2, 3 4)' As box2d));
st_xmax
-----
3
--DIES FUNKTIONIERT NICHT, da eine automatische Typumwandlung von einer Zeichenkette auf ↔
  BOX3D erfolgt.
SELECT ST_XMax('LINESTRING(1 3, 5 6)');

--ERROR:  BOX3D parser - doesn't start with BOX3D(

SELECT ST_XMax(ST_GeomFromEWKT('CIRCULARSTRING(220268 150415 1,220227 150505 2,220227 ↔
  150406 3)'));
st_xmax
-----
220288.248780547
```

Siehe auch

[ST_XMin](#), [ST_YMax](#), [ST_YMin](#), [ST_ZMax](#), [ST_ZMin](#)

8.5.39 ST_XMin

ST_XMin — Gibt das kleinste X des Umgebungsrechtecks in 2D, 3D oder als Geometrie zurück.

Synopsis

float **ST_XMin**(box3d aGeomorBox2DorBox3D);

Beschreibung

Gibt das kleinste X des Umgebungsrechtecks in 2D, 3D oder als Geometrie zurück.



Note

Obwohl diese Funktion nur für Box3D definiert ist, funktioniert sie auch für Box2D und für Geometrie, da eine automatische Typumwandlung für Geometrie und Box2D festgelegt ist. Allerdings kann eine Geometrie oder Box2D nicht in der Textdarstellung eingegeben werden, da es dann zu keiner automatischen Typumwandlung kommt.



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



This method supports Circular Strings and Curves

Beispiele

```
SELECT ST_XMin('BOX3D(1 2 3, 4 5 6)');
st_xmin
-----
1

SELECT ST_XMin(ST_GeomFromText('LINESTRING(1 3 4, 5 6 7)'));
st_xmin
-----
1

SELECT ST_XMin(CAST('BOX(-3 2, 3 4)' As box2d));
st_xmin
-----
-3
--DIES FUNKTIONIERT NICHT, da eine automatische Typumwandlung von einer Zeichenkette auf ↔
BOX3D erfolgt.
SELECT ST_XMin('LINESTRING(1 3, 5 6)');

--ERROR: BOX3D parser - doesn't start with BOX3D(

SELECT ST_XMin(ST_GeomFromEWKT('CIRCULARSTRING(220268 150415 1,220227 150505 2,220227 ↔
150406 3)'));
st_xmin
-----
220186.995121892
```

Siehe auch

[ST_XMax](#), [ST_YMax](#), [ST_YMin](#), [ST_ZMax](#), [ST_ZMin](#)

8.5.40 ST_Y

ST_Y — Gibt die Y-Koordinate eines Punktes, oder NULL wenn diese nicht vorhanden ist, zurück. Die Eingabe muss ein Punkt sein.

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.



This method implements the [OpenGIS Simple Features Implementation Specification for SQL 1.1](#).



This method implements the SQL/MM specification. SQL-MM 3: 6.1.4



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

Beispiele

```
SELECT ST_Y(ST_GeomFromEWKT('POINT(1 2 3 4)'));
st_y
-----
      2
(1 row)

SELECT ST_Y(ST_Centroid(ST_GeomFromEWKT('LINESTRING(1 2 3 4, 1 1 1 1)')));
st_y
-----
   1.5
(1 row)
```

Siehe auch

[ST_Centroid](#), [ST_GeomFromEWKT](#), [ST_M](#), [ST_X](#), [ST_YMax](#), [ST_YMin](#), [ST_Z](#)

8.5.41 ST_YMax

ST_YMax — Gibt das größte Y des Umgebungsrechtecks in 2D, 3D oder als Geometrie zurück.

Synopsis

```
float ST_YMax(box3d aGeomorBox2DorBox3D);
```

Beschreibung

Gibt das größte Y des Umgebungsrechtecks in 2D, 3D oder als Geometrie zurück.



Note

Obwohl diese Funktion nur für Box3D definiert ist, funktioniert sie auch für Box2D und für Geometrie, da eine automatische Typumwandlung für Geometrie und Box2D festgelegt ist. Allerdings kann eine Geometrie oder Box2D nicht in der Textdarstellung eingegeben werden, da es dann zu keiner automatischen Typumwandlung kommt.



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



This method supports Circular Strings and Curves

Beispiele

```
SELECT ST_YMax('BOX3D(1 2 3, 4 5 6)');
st_ymax
-----
5

SELECT ST_YMax(ST_GeomFromText('LINESTRING(1 3 4, 5 6 7)'));
st_ymax
-----
6

SELECT ST_YMax(CAST('BOX(-3 2, 3 4)' As box2d));
st_ymax
-----
4
--DIES FUNKTIONIERT NICHT, da eine automatische Typumwandlung von einer Zeichenkette auf ↩
BOX3D erfolgt.
SELECT ST_YMax('LINESTRING(1 3, 5 6)');

--ERROR: BOX3D parser - doesn't start with BOX3D(

SELECT ST_YMax(ST_GeomFromEWKT('CIRCULARSTRING(220268 150415 1,220227 150505 2,220227 ↩
150406 3)'));
st_ymax
-----
150506.126829327
```

Siehe auch

[ST_XMin](#), [ST_XMax](#), [ST_YMin](#), [ST_ZMax](#), [ST_ZMin](#)

8.5.42 ST_YMin

ST_YMin — Gibt das kleinste Y des Umgebungsrechtecks in 2D, 3D oder als Geometrie zurück.

Synopsis

float **ST_YMin**(box3d aGeomorBox2DorBox3D);

Beschreibung

Gibt das kleinste Y des Umgebungsrechtecks in 2D, 3D oder als Geometrie zurück.



Note

Obwohl diese Funktion nur für Box3D definiert ist, funktioniert sie auch für Box2D und für Geometrie, da eine automatische Typumwandlung für Geometrie und Box2D festgelegt ist. Allerdings kann eine Geometrie oder Box2D nicht in der Textdarstellung eingegeben werden, da es dann zu keiner automatischen Typumwandlung kommt.



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



This method supports Circular Strings and Curves

Beispiele

```
SELECT ST_YMin('BOX3D(1 2 3, 4 5 6)');
st_ymin
-----
2

SELECT ST_YMin(ST_GeomFromText('LINESTRING(1 3 4, 5 6 7)'));
st_ymin
-----
3

SELECT ST_YMin(CAST('BOX(-3 2, 3 4)' As box2d));
st_ymin
-----
2
--DIES FUNKTIONIERT NICHT, da eine automatische Typumwandlung von einer Zeichenkette auf ↩
BOX3D erfolgt
SELECT ST_YMin('LINESTRING(1 3, 5 6)');

--ERROR: BOX3D parser - doesn't start with BOX3D(

SELECT ST_YMin(ST_GeomFromEWKT('CIRCULARSTRING(220268 150415 1,220227 150505 2,220227 ↩
150406 3)'));
st_ymin
-----
150406
```

Siehe auch

[ST_GeomFromEWKT](#), [ST_XMin](#), [ST_XMax](#), [ST_YMax](#), [ST_ZMax](#), [ST_ZMin](#)

8.5.43 ST_Z

ST_Z — Gibt die Z-Koordinate eines Punktes, oder NULL wenn diese nicht vorhanden ist, zurück. Die Eingabe muss ein Punkt sein.

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.



This method implements the SQL/MM specification.



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

Beispiele

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

Siehe auch

[ST_GeomFromEWKT](#), [ST_M](#), [ST_X](#), [ST_Y](#), [ST_ZMax](#), [ST_ZMin](#)

8.5.44 ST_ZMax

ST_ZMax — Gibt das kleinste Z des Umgebungsrechtecks in 2D, 3D oder als Geometrie zurück.

Synopsis

```
float ST_ZMax(box3d aGeomorBox2DorBox3D);
```

Beschreibung

Gibt das größte Z des Umgebungsrechtecks in 2D, 3D oder als Geometrie zurück.

**Note**

Obwohl diese Funktion nur für Box3D definiert ist, funktioniert sie auch für Box2D und für Geometrie, da eine automatische Typumwandlung für Geometrie und Box2D festgelegt ist. Allerdings kann eine Geometrie oder Box2D nicht in der Textdarstellung eingegeben werden, da es dann zu keiner automatischen Typumwandlung kommt.



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



This method supports Circular Strings and Curves

Beispiele

```
SELECT ST_ZMax('BOX3D(1 2 3, 4 5 6)');
 st_zmax
-----
      6

SELECT ST_ZMax(ST_GeomFromEWKT('LINESTRING(1 3 4, 5 6 7)'));
 st_zmax
-----
      7

SELECT ST_ZMax('BOX3D(-3 2 1, 3 4 1)');
 st_zmax
-----
      1
--DIES FUNKTIONIERT NICHT, da eine automatische Typumwandlung von einer Zeichenkette auf ↔
  BOX3D erfolgt
SELECT ST_ZMax('LINESTRING(1 3 4, 5 6 7)');
```

```
--ERROR: BOX3D parser - doesn't start with BOX3D(

SELECT ST_ZMax(ST_GeomFromEWKT('CIRCULARSTRING(220268 150415 1,220227 150505 2,220227 150406 3)'));
st_zmax
-----
3
```

Siehe auch

[ST_GeomFromEWKT](#), [ST_XMin](#), [ST_XMax](#), [ST_YMax](#), [ST_YMin](#), [ST_ZMax](#)

8.5.45 ST_Zmflag

ST_Zmflag — Gibt die ZM (semantische Dimension) Flag der Geometrie als Small Int zurück. Die Werte sind: 0=2D, 1=3DM, 2=3DZ, 3=4D.

Synopsis

```
smallint ST_Zmflag(geometry geomA);
```

Beschreibung

Gibt die ZM (semantische Dimension) Flag der Geometrie als Small Int zurück. Die Werte sind: 0=2D, 1=3DM, 2=3DZ, 3=4D.



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



This method supports Circular Strings and Curves

Beispiele

```
SELECT ST_Zmflag(ST_GeomFromEWKT('LINESTRING(1 2, 3 4)'));
st_zmflag
-----
0

SELECT ST_Zmflag(ST_GeomFromEWKT('LINESTRINGM(1 2 3, 3 4 3)'));
st_zmflag
-----
1

SELECT ST_Zmflag(ST_GeomFromEWKT('CIRCULARSTRING(1 2 3, 3 4 3, 5 6 3)'));
st_zmflag
-----
2

SELECT ST_Zmflag(ST_GeomFromEWKT('POINT(1 2 3 4)'));
st_zmflag
-----
3
```

Siehe auch

[ST_CoordDim](#), [ST_NDims](#), [ST_Dimension](#)

8.5.46 ST_ZMin

ST_ZMin — Gibt das kleinste Z des Umgebungsrechtecks in 2D, 3D oder als Geometrie zurück.

Synopsis

```
float ST_ZMin(box3d aGeomorBox2DorBox3D);
```

Beschreibung

Gibt das kleinste Z des Umgebungsrechtecks in 2D, 3D oder als Geometrie zurück.

**Note**

Obwohl diese Funktion nur für Box3D definiert ist, funktioniert sie auch für Box2D und für Geometrie, da eine automatische Typumwandlung für Geometrie und Box2D festgelegt ist. Allerdings kann eine Geometrie oder Box2D nicht in der Textdarstellung eingegeben werden, da es dann zu keiner automatischen Typumwandlung kommt.



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



This method supports Circular Strings and Curves

Beispiele

```
SELECT ST_ZMin('BOX3D(1 2 3, 4 5 6)');
st_zmin
-----
3

SELECT ST_ZMin(ST_GeomFromEWKT('LINESTRING(1 3 4, 5 6 7)'));
st_zmin
-----
4

SELECT ST_ZMin('BOX3D(-3 2 1, 3 4 1) ');
st_zmin
-----
1
--DIES FUNKTIONIERT NICHT, da eine automatische Typumwandlung von einer Zeichenkette auf ↩
BOX3D erfolgt
SELECT ST_ZMin('LINESTRING(1 3 4, 5 6 7)');
--ERROR: BOX3D parser - doesn't start with BOX3D(

SELECT ST_ZMin(ST_GeomFromEWKT('CIRCULARSTRING(220268 150415 1,220227 150505 2,220227 ↩
150406 3)'));
st_zmin
-----
1
```


Siehe auch

[ST_GeomFromEWKT](#), [ST_GeomFromText](#), [ST_XMin](#), [ST_XMax](#), [ST_YMax](#), [ST_YMin](#), [ST_ZMax](#)

8.6 Geometrische Editoren

8.6.1 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);

Beschreibung

Fügt einen Punkt zu einem Linienzug hinzu; vor point <position> (Index zählt von 0 weg). Der dritte Parameter kann weggelassen werden oder zum Anhängen auf -1 gesetzt werden.

Verfügbarkeit: 1.1.0



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

Beispiele

```
--sicherstellen, das alle Linienzüge in einer Tabelle geschlossen sind,
--indem zu jedem nicht geschlossenen Linienzug, der Anfangspunkt als Endpunkt hinzugefügt wird
UPDATE sometable
SET the_geom = ST_AddPoint(the_geom, ST_StartPoint(the_geom))
FROM sometable
WHERE ST_IsClosed(the_geom) = false;

--Einen Punkt zu einer 3-D Linie hinzufügen
SELECT ST_AsEWKT(ST_AddPoint(ST_GeomFromEWKT('LINESTRING(0 0 1, 1 1 1)'),
    ST_MakePoint(1, 2, 3)));

--result
st_asewkt
-----
LINESTRING(0 0 1,1 1 1,1 2 3)
```

Siehe auch

[ST_RemovePoint](#), [ST_SetPoint](#)

8.6.2 ST_Affine

ST_Affine — Wendet eine affine 3D-Transformation auf die 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, rotieren und skalieren in einem Schritt zu erledigen.

Version 1: Der Aufruf

```
ST_Affine(geom, a, b, c, d, e, f, g, h, i, xoff, yoff, zoff)
```

repräsentiert die Transformationsmatrix

```
/ a b c xoff \
| d e f yoff |
| g h i zoff |
\ 0 0 0 1 /
```

und die Knoten werden wie folgt transformiert:

```
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 unteren verschiebenden / skalierenden Funktionen werden durch eine solche affine Transformation ausgedrückt.

Version 2: Wendet eine affine 2D-Transformation auf die Geometrie an. Der Aufruf

```
ST_Affine(geom, a, b, d, e, xoff, yoff)
```

stellt die Transformationsmatrix dar

```
/ 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 Knoten werden wie folgt transformiert:

```
x' = a*x + b*y + xoff
y' = d*x + e*y + yoff
z' = z
```

Dies ist ein Spezialfall der 3D-Methode oberhalb.

Erweiterung: Mit 2.0.0 wurde die Unterstützung für polyedrische Oberflächen, Dreiecke und TIN eingeführt.

Verfügbarkeit: 1.1.2. Mit 1.2.2 wurde die Bezeichnung von Affine auf ST_Affine geändert



Note

Vor 1.3.4 ist diese Funktion abgestürzt, wenn die Geometrien CURVES enthalten. Dies wurde mit 1.3.4+ behoben



This function supports Polyhedral surfaces.



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



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



This method supports Circular Strings and Curves

Beispiele

```
--Eine Linie um 180 Grad um die Z-Achse drehen/rotieren. Dies ist die lange Version von ST_Rotate();
SELECT ST_AsEWKT(ST_Affine(the_geom, cos(pi()), -sin(pi()), 0, sin(pi()), cos(pi()), 0, 0, 0, 1, 0, 0, 0)) As using_affine,
       ST_AsEWKT(ST_Rotate(the_geom, pi())) As using_rotate
FROM (SELECT ST_GeomFromEWKT('LINESTRING(1 2 3, 1 4 3)') As the_geom) As foo;
      using_affine      |      using_rotate
-----+-----
LINESTRING(-1 -2 3,-1 -4 3) | LINESTRING(-1 -2 3,-1 -4 3)
(1 row)

--Eine Linie um 180 Grad um die X-Achse und um die Z-Achse drehen.
SELECT ST_AsEWKT(ST_Affine(the_geom, cos(pi()), -sin(pi()), 0, sin(pi()), cos(pi()), -sin(pi()), 0, sin(pi()), cos(pi()), 0, 0, 0))
FROM (SELECT ST_GeomFromEWKT('LINESTRING(1 2 3, 1 4 3)') As the_geom) As foo;
      st_asewkt
-----
LINESTRING(-1 -2 -3,-1 -4 -3)
(1 row)
```

Siehe auch

[ST_Rotate](#), [ST_Scale](#), [ST_Translate](#), [ST_TransScale](#)

8.6.3 ST_Force2D

ST_Force2D — Die Geometrien in einen "2-dimensionalen Modus" zwingen.

Synopsis

geometry **ST_Force2D**(geometry geomA);

Beschreibung

Zwingt die Geometrien in einen "2-dimensionalen Modus", sodass in der Ausgabe nur die X- und Y-Koordinaten dargestellt werden. Nützlich um eine OGC-konforme Ausgabe zu erhalten (da OGC nur 2-D Geometrien spezifiziert).

Erweiterung: Mit 2.0.0 wurde die Unterstützung für polyedrische Oberflächen eingeführt.

Änderung: 2.1.0. Bis zu 2.0.x wurde diese Funktion mit ST_Force_2D bezeichnet.



This method supports Circular Strings and Curves



This function supports Polyhedral surfaces.



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

Beispiele

```
SELECT ST_AsEWKT(ST_Force2D(ST_GeomFromEWKT('CIRCULARSTRING(1 1 2, 2 3 2, 4 5 2, 6 7 2, 5 6 2)')));
          st_asewkt
-----
CIRCULARSTRING(1 1,2 3,4 5,6 7,5 6)

SELECT ST_AsEWKT(ST_Force2D('POLYGON((0 0 2,0 5 2,5 0 2,0 0 2),(1 1 2,3 1 2,1 3 2,1 1 2))'));
          st_asewkt
-----
POLYGON((0 0,0 5,5 0,0 0),(1 1,3 1,1 3,1 1))
```

Siehe auch

[ST_Force3D](#)

8.6.4 ST_Force3D

ST_Force3D — Zwingt die Geometrien in einen XYZ Modus. Dies ist ein Alias für ST_Force3DZ.

Synopsis

geometry **ST_Force3D**(geometry geomA);

Beschreibung

Zwingt die Geometrien in einen XYZ Modus. Dies ist ein Alias für ST_Force3DZ. Wenn die Geometrie keine Z-Komponente aufweist, wird eine Z-Koordinate mit dem Wert 0 angeheftet.

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.



This function supports Polyhedral surfaces.



This method supports Circular Strings and Curves



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

Beispiele

```
--Wenn bereits eine 3D-Geometrie vorliegt, geschieht nichts
SELECT ST_AsEWKT(ST_Force3D(ST_GeomFromEWKT('CIRCULARSTRING(1 1 2, 2 3 2, 4 5 2, 6 7 2, 5 6 2)')));
           st_asewkt
-----
CIRCULARSTRING(1 1 2,2 3 2,4 5 2,6 7 2,5 6 2)

SELECT ST_AsEWKT(ST_Force3D('POLYGON((0 0,0 5,5 0,0 0),(1 1,3 1,1 3,1 1))'));
           st_asewkt
-----
POLYGON((0 0 0,0 5 0,5 0 0,0 0 0),(1 1 0,3 1 0,1 3 0,1 1 0))
```

Siehe auch

[ST_AsEWKT](#), [ST_Force2D](#), [ST_Force3DM](#), [ST_Force3DZ](#)

8.6.5 ST_Force3DZ

ST_Force3DZ — Zwingt die Geometrien in einen XYZ Modus.

Synopsis

geometry **ST_Force3DZ**(geometry geomA);

Beschreibung

Zwingt die Geometrien in einen XYZ Modus. Dies ist ein Alias für ST_Force3DZ. Wenn die Geometrie keine Z-Komponente aufweist, wird eine Z-Koordinate mit dem Wert 0 angeheftet.

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.



This function supports Polyhedral surfaces.



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



This method supports Circular Strings and Curves

Beispiele

```
--Wenn bereits eine 3D-Geometrie vorliegt, geschieht nichts
SELECT ST_AsEWKT(ST_Force3DZ(ST_GeomFromEWKT('CIRCULARSTRING(1 1 2, 2 3 2, 4 5 2, 6 7 2, 5 6 2)')));
           st_asewkt
-----
CIRCULARSTRING(1 1 2,2 3 2,4 5 2,6 7 2,5 6 2)

SELECT ST_AsEWKT(ST_Force3DZ('POLYGON((0 0,0 5,5 0,0 0),(1 1,3 1,1 3,1 1))'));
           st_asewkt
-----
POLYGON((0 0 0,0 5 0,5 0 0,0 0 0),(1 1 0,3 1 0,1 3 0,1 1 0))
```

```

-----st_asewkt-----
POLYGON((0 0 0,0 5 0,5 0 0,0 0 0),(1 1 0,3 1 0,1 3 0,1 1 0))

```

Siehe auch

[ST_AsEWKT](#), [ST_Force2D](#), [ST_Force3DM](#), [ST_Force3D](#)

8.6.6 ST_Force3DM

ST_Force3DM — Zwingt die Geometrien in einen XYM Modus.

Synopsis

geometry **ST_Force3DM**(geometry geomA);

Beschreibung

Zwingt die Geometrien in einen XYM Modus. Wenn die Geometrie keine M-Komponente aufweist, wird eine M-Koordinate mit dem Wert 0 angeheftet. Falls die Geometrie eine Z-Komponente aufweist, wird diese gelöscht.

Änderung: 2.1.0. Bis zu 2.0.x wurde diese Funktion mit `ST_Force_3DM` bezeichnet.



This method supports Circular Strings and Curves

Beispiele

```

----Wenn bereits eine 3D-Geometrie vorliegt, geschieht nichts
SELECT ST_AsEWKT(ST_Force3DM(ST_GeomFromEWKT('CIRCULARSTRING(1 1 2, 2 3 2, 4 5 2, 6 7 2, 5 6 2)')));
-----st_asewkt-----
CIRCULARSTRINGM(1 1 0,2 3 0,4 5 0,6 7 0,5 6 0)

SELECT ST_AsEWKT(ST_Force3DM('POLYGON((0 0 1,0 5 1,5 0 1,0 0 1),(1 1 1,3 1 1,1 3 1,1 1 1))'));
-----st_asewkt-----
POLYGONM((0 0 0,0 5 0,5 0 0,0 0 0),(1 1 0,3 1 0,1 3 0,1 1 0))

```

Siehe auch

[ST_AsEWKT](#), [ST_Force2D](#), [ST_Force3DM](#), [ST_Force3D](#), [ST_GeomFromEWKT](#)

8.6.7 ST_Force4D

ST_Force4D — Zwingt die Geometrien in einen XYZM Modus.

Synopsis

geometry **ST_Force4D**(geometry geomA);

Beschreibung

Zwingt die Geometrien in einen XYZM Modus. Fehlenden Z- und M-Dimensionen wird eine 0 angeheftet.

Änderung: 2.1.0. Bis zu 2.0.x wurde diese Funktion mit ST_Force_4D bezeichnet.



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



This method supports Circular Strings and Curves

Beispiele

```
--Wenn bereits eine 3D-Geometrie vorliegt, geschieht nichts
SELECT ST_AsEWKT(ST_Force4D(ST_GeomFromEWKT('CIRCULARSTRING(1 1 2, 2 3 2, 4 5 2, 6 7 2, 5 6 2)')));
```

	st_asewkt
	CIRCULARSTRING(1 1 2 0,2 3 2 0,4 5 2 0,6 7 2 0,5 6 2 0)

```
SELECT ST_AsEWKT(ST_Force4D('MULTILINESTRINGM((0 0 1,0 5 2,5 0 3,0 0 4),(1 1 1,3 1 1,1 3 1,1 1 1))'));
```

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

Siehe auch

[ST_AsEWKT](#), [ST_Force2D](#), [ST_Force3DM](#), [ST_Force3D](#)

8.6.8 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.



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



This function supports M coordinates.

Siehe auch

ST_ForcePolygonCW , ST_IsPolygonCCW , ST_IsPolygonCW

8.6.9 ST_ForceCollection

ST_ForceCollection — Wandelt eine Geometrie in eine GEOMETRYCOLLECTION um.

Synopsis

```
geometry ST_ForceCollection(geometry geomA);
```

Beschreibung

Wandelt eine Geometrie in eine GEOMETRYCOLLECTION um. Nützlich um eine WKB-Darstellung zu vereinfachen.

Erweiterung: Mit 2.0.0 wurde die Unterstützung für polyedrische Oberflächen eingeführt.

Verfügbarkeit: 1.2.2, Vor 1.3.4 ist diese Funktion bei CURVES abgestürzt. Dies wurde mit 1.3.4+ behoben

Änderung: 2.1.0. Bis zu 2.0.x wurde diese Funktion mit ST_Force_Collection bezeichnet.



This function supports Polyhedral surfaces.



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



This method supports Circular Strings and Curves

Beispiele

```
SELECT ST_AsEWKT(ST_ForceCollection('POLYGON((0 0 1,0 5 1,5 0 1,0 0 1),(1 1 1,3 1 1,1 3 1,1 1 1))'));
```

```
st asewkt
```

```
GEOMETRYCOLLECTION(POLYGON((0 0 1,0 5 1,5 0 1,0 0 1),(1 1 1,3 1 1,1 3 1,1 1 1)))
```

```
SELECT ST_AsText(ST_ForceCollection('CIRCULARSTRING(220227 150406,220227 150407,220227 150406)'));

```

```
st_astext
```

```
GEOMETRYCOLLECTION(CIRCULARSTRING(220227 150406,220227 150407,220227 150406))
(1 row)
```

```
-- Beispiel für eine polyedrische Oberfläche --
```

```
SELECT ST_AsEWKT(ST_ForceCollection('POLYHEDRALSURFACE(((0 0 0,0 0 1,0 1 1,0 1 0,0 0 0)),
((0 0 0,0 1 0,1 1 0,1 0 0,0 0 0)),
((0 0 0,1 0 0,1 0 1,0 0 1,0 0 0)),
((1 1 0,1 1 1,1 0 1,1 0 0,1 1 0)),
((0 1 0,0 1 1,1 1 1 1,1 1 0,0 1 0)),
((0 0 1,1 0 1,1 1 1,0 1 1,0 0 1)))'))
```

st_asewkt


```

GEOMETRYCOLLECTION (
  POLYGON ((0 0 0,0 0 1,0 1 1,0 1 0,0 0 0)),
  POLYGON ((0 0 0,0 1 0,1 1 0,1 0 0,0 0 0)),
  POLYGON ((0 0 0,1 0 0,1 0 1,0 0 1,0 0 0)),
  POLYGON ((1 1 0,1 1 1,1 0 1,1 0 0,1 1 0)),
  POLYGON ((0 1 0,0 1 1,1 1 1,1 1 0,0 1 0)),
  POLYGON ((0 0 1,1 0 1,1 1 1,0 1 1,0 0 1))
)

```

Siehe auch

[ST_AsEWKT](#), [ST_Force2D](#), [ST_Force3DM](#), [ST_Force3D](#), [ST_GeomFromEWKT](#)

8.6.10 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.



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



This function supports M coordinates.

Siehe auch

[ST_ForcePolygonCCW](#), [ST_IsPolygonCCW](#), [ST_IsPolygonCW](#)

8.6.11 ST_ForceSFS

ST_ForceSFS — Erzwingt, dass Geometrien nur vom Typ SFS 1.1 sind.

Synopsis

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

Beschreibung



This function supports Polyhedral surfaces.



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



This method supports Circular Strings and Curves



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

8.6.12 ST_ForceRHR

ST_ForceRHR — Orientiert die Knoten in einem Polygon so, dass sie der Drei-Finger-Regel folgen.

Synopsis

geometry **ST_ForceRHR**(geometry g);

Beschreibung

Orientiert die Knoten in einem Polygon so, dass sie der Drei-Finger-Regel folgen. Dadurch kommt die durch das Polygon begrenzte Fläche auf der rechten Seite der Begrenzung zu liegen. Insbesondere sind der äussere Ring im Uhrzeigersinn und die inneren Ringe gegen den Uhrzeigersinn orientiert. Diese Funktion ist ein Synonym für **ST_ForcePolygonCW**



Note

Die obere Definition mit der Drei-Finger-Regel widerspricht den Definitionen, die in anderen Zusammenhängen verwendet werden. Um Verwirrung zu vermeiden, wird die Verwendung von ST_ForcePolygonCW empfohlen.

Erweiterung: Mit 2.0.0 wurde die Unterstützung für polyedrische Oberflächen eingeführt.



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



This function supports Polyhedral surfaces.

Beispiele

```
SELECT ST_AsEWKT(
  ST_ForceRHR(
    'POLYGON((0 0 2, 5 0 2, 0 5 2, 0 0 2),(1 1 2, 1 3 2, 3 1 2, 1 1 2))'
  )
);
```

	st_asewkt
	POLYGON((0 0 2,0 5 2,5 0 2,0 0 2),(1 1 2,3 1 2,1 3 2,1 1 2))
(1 row)	

Siehe auch

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

8.6.13 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 CompoundCurves, MultiLines werden MultiCurves, Polygone werden zu CurvePolygons, Multipolygons werden MultiSurfaces. Wenn die Geometrie bereits in Kurvendarstellung vorliegt, wird sie unverändert zurückgegeben.

Verfügbarkeit: 2.2.0



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



This method supports Circular Strings and Curves

Beispiele

```
SELECT ST_AsText (
  ST_ForceCurve (
    'POLYGON((0 0 2, 5 0 2, 0 5 2, 0 0 2),(1 1 2, 1 3 2, 3 1 2, 1 1 2))'::geometry
  )
);
```

	st_astext
	CURVEPOLYGON Z ((0 0 2,5 0 2,0 5 2,0 0 2),(1 1 2,1 3 2,3 1 2,1 1 2))
(1 row)	

Siehe auch

[ST_LineToCurve](#)

8.6.14 ST_LineMerge

ST_LineMerge — Gibt einen (Satz von) LineString(s) zurück, der aus einem MultiLineString "zusammengebastelt" wird.

Synopsis

geometry **ST_LineMerge**(geometry amultilinestring);

Beschreibung

Gibt einen (Satz von) LineString(s) zurück, der aus den Bestandteilen eines MultiLineString zusammengesetzt wird.



Note

Ist nur mit MULTILINESTRING/LINESTRING verwendbar. Wenn Sie ein Polygon oder eine Sammelgeometrie in diese Funktion einspeisen, wird eine leere GEOMETRYCOLLECTION zurückgegeben

Verfügbarkeit: 1.1.0

**Note**

benötigt GEOS >= 2.1.0

Beispiele

```
SELECT ST_AsText(ST_LineMerge(
ST_GeomFromText('MULTILINESTRING((-29 -27,-30 -29.7,-36 -31,-45 -33), (-45 -33,-46 -32))'))
);
st_astext
-----
LINESTRING(-29 -27,-30 -29.7,-36 -31,-45 -33,-46 -32)
(1 row)

--Wenn ein Verschmelzen nicht möglich ist, wird der ursprüngliche MULTILINESTRING ↵
    zurückgegeben
SELECT ST_AsText(ST_LineMerge(
ST_GeomFromText('MULTILINESTRING((-29 -27,-30 -29.7,-36 -31,-45 -33), (-45.2 -33.2,-46 -32))')
);
st_astext
-----
MULTILINESTRING((-45.2 -33.2,-46 -32), (-29 -27,-30 -29.7,-36 -31,-45 -33))
```

Siehe auch[ST_Segmentize](#), [ST_LineSubstring](#)**8.6.15 ST_CollectionExtract**

ST_CollectionExtract — Von einer gegebenen (Sammel)Geometrie wird eine (Sammel)Geometrie zurückgegeben, welche nur die Elemente des vorgegebenen Datentyps enthält.

Synopsis

geometry **ST_CollectionExtract**(geometry collection, integer type);

Beschreibung

Von einer (Sammel)Geometrie wird eine (Mehrfach)Geometrie zurückgegeben, welche nur die Elemente des vorgegebenen Datentyps enthält. Teilgeometrien, die nicht dem vorgegebenen Datentyp entsprechen, werden ausgelassen. Wenn keine der Teilgeometrien den richtigen Datentyp aufweist, wird eine LEERE Geometrie zurückgegeben. Es werden nur Punkte, Linien und Polygone unterstützt. Die Kennungen sind 1 == POINT, 2 == LINESTRING, 3 == POLYGON.

Verfügbarkeit: 1.5.0

**Note**

Vor 1.5.3 gab diese Funktion, wenn die Eingabe keine Sammelgeometrie war, diese unabhängig vom Datentyp unangestastet zurück. Bei 1.5.3 wurde bei unpassenden Einzelgeometrien NULL zurückgegeben. Ab 2.0.0 wird bei fehlender Übereinstimmung immer eine LEERE Ausgabe zurückgegeben.

**Warning**

Wenn 3 == POLYGON angegeben ist, wird ein Mehrfachpolygon zurückgegeben, sogar dann wenn die Kanten gemeinsam genutzt werden. Dies endet in vielen Fällen einem invaliden Mehrfachpolygon, zum Beispiel wenn diese Funktion auf ein [ST_Split](#) Ergebnis angewendet wird.

Beispiele

```
-- Konstante: 1 == POINT, 2 == LINESTRING, 3 == POLYGON
SELECT ST_AsText(ST_CollectionExtract(ST_GeomFromText('GEOMETRYCOLLECTION( ↵
    GEOMETRYCOLLECTION(POINT(0 0))'),1));
st_astext
-----
MULTIPOINT(0 0)
(1 row)

SELECT ST_AsText(ST_CollectionExtract(ST_GeomFromText('GEOMETRYCOLLECTION( ↵
    GEOMETRYCOLLECTION(LINESTRING(0 0, 1 1)),LINESTRING(2 2, 3 3))'),2));
st_astext
-----
MULTILINESTRING((0 0, 1 1), (2 2, 3 3))
(1 row)
```

Siehe auch

[ST_Multi](#), [ST_Dump](#), [ST_CollectionHomogenize](#)

8.6.16 ST_CollectionHomogenize

ST_CollectionHomogenize — Von einer gegebenen Sammelgeometrie wird die "einfachste" Darstellung der Inhalte zurückgegeben.

Synopsis

```
geometry ST_CollectionHomogenize(geometry collection);
```

Beschreibung

Von einer Sammelgeometrie wird die "einfachste" Darstellung der Inhalte zurückgegeben. Einelementige Geometrien werden ebenso zurückgegeben. Einheitliche Sammelgeometrien werden, entsprechend dem Datentyp, als Mehrfachgeometrien ausgegeben.

**Warning**

Wenn 3 == POLYGON angegeben ist, wird ein Mehrfachpolygon zurückgegeben, sogar dann wenn die Kanten gemeinsam genutzt werden. Dies endet in vielen Fällen einem invaliden Mehrfachpolygon, zum Beispiel wenn diese Funktion auf ein [ST_Split](#) Ergebnis angewendet wird.

Verfügbarkeit: 2.0.0

Beispiele

```
SELECT ST_AsText(ST_CollectionHomogenize('GEOMETRYCOLLECTION(POINT(0 0))'));

 st_astext
-----
POINT(0 0)
(1 row)

SELECT ST_AsText(ST_CollectionHomogenize('GEOMETRYCOLLECTION(POINT(0 0),POINT(1 1))'));

 st_astext
-----
MULTIPOINT(0 0,1 1)
(1 row)
```

Siehe auch

[ST_Multi](#), [ST_CollectionExtract](#)

8.6.17 ST_Multi

ST_Multi — Gibt die Geometrie als MULTI* Geometrie zurück.

Synopsis

geometry **ST_Multi**(geometry g1);

Beschreibung

Gibt die Geometrie als MULTI* Geometrie zurück. Falls es sich bereits um eine MULTI* Geometrie handelt, bleibt diese unverändert.

Beispiele

```
SELECT ST_AsText(ST_Multi(ST_GeomFromText('POLYGON((743238 2967416,743238 2967450,
743265 2967450,743265.625 2967416,743238 2967416))')));
 st_astext
-----
MULTIPOLYGON(((743238 2967416,743238 2967450,743265  ↵
2967450,743265.625 2967416,
743238 2967416)))
(1 row)
```

Siehe auch

[ST_AsText](#)

8.6.18 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).

Beispiele

```
SELECT ST_AsText(ST_Normalize(ST_GeomFromText(
  'GEOMETRYCOLLECTION(
    POINT(2 3),
    MULTILINESTRING((0 0, 1 1),(2 2, 3 3)),
    POLYGON(
      (0 10,0 0,10 0,10 10,0 10),
      (4 2,2 2,2 4,4 4,4 2),
      (6 8,8 8,8 6,6 6,6 8)
    )
  ) '
))) ;
```

st_astext

```
GEOMETRYCOLLECTION(POLYGON((0 0,0 10,10 10,10 0,0 0),(6 6,8 6,8 8,6 8,6 6),(2 2,4 2,4 4,2 4,2 2)),MULTILINESTRING((2 2,3 3),(0 0,1 1)),POINT(2 3))
(1 row)
```

Siehe auch

[ST_Equals](#),

8.6.19 ST_RemovePoint

ST_RemovePoint — Entfernt einen Punkt aus einem Linienzug.

Synopsis

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

Beschreibung

Entfernt einen Punkt aus einem LineString; der Index beginnt mit 0. Nützlich, um einen geschlossenen Ring in einen offenen Linienzug umzuwandeln

Verfügbarkeit: 1.1.0



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

Beispiele

```
--stellt sicher, dass keine LINESTRINGS geschlossen sind,
--indem der Endpunkt entfernt wird. Unten wird angenommen, dass the_geom vom Datentyp  $\leftrightarrow$ 
  LINESTRING ist
UPDATE sometable
   SET the_geom = ST_RemovePoint(the_geom, ST_NPoints(the_geom) - 1)
   FROM sometable
  WHERE ST_IsClosed(the_geom) = true;
```

Siehe auch

[ST_AddPoint](#), [ST_NPoints](#), [ST_NumPoints](#)

8.6.20 ST_Reverse

ST_Reverse — Gibt die Geometrie in umgekehrter Knotenreihenfolge zurück.

Synopsis

geometry **ST_Reverse**(geometry g1);

Beschreibung

Kann mit jedem geometrischen Datentyp verwendet werden; kehrt die Reihenfolge der Knoten um

Erweiterung: mit 2.4.0 wurde die Unterstützung für Kurven eingeführt.



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



This function supports Polyhedral surfaces.

Beispiele

```
SELECT ST_AsText(the_geom) as line, ST_AsText(ST_Reverse(the_geom)) As reverseline
FROM
  (SELECT ST_MakeLine(ST_MakePoint(1,2),
                     ST_MakePoint(1,10)) As the_geom) as foo;
--result
-----+-----
line          | reverseline
-----+-----
LINESTRING(1 2,1 10) | LINESTRING(1 10,1 2)
```

8.6.21 ST_Rotate

ST_Rotate — Dreht die Geometrie um rotRadians gegen den Uhrzeigersinn um ein Zentrum.

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 um `rotRadians` um ein Zentrum und zwar gegen den Uhrzeigersinn. Das Drehzentrum kann entweder durch eine Punktgeometrie oder über X- und Y-Koordinaten festgelegt werden. Wenn das Drehzentrum nicht angegeben ist, wird die Geometrie um `POINT(0 0)` gedreht.

Erweiterung: Mit 2.0.0 wurde die Unterstützung für polyedrische Oberflächen, Dreiecke und TIN eingeführt.

Erweiterung: 2.0.0 zusätzliche Parameter zur Festlegung des Drehzentrums.

Verfügbarkeit: 1.1.2. Mit 1.2.2 wurde die Bezeichnung von `Rotate` auf `ST_Rotate` geändert.



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



This method supports Circular Strings and Curves



This function supports Polyhedral surfaces.



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

Beispiele

```
--Um 180 Grad drehen
SELECT ST_AsEWKT(ST_Rotate('LINESTRING (50 160, 50 50, 100 50)', pi()));
           st_asewkt
-----
LINESTRING(-50 -160,-50 -50,-100 -50)
(1 row)

--Von x=50, y=160 aus, gegen den Uhrzeigersinn um 30 Grad drehen
SELECT ST_AsEWKT(ST_Rotate('LINESTRING (50 160, 50 50, 100 50)', pi()/6, 50, 160));
           st_asewkt
-----
LINESTRING(50 160,105 64.7372055837117,148.301270189222 89.7372055837117)
(1 row)

--Vom geometrischen Schwerpunkt aus, im Uhrzeigersinn um 60 Grad drehen
SELECT ST_AsEWKT(ST_Rotate(geom, -pi()/3, ST_Centroid(geom)))
FROM (SELECT 'LINESTRING (50 160, 50 50, 100 50)'::geometry AS geom) AS foo;
           st_asewkt
-----
LINESTRING(116.4225 130.6721,21.1597 75.6721,46.1597 32.3708)
(1 row)
```

Siehe auch

[ST_Affine](#), [ST_RotateX](#), [ST_RotateY](#), [ST_RotateZ](#)

8.6.22 ST_RotateX

`ST_RotateX` — Dreht eine Geometrie um `rotRadians` um die X-Achse.

Synopsis

geometry **ST_RotateX**(geometry geomA, float rotRadians);

Beschreibung

Dreht die Geometrie "geomA" - rotRadians um die X-Achse.



Note

`ST_RotateX(geomA, rotRadians)` is short-hand for `ST_Affine(geomA, 1, 0, 0, 0, cos(rotRadians), -sin(rotRadians), 0, sin(rotRadians), cos(rotRadians), 0, 0, 0)`.

Erweiterung: Mit 2.0.0 wurde die Unterstützung für polyedrische Oberflächen, Dreiecke und TIN eingeführt.

Verfügbarkeit: 1.1.2. Mit 1.2.2 wurde die Bezeichnung von RotateX auf ST_RotateX geändert.



This function supports Polyhedral surfaces.



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



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

Beispiele

```
--Dreht eine Linie um 90 Grad entlang der X-Achse
SELECT ST_AsEWKT(ST_RotateX(ST_GeomFromEWKT('LINESTRING(1 2 3, 1 1 1)'), pi()/2));
           st_asewkt
-----
LINESTRING(1 -3 2,1 -1 1)
```

Siehe auch

[ST_Affine](#), [ST_RotateY](#), [ST_RotateZ](#)

8.6.23 ST_RotateY

`ST_RotateY` — Dreht eine Geometrie um rotRadians um die Y-Achse.

Synopsis

geometry **ST_RotateY**(geometry geomA, float rotRadians);

Beschreibung

Dreht die Geometrie "geomA" - rotRadians um die Y-Achse.



Note

`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)`.

Verfügbarkeit: 1.1.2. Mit 1.2.2 wurde die Bezeichnung von RotateY auf ST_RotateY geändert.

Erweiterung: Mit 2.0.0 wurde die Unterstützung für polyedrische Oberflächen, Dreiecke und TIN eingeführt.



This function supports Polyhedral surfaces.



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



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

Beispiele

```
--Dreht eine Linie um 90 Grad entlang der Y-Achse
SELECT ST_AsEWKT(ST_RotateY(ST_GeomFromEWKT('LINESTRING(1 2 3, 1 1 1)'), pi()/2));
          st_asewkt
-----
LINESTRING(3 2 -1,1 1 -1)
```

Siehe auch

[ST_Affine](#), [ST_RotateX](#), [ST_RotateZ](#)

8.6.24 ST_RotateZ

ST_RotateZ — Dreht eine Geometrie um rotRadians um die Z-Achse.

Synopsis

geometry **ST_RotateZ**(geometry geomA, float rotRadians);

Beschreibung

Dreht die Geometrie "geomA" - rotRadians um die Z-Achse.



Note

Dies ist ein Synonym für ST_Rotate



Note

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).

Erweiterung: Mit 2.0.0 wurde die Unterstützung für polyedrische Oberflächen, Dreiecke und TIN eingeführt.

Verfügbarkeit: 1.1.2. Mit 1.2.2 wurde die Bezeichnung von RotateZ auf ST_RotateZ geändert.

**Note**

Vor 1.3.4 ist diese Funktion abgestürzt, wenn die Geometrien CURVES enthalten. Dies wurde mit 1.3.4+ behoben



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



This method supports Circular Strings and Curves



This function supports Polyhedral surfaces.



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

Beispiele

```
--Eine Linie um 90 Grad um die Z-Achse drehen
SELECT ST_AseWKT(ST_RotateZ(ST_GeomFromEWKT('LINESTRING(1 2 3, 1 1 1)'), pi()/2));
           st_asewkt
-----
LINESTRING(-2 1 3,-1 1 1)

--Einen Kreisbogen um die Z-Achse drehen
SELECT ST_AseWKT(ST_RotateZ(the_geom, pi()/2))
FROM (SELECT ST_LineToCurve(ST_Buffer(ST_GeomFromText('POINT(234 567)'), 3)) As the_geom) ←
      As foo;

-----

CURVEPOLYGON(CIRCULARSTRING(-567 237,-564.87867965644 236.12132034356,-564 ←
234,-569.12132034356 231.87867965644,-567 237))
```

Siehe auch

[ST_Affine](#), [ST_RotateX](#), [ST_RotateY](#)

8.6.25 ST_Scale

ST_Scale — Skaliert eine Geometrie anhand der gegebenen 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);
```

Beschreibung

Skaliert eine Geometrie auf eine neue Größe, indem die Ordinaten mit den entsprechenden Faktoren multipliziert werden.

Jene Version, die Geometrie als den Parameter `factor` entgegennimmt, erlaubt die Eingabe eines 2D-, 3DM, 3DZ oder 4D-Punktes zum Setzen des Skalierungsfaktor in den unterstützten Dimensionen. Fehlt eine Dimension im Punkt `factor`, so wird diese Dimension nicht skaliert.



Note

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.

Erweiterung: 2.2.0 Skalierbarkeit in allen Dimensionen (geometrischer Parameter) eingeführt.



This function supports Polyhedral surfaces.



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



This method supports Circular Strings and Curves



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



This function supports M coordinates.

Beispiele

```
--Version 1: Skalierung von X, Y, Z
SELECT ST_AsEWKT(ST_Scale(ST_GeomFromEWKT('LINESTRING(1 2 3, 1 1 1)'), 0.5, 0.75, 0.8));
           st_asewkt
-----
LINESTRING(0.5 1.5 2.4,0.5 0.75 0.8)

--Version 2: Skalierung von X, Y
SELECT ST_AsEWKT(ST_Scale(ST_GeomFromEWKT('LINESTRING(1 2 3, 1 1 1)'), 0.5, 0.75));
           st_asewkt
-----
LINESTRING(0.5 1.5 3,0.5 0.75 1)

--Version 3: Skalierung von X, Y, Z, M
SELECT ST_AsEWKT(ST_Scale(ST_GeomFromEWKT('LINESTRING(1 2 3 4, 1 1 1 1)'),
  ST_MakePoint(0.5, 0.75, 2, -1)));
           st_asewkt
-----
LINESTRING(0.5 1.5 6 -4,0.5 0.75 2 -1)
```

Siehe auch

[ST_Affine](#), [ST_TransScale](#)

8.6.26 ST_Segmentize

ST_Segmentize — Gibt eine veränderte Geometrie/Geographie zurück, bei der kein Segment länger als der gegebene Abstand ist.

Synopsis

```
geometry ST_Segmentize(geometry geom, float max_segment_length);
geography ST_Segmentize(geography geog, float max_segment_length);
```

Beschreibung

Gibt eine veränderte Geometrie/Geographie zurück, bei der kein Segment länger als die gegebene `max_segment_length` ist. Die Entfernungsberechnung wird nur in 2D ausgeführt. Beim geometrischen Datentyp ist die Längeneinheit die Einheit des Koordinatenreferenzsystems. Beim geographischen Datentyp ist die Einheit Meter.

Verfügbarkeit: 1.2.2

Erweiterung: 2.3.0 - Das Segmentieren des geographischen Datentyps ergibt nun Segmente gleicher Länge

Erweiterung: mit 2.1.0 wurde die Unterstützung des geographischen Datentyps eingeführt.

Änderung: 2.1.0 Als Ergebnis der eingeführten Unterstützung für den geographischen Datentyp: Das Konstrukt `SELECT ST_Segmentize('LINESTRING(1 2, 3 4)', 0.5);` resultiert in einen Funktionsfehler aufgrund von Mehrdeutigkeit. Sie benötigen korrekt typisierte Geoobjekte; Verwenden Sie z.B. `ST_GeomFromText`, `ST_GeogFromText` oder `SELECT ST_Segmentize('LINESTRING(1 2, 3 4)'::geometry, 0.5);` für Ihre Geometrie-/Geographiespalte.



Note

Segmente werden lediglich verlängert. Die Länge von Segmenten, die kürzer als `max_segment_length` sind, wird nicht verändert.

Beispiele

```
SELECT ST_AsText(ST_Segmentize(
  ST_GeomFromText('MULTILINESTRING((-29 -27,-30 -29.7,-36 -31,-45 -33), (-45 -33,-46 -32))')
  , 5)
);
st_astext
```

```
MULTILINESTRING((-29 -27,-30 -29.7,-34.886615700134 -30.758766735029,-36 -31,
-40.8809353009198 -32.0846522890933,-45 -33),
(-45 -33,-46 -32))
(1 row)
```

```
SELECT ST_AsText(ST_Segmentize(ST_GeomFromText('POLYGON((-29 28, -30 40, -29 28))'), 10));
st_astext
```

```
POLYGON((-29 28,-29.8304547985374 37.9654575824488,-30 40,-29.1695452014626
30.0345424175512,-29 28))
(1 row)
```

Siehe auch

[ST_LineSubstring](#)

8.6.27 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, sodass -1 der letzte Punkt ist. Dies findet insbesondere bei Triggern Verwendung, wenn man die Beziehung zwischen den Verbindungsstücken beim Verschieben von Knoten erhalten will

Verfügbarkeit: 1.1.0

Änderung: 2.3.0 : negatives Indizieren



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

Beispiele

```
--Ändert den ersten Punkt eines Linienzuges von -1 3 auf -1 1
SELECT ST_AsText(ST_SetPoint('LINESTRING(-1 2,-1 3)', 0, 'POINT(-1 1)'));
          st_astext
-----
LINESTRING(-1 1,-1 3)

---Ändert den Endpunkt eines Linienzuges (diesmal ein 3D-Linienzug)
SELECT ST_AsEWKT(ST_SetPoint(foo.the_geom, ST_NumPoints(foo.the_geom) - 1, ST_GeomFromEWKT ←
('POINT(-1 1 3)'))
FROM (SELECT ST_GeomFromEWKT('LINESTRING(-1 2 3,-1 3 4, 5 6 7)') As the_geom) As foo;
          st_asewkt
-----
LINESTRING(-1 2 3,-1 3 4,-1 1 3)

SELECT ST_AsText(ST_SetPoint(g, -3, p))
FROM ST_GeomFromText('LINESTRING(0 0, 1 1, 2 2, 3 3, 4 4)') AS g
      , ST_PointN(g,1) as p;
          st_astext
-----
LINESTRING(0 0,1 1,0 0,3 3,4 4)
```

Siehe auch

[ST_AddPoint](#), [ST_NPoints](#), [ST_NumPoints](#), [ST_PointN](#), [ST_RemovePoint](#)

8.6.28 ST_SetSRID

ST_SetSRID — Weist der SRID einer Geometrie einen bestimmten Ganzzahlwert zu.

Synopsis

geometry **ST_SetSRID**(geometry geom, integer srid);

Beschreibung

Weist der SRID einer Geometrie einen bestimmten Ganzzahlwert zu. Nützlich um Umgebungsrechtecke für Abfragen zu erzeugen.



Note

Diese Funktion führt in keiner Weise eine Koordinatentransformation durch - sie setzt nur die Metadaten, welche das Koordinatenreferenzsystem definieren, in dem die Geometrie vorliegt. Verwenden Sie bitte **ST_Transform**, falls Sie die Geometrie in ein neues Koordinatensystem überführen wollen.



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



This method supports Circular Strings and Curves

Beispiele

-- Einem Punkt das geodätische Referenzsystem "WGS 84" zuweisen --

```
SELECT ST_SetSRID(ST_Point(-123.365556, 48.428611),4326) As wgs84long_lat;
-- EWKT-Darstellung (mit ST_AsEWKT umschließen) -
SRID=4326;POINT(-123.365556 48.428611)
```

-- Einem Punkt das Referenzsystem WGS 84 zuweisen und nach Web Mercator (Spherical Mercator) transformieren --

```
SELECT ST_Transform(ST_SetSRID(ST_Point(-123.365556, 48.428611),4326),3785) As spere_merc;
-- EWKT-Darstellung (mit ST_AsEWKT umschließen) -
SRID=3785;POINT(-13732990.8753491 6178458.96425423)
```

Siehe auch

Section [4.3.1](#), [ST_AsEWKT](#), [ST_Point](#), [ST_SRID](#), [ST_Transform](#), [UpdateGeometrySRID](#)

8.6.29 ST_SnapToGrid

ST_SnapToGrid — Fängt alle Punkte der Eingabegeometrie auf einem regelmäßigen Gitter.

Synopsis

```
geometry ST_SnapToGrid(geometry geomA, float originX, float originY, float sizeX, float sizeY);
geometry ST_SnapToGrid(geometry geomA, float sizeX, float sizeY);
geometry ST_SnapToGrid(geometry geomA, float size);
geometry ST_SnapToGrid(geometry geomA, geometry pointOrigin, float sizeX, float sizeY, float sizeZ, float sizeM);
```

Beschreibung

Variante 1, 2 und 3: Fängt alle Punkte der Eingabegeometrie auf den Gitterpunkten, die durch Ursprung und Gitterkästchengröße festgelegt sind. Aufeinanderfolgende Punkte, die in dasselbe Gitterkästchen fallen, werden gelöscht, wobei NULL zurückgegeben wird, wenn nicht mehr genug Punkte für den jeweiligen geometrischen Datentyp vorhanden sind. Collapsed geometries in a collection are stripped from it. Kollabierte Geometrien einer Kollektion werden von dieser entfernt. Nützlich um die Genauigkeit zu verringern.

Variante 4: wurde mit 1.1.0 eingeführt - Fängt alle Punkte der Eingabegeometrie auf den Gitterpunkten, welche durch den Ursprung des Gitters (der zweite Übergabewert muss ein Punkt sein) und die Gitterkästchengröße bestimmt sind. Geben Sie 0 als Größe für jene Dimension an, die nicht auf den Gitterpunkten gefangen werden soll.

**Note**

Die zurückgegebene Geometrie kann ihre Simplizität verlieren (siehe [ST_IsSimple](#)).

**Note**

Vor Release 1.1.0 gab diese Funktion immer eine 2D-Geometrie zurück. Ab 1.1.0 hat die zurückgegebene Geometrie dieselbe Dimensionalität wie die Eingabegometrie, wobei höhere Dimensionen unangetastet bleiben. Verwenden Sie die Version, welche einen zweiten geometrischen Übergabewert annimmt, um sämtliche Grid-Dimensionen zu bestimmen.

Verfügbarkeit: 1.0.0RC1

Verfügbarkeit: 1.1.0, Unterstützung für Z und M



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

Beispiele

```
--Fängt die Geometrien an einem Gitter mit einer Genauigkeit von 10^-3
UPDATE mytable
  SET the_geom = ST_SnapToGrid(the_geom, 0.001);

SELECT ST_AsText(ST_SnapToGrid(
    ST_GeomFromText('LINESTRING(1.1115678 2.123, 4.111111 3.2374897, 4.11112 3.23748667)'),
    0.001)
    ) AS st_astext
-----
LINESTRING(1.112 2.123,4.111 3.237)
--Fängt eine 4D-Geometrie
SELECT ST_AsEWKT(ST_SnapToGrid(
    ST_GeomFromEWKT('LINESTRING(-1.1115678 2.123 2.3456 1.11111, 4.111111 3.2374897 3.1234 1.1111, -1.1111112 2.123 2.3456 1.111112)'),
    ST_GeomFromEWKT('POINT(1.12 2.22 3.2 4.4444)'),
    0.1, 0.1, 0.1, 0.01) ) AS st_asewkt
-----
LINESTRING(-1.08 2.12 2.3 1.1144,4.12 3.22 3.1 1.1144,-1.08 2.12 2.3 1.1144)

--Bei einer 4D-Geometrie - ST_SnapToGrid(geom,size) behandelt nur die X- und Y-Koordinaten
--und belässt die M- und Z-Koordinaten
SELECT ST_AsEWKT(ST_SnapToGrid(ST_GeomFromEWKT('LINESTRING(-1.1115678 2.123 3 2.3456, 4.111111 3.2374897 3.1234 1.1111)'),
    0.01) ) AS st_asewkt
-----
LINESTRING(-1.11 2.12 3 2.3456,4.11 3.24 3.1234 1.1111)
```

Siehe auch

[ST_Snap](#), [ST_AsEWKT](#), [ST_AsText](#), [ST_GeomFromText](#), [ST_GeomFromEWKT](#), [ST_Simplify](#)

8.6.30 ST_Snap

ST_Snap — Fängt die Segmente und Knoten einer Eingabegeometrie an den Knoten einer Referenzgeometrie.

Synopsis

geometry **ST_Snap**(geometry input, geometry reference, float tolerance);

Beschreibung

Fängt die Knoten und Segmente einer Geometrie an den Knoten einer anderen Geometrie. Eine Entfernungstoleranz bestimmt, wo das Fangen durchgeführt wird. Die Ergebnisgeometrie ist die Eingabegeometrie mit gefangenen Knoten. Wenn kein Fangen auftritt, wird die Eingabegeometrie unverändert ausgegeben..

Eine Geometrie an einer anderen zu fangen, kann die Robustheit von Überlagerungs-Operationen verbessern, indem nahe zusammenfallende Kanten beseitigt werden (diese verursachen Probleme bei der Knoten- und Verschneidungsberechnung).

Übermäßiges Fangen kann zu einer invaliden Topologie führen. Die Anzahl und der Ort an dem Knoten sicher gefangen werden können wird mittels Heuristik bestimmt. Dies kann allerdings dazu führen, dass einige potentielle Knoten nicht gefangen werden.

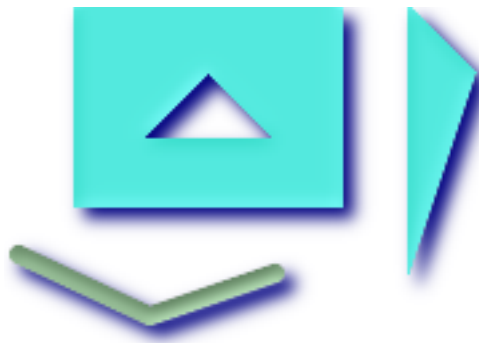


Note

Die zurückgegebene Geometrie kann ihre Simplität (see [ST_IsSimple](#)) und Validität (see [ST_IsValid](#)) verlieren.

Verfügbarkeit: 2.0.0 benötigt GEOS >= 3.3.0.

Beispiele



Ein Mehrfachpolygon mit einem Linienzug (vor dem Fangen)



Ein Mehrfachpolygon das an einem Linienzug gefangen wird; die Toleranz beträgt 1.01 der Entfernung. Das neue Mehrfachpolygon wird mit dem betreffenden Linienzug angezeigt.

```
SELECT ST_AsText(ST_Snap(poly,line, ↵
    ST_Distance(poly,line)*1.01)) AS polysnapped ↵
FROM (SELECT ↵
    ST_GeomFromText('MULTIPOLYGON( ↵
        ((26 125, 26 200, 126 200, 126 125, ↵
        26 125 ), ↵
        ( 51 150, 101 150, 76 175, 51 150 ) ↵
        ), ↵
        (( 151 100, 151 200, 176 175, 151 ↵
        100 )))') As poly, ↵
    ST_GeomFromText('LINESTRING (5 ↵
        107, 54 84, 101 100)') As line ↵
    ) As foo; ↵
                                polysnapped
```

```
MULTIPOLYGON(((26 125,26 200,126 200,126 ↵
    125,101 100,26 125), ↵
    (51 150,101 150,76 175,51 150)),((151 ↵
    100,151 200,176 175,151 100)))
```



Ein Mehrfachpolygon das an einem Linienzug gefangen wird; die Toleranz beträgt 1.25 der Entfernung. Das neue Mehrfachpolygon wird mit dem betreffenden Linienzug angezeigt.

```
SELECT ST_AsText ( ↵
    ST_Snap(poly,line, ST_Distance(poly, ↵
    line)*1.25) ↵
    ) AS polysnapped ↵
FROM (SELECT ↵
    ST_GeomFromText('MULTIPOLYGON( ↵
        (( 26 125, 26 200, 126 200, 126 125, ↵
        26 125 ), ↵
        ( 51 150, 101 150, 76 175, 51 150 ) ↵
        ), ↵
        (( 151 100, 151 200, 176 175, 151 ↵
        100 )))') As poly, ↵
    ST_GeomFromText('LINESTRING (5 ↵
        107, 54 84, 101 100)') As line ↵
    ) As foo; ↵
                                ↵ polysnapped
```

```
MULTIPOLYGON(((5 107,26 200,126 200,126 ↵
    125,101 100,54 84,5 107), ↵
    (51 150,101 150,76 175,51 150)),((151 ↵
    100,151 200,176 175,151 100)))
```



Ein Linienzug der an dem ursprünglichen Mehrfachpolygon gefangen wird; die Toleranz beträgt 1.01 der Entfernung. Das neue Linienzug wird mit dem betreffenden Mehrfachpolygon angezeigt.

```
SELECT ST_AsText (
  ST_Snap(line, poly, ST_Distance(poly, ↵
    line)*1.01)
) AS linesnapped
FROM (SELECT
  ST_GeomFromText('MULTIPOLYGON(
    ((26 125, 26 200, 126 200, 126 125, ↵
    26 125),
    (51 150, 101 150, 76 175, 51 150 )) ↵
  ',
    ((151 100, 151 200, 176 175, 151 ↵
    100)))') As poly,
    ST_GeomFromText('LINESTRING (5 ↵
    107, 54 84, 101 100)') As line
  ) As foo;

          linesnapped
-----
LINESTRING(5 107,26 125,54 84,101 100)
```



Ein Linienzug der an dem ursprünglichen Mehrfachpolygon gefangen wird; die Toleranz beträgt 1.25 der Entfernung. Das neue Linienzug wird mit dem betreffenden Mehrfachpolygon angezeigt.

```
SELECT ST_AsText (
  ST_Snap(line, poly, ST_Distance(poly, ↵
    line)*1.25)
) AS linesnapped
FROM (SELECT
  ST_GeomFromText('MULTIPOLYGON(
    (( 26 125, 26 200, 126 200, 126 125, ↵
    26 125 ),
    (51 150, 101 150, 76 175, 51 150 )) ↵
  ',
    ((151 100, 151 200, 176 175, 151 ↵
    100 ))') As poly,
    ST_GeomFromText('LINESTRING (5 ↵
    107, 54 84, 101 100)') As line
  ) As foo;

          linesnapped
-----
LINESTRING(26 125,54 84,101 100)
```

Siehe auch

[ST_SnapToGrid](#)

8.6.31 ST_Transform

ST_Transform — Gibt eine neue Geometrie zurück, wobei eine Koordinatentransformation in ein anderes räumliches Bezugssystem durchgeführt wird.

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, wobei eine Koordinatentransformation in ein anderes Koordinatenreferenzsystem durchgeführt wird. Das Zielkoordinatensystem `to_srid` kann durch einen gültigen, ganzzahligen SRID-Parameter (insbesondere muss es in der Tabelle `spatial_ref_sys` existieren) angegeben werden. Alternativ kann das Koordinatenreferenzsystem als PROJ.4 Zeichenkette definiert, für `to_proj` und/oder `from_proj` benutzt werden; Allerdings sind diese Methoden nicht optimiert. Falls das Koordinatenreferenzsystem mit einer PROJ.4 Zeichenkette anstatt über die SRID ausgedrückt ist, so wird die SRID der Ausgabegeometrie auf null gesetzt. Ausgenommen der Funktionen mit `from_proj` benötigt die Eingabegeometrie eine definierte SRID.

`ST_Transform` wird oft mit `ST_SetSRID()` verwechselt. `ST_Transform` ändert die Koordinaten der Geometrie von einem Koordinatenreferenzsystem auf ein anderes, während `ST_SetSRID()` nur den Identifikator "SRID" ändert.



Note

Setzt voraus, dass PostGIS mit Proj-Unterstützung kompiliert wurde. Verwenden Sie bitte [PostGIS_Full_Version](#), um festzustellen ob mit proj kompiliert wurde.



Note

Falls mehrere Koordinatentransformationssysteme verwendet werden, ist es sinnvoll funktionale Indizes auf die üblichen Transformationen zu legen, um die Vorteile der Indexverwendung auszunutzen.



Note

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.

Erweiterung: 2.3.0 - die Unterstützung von direktem PROJ.4 Text wurde eingeführt



This method implements the SQL/MM specification. SQL-MM 3: 5.1.6



This method supports Circular Strings and Curves



This function supports Polyhedral surfaces.

Beispiele

Transformation einer "Massachusetts state plane US feet" Geometrie nach WGS 84 Länge und Breite

```
SELECT ST_AsText(ST_Transform(ST_GeomFromText('POLYGON((743238 2967416,743238 2967450,
      743265 2967450,743265.625 2967416,743238 2967416))',2249),4326)) As wgs_geom;

wgs_geom
```

```

-----
POLYGON((-71.1776848522251 42.3902896512902,-71.1776843766326 42.3903829478009,
-71.1775844305465 42.3903826677917,-71.1775825927231 42.3902893647987,-71.177684
8522251 42.3902896512902));
(1 row)

--Beispiel für einen 3D Kreisbogen
SELECT ST_AsEWKT(ST_Transform(ST_GeomFromEWKT('SRID=2249;CIRCULARSTRING(743238 2967416 ↵
1,743238 2967450 2,743265 2967450 3,743265.625 2967416 3,743238 2967416 4)'),4326));

                                st_asewkt
-----
SRID=4326;CIRCULARSTRING(-71.1776848522251 42.3902896512902 1,-71.1776843766326 ↵
42.3903829478009 2,
-71.1775844305465 42.3903826677917 3,
-71.1775825927231 42.3902893647987 3,-71.1776848522251 42.3902896512902 4)

```

Ein Beispiel für die Erzeugung eines funktionalen, partiellen Index. Für Tabellen, bei denen nicht alle Geometrien ausgefüllt sind, verwendet man am Besten einen partiellen Index, welcher die NULL-Geometrien auslässt. Dies spart Speicherplatz und macht den Index kleiner und effizienter.

```

CREATE INDEX idx_the_geom_26986_parcel
ON parcels
USING gist
(ST_Transform(the_geom, 26986))
WHERE the_geom IS NOT NULL;

```

Beispiele zur Benutzung des PROJ.4 Textes, um in benutzerdefinierte Koordinatenreferenzsysteme zu transformieren.

```

-- Die Verschneidung zweier Polygone in der Nähe des Nordpols mit einer Projektion eruieren
-- See http://boundlessgeo.com/2012/02/flattening-the-peel/
WITH data AS (
  SELECT
    ST_GeomFromText('POLYGON((170 50,170 72,-130 72,-130 50,170 50))', 4326) AS p1,
    ST_GeomFromText('POLYGON((-170 68,-170 90,-141 90,-141 68,-170 68))', 4326) AS p2,
    '+proj=gnom +ellps=WGS84 +lat_0=70 +lon_0=-160 +no_defs'::text AS gnom
)
SELECT ST_AsText(
  ST_Transform(
    ST_Intersection(ST_Transform(p1, gnom), ST_Transform(p2, gnom)),
    gnom, 4326))
FROM data;

                                st_astext
-----
POLYGON((-170 74.053793645338,-141 73.4268621378904,-141 68,-170 68,-170 74.053793645338) ↵
)

```

Konfiguration des Transformationsverhalten

Manchmal kann eine Koordinatentransformation versagen, wenn diese eine Gradnetzverschiebung/grid-shift mit einbezieht; zum Beispiel wenn PROJ.4 nicht mit grid-shift Dateien kompiliert wurde oder die Koordinate außerhalb des durch die grid-shift Datei festgelegten Bereiches liegt. Standardmäßig gibt PostGIS eine Fehlermeldung aus, wenn keine grid-shift Datei vorhanden ist. Dieses Verhalten kann allerdings durch SRID basiertes ausprobieren von unterschiedlichen `to_proj` Werten des PROJ.4 Textes, oder durch Änderung des `proj4text` Wertes in der Tabelle `spatial_ref_sys` konfiguriert werden.

Zum Beispiel ist der `proj4text` Parameter `+datum=NAD87` die Kurzform für die folgenden `+nadgrids` Parameter:

```
+nadgrids=@conus,@alaska,@ntv2_0.gsb,@ntv1_can.dat
```

Der Präfix @ bedeutet, dass kein Fehler gemeldet wird, wenn die Dateien nicht vorhanden sind, aber wenn das Ende der Liste erreicht ist und keine Datei geeignet war (insbes. gefunden und überlagert wurde) so wird ein Fehler gemeldet.

Umgekehrt, wenn Sie sicherstellen wollen, dass zumindest die Standarddateien vorhanden sind, aber wenn alle Dateien ohne Treffer durchsucht wurden keine Transformation stattfindet, so können Sie folgendes verwenden:

```
+nadgrids=@conus,@alaska,@ntv2_0.gsb,@ntv1_can.dat,null
```

Die null-grid-shift Datei ist eine gültige grid-shift Datei, welche die gesamte Erde umfasst und zu keiner Verschiebung führt. Für ein vollständiges Beispiel, wenn Sie PostGIS so abändern wollen, dass Transformationen nach SRID 4267, welche nicht in dem zutreffenden Bereich liegen, zu keinem Fehler führen, können Sie folgendes anwenden:

```
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

[PostGIS_Full_Version](#), [ST_AsText](#), [ST_SetSRID](#), [UpdateGeometrySRID](#)

8.6.32 ST_Translate

ST_Translate — Verschiebt eine Geometrie um die angegebenen Versätze.

Synopsis

```
geometry ST_Translate(geometry g1, float deltax, float deltay);
geometry ST_Translate(geometry g1, float deltax, float deltay, float deltaz);
```

Beschreibung

Gibt eine neue Geometrie zurück, deren Koordinaten umgewandelte delta x, delta y, delta z Einheiten sind. Die Einheiten basieren auf jenen Einheiten, die im Koordinatenreferenzsystem (SRID) für diese Geometrie vorgegeben sind.



Note

Vor 1.3.4 ist diese Funktion abgestürzt, wenn die Geometrien CURVES enthalten. Dies wurde mit 1.3.4+ behoben

Verfügbarkeit: 1.2.2



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



This method supports Circular Strings and Curves

Beispiele

Einen Punkt um 1 Grad Länge verschieben

```
SELECT ST_AsText(ST_Translate(ST_GeomFromText('POINT(-71.01 42.37)',4326),1,0)) As ↵
    wgs_transgeomtxt;

    wgs_transgeomtxt
    -----
    POINT(-70.01 42.37)
```

Einen Linienzug um 1 Grad Länge und 1/2 Grad Breite verschieben

```
SELECT ST_AsText(ST_Translate(ST_GeomFromText('LINESTRING(-71.01 42.37,-71.11 42.38)',4326) ←
,1,0.5)) As wgs_transgeomtxt;
           wgs_transgeomtxt
-----
LINESTRING(-70.01 42.87,-70.11 42.88)
```

Einen 3D-Punkt verschieben

```
SELECT ST_AsEWKT(ST_Translate(CAST('POINT(0 0 0)' As geometry), 5, 12,3));
           st_asewkt
-----
POINT(5 12 3)
```

Eine Kurve und einen Punkt verschieben

```
SELECT ST_AsText(ST_Translate(ST_Collect('CURVEPOLYGON(CIRCULARSTRING(4 3,3.12 0.878,1 ←
0,-1.121 5.1213,6 7, 8 9,4 3))','POINT(1 3)'),1,2));

-----

GEOMETRYCOLLECTION(CURVEPOLYGON(CIRCULARSTRING(5 5,4.12 2.878,2 2,-0.121 7.1213,7 9,9 11,5 ←
5)),POINT(2 5))
```

Siehe auch

[ST_Affine](#), [ST_AsText](#), [ST_GeomFromText](#)

8.6.33 ST_TransScale

ST_TransScale — Umwandlung einer Geometrie entsprechend den gegebenen Skalierungsfaktoren und Versätzen.

Synopsis

geometry **ST_TransScale**(geometry geomA, float deltaX, float deltaY, float XFactor, float YFactor);

Beschreibung

Verschiebt die Geometrie entsprechend den Argumenten deltaX und deltaY; skaliert anschließend mittels der Argumente XFactor und YFactor. Funktioniert nur in 2D.



Note

`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)`.



Note

Vor 1.3.4 ist diese Funktion abgestürzt, wenn die Geometrien CURVES enthalten. Dies wurde mit 1.3.4+ behoben

Verfügbarkeit: 1.1.0



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



This method supports Circular Strings and Curves

Beispiele

```
SELECT ST_AsEWKT(ST_TransScale(ST_GeomFromEWKT('LINESTRING(1 2 3, 1 1 1)'), 0.5, 1, 1, 2));
           st_asewkt
-----
LINESTRING(1.5 6 3,1.5 4 1)

--Puffert einen Punkt, um die Näherung eines Kreises zu erhalten; wandelt ihn in eine Kurve ←
  um, verschiebt diese um 1,2 und skaliert diese um 3,4
SELECT ST_AsText(ST_Transscale(ST_LineToCurve(ST_Buffer('POINT(234 567)', 3)),1,2,3,4));

-----

CURVEPOLYGON(CIRCULARSTRING(714 2276,711.363961030679 2267.51471862576,705 ←
2264,698.636038969321 2284.48528137424,714 2276))
```

Siehe auch

[ST_Affine](#), [ST_Translate](#)

8.7 Geometrie Ausgabe

8.7.1 ST_AsBinary

ST_AsBinary — Gibt die Well-known-Binary(WKB)-Darstellung der Geometrie/Geographie ohne die SRID Metadaten zurück.

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

Gibt die Well-known Binary Darstellung der Geometrie aus. Die Funktion hat 2 Varianten. Die erste Variante nimmt einen nicht als Endian kodierten Parameter entgegen und setzt diesen in das Endian der Rechnerarchitektur um. Die zweite Variante nimmt ein zweites Argument, welches die Zeichenkodierung festlegt - nämlich Little-Endian ('NDR') oder Big-Endian ('XDR') Zeichenkodierung.

Dies ist nützlich mit binären Cursor, um Daten aus der Datenbank zu holen ohne dass diese in eine Zeichenkette-Darstellung konvertiert werden.

**Note**

Die WKB Spezifikation bezieht die SRID nicht mit ein. Um eine WKB-Darstellung im SRID-Format zu erhalten verwenden Sie bitte `ST_AsEWKB`.

**Note**

`ST_AsBinary` ist das Gegenstück von `ST_GeomFromWKB` für Geometrie. Verwenden Sie `ST_GeomFromWKB` um eine `ST_AsBinary` Darstellung in eine PostGIS Geometrie zu konvertieren.

**Note**

Das Standardverhalten von PostgreSQL 9.0 wurde dahingegen geändert, dass die Ausgabe von BYTEA jetzt in hexadezimaler Kodierung erfolgt. `ST_AsBinary` ist das Gegenteil von `ST_GeomFromWKB` für Geometrie. Falls Ihre graphischen Werkzeuge die alte Verhaltensweise benötigen, dann führen Sie bitte `SET bytea_output='escape'` in Ihrer Datenbank aus.

Erweiterung: 2.0.0 - Unterstützung für polyedrische Oberflächen, Dreiecke und TIN eingeführt.

Erweiterung: 2.0.0 - Unterstützung für höherdimensionale Koordinatensysteme eingeführt.

Erweiterung: 2.0.0 Unterstützung zum Festlegen des Endian beim geographischen Datentyp eingeführt.

Verfügbarkeit: 1.5.0 Unterstützung von geographischen Koordinaten.

Änderung: 2.0.0 - Eingabewerte für diese Funktion dürfen nicht "unknown" sein -- es muss sich um eine Geometrie handeln. Konstrukte, wie `ST_AsBinary('POINT(1 2)')`, sind nicht länger gültig und geben folgende Fehlermeldung aus: `n st_asbinary(unknown) is not unique error`. Dieser Code muss in `ST_AsBinary('POINT(1 2)::geometry');` geändert werden. Falls dies nicht möglich ist, so installieren Sie bitte `legacy.sql`.



This method implements the [OpenGIS Simple Features Implementation Specification for SQL 1.1](#). s2.1.1.1



This method implements the SQL/MM specification. SQL-MM 3: 5.1.37



This method supports Circular Strings and Curves



This function supports Polyhedral surfaces.



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



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

Beispiele

```
SELECT ST_AsBinary(ST_GeomFromText('POLYGON((0 0,0 1,1 1,1 0,0 0))',4326));
```

```
st_asbinary
```

```
-----
\001\003\000\000\000\001\000\000\000\005
\000\000\000\000\000\000\000\000\000\000
\000\000\000\000\000\000\000\000\000\000
\000\000\000\000\000\000\000\000\000\000
\000\000\000\360?\000\000\000\000\000\000
\360?\000\000\000\000\000\000\360?\000\000
```

```
\000\000\000\000\360?\000\000\000\000\000
\000\000\000\000\000\000\000\000\000\000
\000\000\000\000\000\000\000\000
(1 row)
```

```
SELECT ST_AsBinary(ST_GeomFromText('POLYGON((0 0,0 1,1 1,1 0,0 0))',4326), 'XDR');
          st_asbinary
-----
\000\000\000\000\003\000\000\000\001\000\000\000\005\000\000\000\000\000
\000\000\000\000\000\000\000\000\000\000\000\000\000\000\000\000\000\000
\000?\360\000\000\000\000\000\000?\360\000\000\000\000\000\000?\360\000\000
\000\000\000\000?\360\000\000\000\000\000\000\000\000\000\000\000\000\000
\000\000\000\000\000\000\000\000\000\000\000\000\000\000\000
(1 row)
```

Siehe auch

[ST_GeomFromWKB](#), [ST_AsEWKB](#), [ST_AsTWKB](#), [ST_AsText](#),

8.7.2 ST_AsEncodedPolyline

ST_AsEncodedPolyline — Erzeugt eine codierte Polylinie aus einer LineString Geometrie.

Synopsis

text **ST_AsEncodedPolyline**(geometry geom, integer precision=5);

Beschreibung

Gibt die Geometrie als codierte Polylinie zurück. Dieses Format ist geeignet, wenn Sie Google Maps verwenden.

Verfügbarkeit: 2.2.0

Beispiele

Grundlegendes

```
SELECT ST_AsEncodedPolyline(GeomFromEWKT('SRID=4326;LINESTRING(-120.2 38.5,-120.95 40.7,-126.453 43.252)'));
--result--
|_p~iF~ps|U_ulLnnqC_mqNvxq`@
```

Anwendung in Verbindung mit **LINESTRING** und **ST_Segmentize** für den geographischen Datentyp, und auf Google Maps stellen

```
-- das SQL von Boston nach San Francisco, segmentiert alle 100 KM
SELECT ST_AsEncodedPolyline(
    ST_Segmentize(
        ST_GeogFromText('LINESTRING(-71.0519 42.4935,-122.4483 37.64)'),
        100000)::geometry) As encodedFlightPath;
```

In JavaScript sieht dies ungefähr wie folgt aus, wobei die \$ Variable durch das Abfrageergebnis ersetzt wird

```

<script type="text/javascript" src="http://maps.googleapis.com/maps/api/js?libraries= ↵
  geometry"
></script>
<script type="text/javascript">
  flightPath = new google.maps.Polyline({
    path: google.maps.geometry.encoding.decodePath("$encodedFlightPath ↵
    "),
    map: map,
    strokeColor: '#0000CC',
    strokeOpacity: 1.0,
    strokeWeight: 4
  });
</script>

```

Siehe auch

[ST_LineFromEncodedPolyline](#), [ST_Segmentize](#)

8.7.3 ST_AsEWKB

ST_AsEWKB — Gibt die Well-known-Binary(WKB)-Darstellung der Geometrie mit den SRID Metadaten zurück.

Synopsis

```

bytea ST_AsEWKB(geometry g1);
bytea ST_AsEWKB(geometry g1, text NDR_or_XDR);

```

Beschreibung

Gibt die Well-known Binary Darstellung der Geometrie mit den SRID Metadaten aus. Die Funktion hat 2 Varianten. Die erste Variante nimmt einen nicht als Endian kodierten Parameter entgegen und setzt diesen auf Little-Endian. Die zweite Variante nimmt ein zweites Argument, welches die Zeichenkodierung festlegt - nämlich Little-Endian ('NDR') oder Big-Endian ('XDR') Zeichenkodierung.

Dies ist nützlich mit binären Cursor, um Daten aus der Datenbank zu holen ohne dass diese in eine Zeichenkette-Darstellung konvertiert werden.



Note

Die WKT Spezifikation bezieht die SRID nicht mit ein. Für die OGC WKB-Darstellung verwenden Sie bitte `ST_AsBinary`.



Note

`ST_AsEWKB` ist das Gegenstück von `ST_GeomFromEWKB`. Verwenden Sie `ST_GeomFromEWKB` um eine `ST_AsEWKB` Darstellung in eine PostGIS Geometrie zu konvertieren.

Erweiterung: 2.0.0 - Unterstützung für polyedrische Oberflächen, Dreiecke und TIN eingeführt.



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



This method supports Circular Strings and Curves



This function supports Polyhedral surfaces.



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

Beispiele

```
SELECT ST_AsEWKB(ST_GeomFromText('POLYGON((0 0,0 1,1 1,1 0,0 0))',4326));
```

```

          st_asewkb
-----
\001\003\000\000 \346\020\000\000\001\000
\000\000\005\000\000\000\000
\000\000\000\000\000\000\000\000
\000\000\000\000\000\000\000\000
\000\000\000\000\000\000\000\000
\000\000\360?\000\000\000\000\000\000\360?
\000\000\000\000\000\000\360?\000\000\000\000\000
\000\360?\000\000\000\000\000\000\000\000\000\000
\000\000\000\000\000\000\000\000\000\000\000
(1 row)
```

```
SELECT ST_AsEWKB(ST_GeomFromText('POLYGON((0 0,0 1,1 1,1 0,0 0))',4326), 'XDR');
          st_asewkb
```

```

-----
\000 \000\000\003\000\000\020\346\000\000\000\001\000\000\000\005\000\000\000\000\
000\000\000\000\000\000\000\000\000\000\000\000\000\000\000\000\000\000\000?
\360\000\000\000\000\000\000\000?\360\000\000\000\000\000\000\000\000\000\000
\000\000?\360\000\000\000\000\000\000\000\000\000\000\000\000\000\000\000\000
\000\000\000\000\000\000\000\000\000\000\000\000\000\000\000\000\000\000
```

Siehe auch

[ST_AsBinary](#), [ST_AsEWKT](#), [ST_AsText](#), [ST_GeomFromEWKT](#), [ST_SRID](#)

8.7.4 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(geography g1);
```

Beschreibung

Gibt die Well-known-Text Darstellung der Geometrie mit vorausgestellter SRID zurück.



Note

Die WKT Spezifikation bezieht die SRID nicht mit ein. Für die OGC WKT-Darstellung verwenden Sie bitte `ST_AsText`.



 Das WKT Format erhält die Genauigkeit von Fließpunktzahlen nicht. Um das Abschneiden von Kommastellen zu verhindern, benutzen Sie bitte das ST_AsBinary oder das ST_AsEWBK Format für die Übertragung.



Note

ST_AsEWKT ist die Umkehrfunktion von **ST_GeomFromEWKT**. Verwenden Sie bitte **ST_GeomFromEWKT** um eine PostGIS Geometrie aus einer ST_AsEWKT Darstellung zu erstellen.

Erweiterung: Mit 2.0.0 wurde die Unterstützung für den geographischen Datentyp, polyedrische Oberflächen, Dreiecke und TIN eingeführt.



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



This method supports Circular Strings and Curves



This function supports Polyhedral surfaces.



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

Beispiele

[illegible]

Siehe auch

ST_AsBinary, ST_AsEWKB, ST_AsText, ST_GeomFromEWKT

8.7.5 ST AsGeoJSON

ST AsGeoJSON — Gibt die Geometrie eines GeoJSON Elements zurück.

Synopsis

```
text ST_AsGeoJSON(geometry geom, integer maxdecimaldigits=15, integer options=0);
text ST_AsGeoJSON(geography geog, integer maxdecimaldigits=15, integer options=0);
text ST_AsGeoJSON(integer gj_version, geometry geom, integer maxdecimaldigits=15, integer options=0);
text ST_AsGeoJSON(integer gj_version, geography geog, integer maxdecimaldigits=15, integer options=0);
```

Beschreibung

Gibt die Geometrie als Geometrische Javascript Objekt Notation (GeoJSON) Element aus. (Vgl. [GeoJSON specifications 1.0](#)). Es werden sowohl 2D- als auch 3D-Geometrien unterstützt. GeoJSON unterstützt lediglich den geometrischer Datentyp SFS 1.1 (hat z.B. keine Unterstützung für Kurven).

Der Parameter `gj_version` ist die Hauptversion der GeoJSON Spezifikation. Diese muss 1 sein, wenn angegeben. Dies stellt die Version der GeoJSON Spezifikation dar.

Das dritte Argument kann verwendet werden um die maximale Anzahl der Dezimalstellen in der Ausgabe zu beschränken (Standardwert ist 15).

Mit dem letzten 'options' Argument kann eine BBox oder ein CRS zur GeoJSON Ausgabe hinzugefügt werden:

- 0: bedeutet keine Option (Standardwert)
- 1: GeoJSON Bbox
- 2: GeoJSON CRS-Kurzform (z.B. EPSG:4326)
- 4: GeoJSON CRS-Langform (z.B. urn:ogc:def:crs:EPSG::4326)

Version 1: `ST_AsGeoJSON(geom) / precision=15 version=1 options=0`

Version 2: `ST_AsGeoJSON(geom, precision) / version=1 options=0`

Version 3: `ST_AsGeoJSON(geom, precision, options) / version=1`

Version 4: `ST_AsGeoJSON(gj_version, geom) / precision=15 options=0`

Version 5: `ST_AsGeoJSON(gj_version, geom, precision) / options=0`

Version 6: `ST_AsGeoJSON(gj_version, geom, precision, options)`

Verfügbarkeit: 1.3.4

Verfügbarkeit: 1.5.0 Unterstützung von geographischen Koordinaten.

Änderung: 2.0.0 Unterstützung für Standardargumente und benannte Argumente.



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

Beispiele

Das GeoJSON Format ist im Allgemeinen bei AJAX-Karten effizienter als andere Formate. Ein beliebter JavaScript Client der dies unterstützt ist OpenLayers. Ein Anwendungsbeispiel finden Sie unter [GeoJSON specifications 1.0](#)

```
SELECT ST_AsGeoJSON(the_geom) from fe_edges limit 1;
               st_asgeojson
-----
{"type":"MultiLineString","coordinates":[[[-89.734634999999997,31.492072000000000],
[-89.734955999999997,31.492237999999997]]]}
(1 row)
--3d point
SELECT ST_AsGeoJSON('LINESTRING(1 2 3, 4 5 6)');
               st_asgeojson
-----
{"type":"LineString","coordinates":[[1,2,3],[4,5,6]]}
```

8.7.6 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 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. Der Versionsparameter kann 2 oder 3 sein, falls angegeben. Wenn kein Versionsparameter angegeben ist, wird dieser standardmäßig mit 2 angenommen. Das Genauigkeit-sargument "precision" kann verwendet werden, um die Anzahl der Dezimalstellen (maxdecimaldigits) bei der Ausgabe zu reduzieren (standardmäßig 15).

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 geographischen Koordinaten.

Erweiterung: 2.0.0 Unterstützung durch Präfix eingeführt. Für GML3 wurde die Option 4 eingeführt, um die Verwendung von LineString anstatt von Kurven für Linien zu erlauben. Ebenfalls wurde die GML3 Unterstützung für polyedrische Oberflächen und TINS eingeführt, sowie die Option 32 zur Ausgabe der BBox.

Änderung: 2.0.0 verwendet standardmäßig benannte Argumente.

Erweiterung: 2.1.0 Für GML 3 wurde die Unterstützung einer ID eingeführt.



Note

Nur die Version 3+ von ST_AsGML unterstützt polyedrische Oberflächen und TINs.



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



This function supports Polyhedral surfaces.



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

Beispiele: Version 2

```
SELECT ST_AsGML(ST_GeomFromText('POLYGON((0 0,0 1,1 1,1 0,0 0))',4326));
      st_asgml
      -----
      <gml:Polygon srsName="EPSG:4326"
><gml:outerBoundaryIs
><gml:LinearRing
><gml:coordinates
>0,0 0,1 1,1 1,0 0,0</gml:coordinates
></gml:LinearRing
></gml:outerBoundaryIs
></gml:Polygon
>
```

Beispiele: Version 3

```
-- Koordinaten umdrehen und Ausgabe in erweitertem EPSG (16 | 1)--
SELECT ST_AsGML(3, ST_GeomFromText('POINT(5.234234233242 6.34534534534)',4326), 5, 17);
      st_asgml
      -----
      <gml:Point srsName="urn:ogc:def:crs:EPSG::4326"
><gml:pos
>6.34535 5.23423</gml:pos
></gml:Point
>
```

```
-- Die Umhüllende/Envelope ausgeben (32) --
SELECT ST_AsGML(3, ST_GeomFromText('LINESTRING(1 2, 3 4, 10 20)',4326), 5, 32);
      st_asgml
      -----
      <gml:Envelope srsName="EPSG:4326">
        <gml:lowerCorner
>1 2</gml:lowerCorner>
        <gml:upperCorner
>10 20</gml:upperCorner>
      </gml:Envelope
>
```

```
-- Die Umhüllende (32) ausgeben, umgedreht (Breite/Länge anstatt Länge/Bereite) (16), long ←
srs (1)= 32 | 16 | 1 = 49 --
SELECT ST_AsGML(3, ST_GeomFromText('LINESTRING(1 2, 3 4, 10 20)',4326), 5, 49);
      st_asgml
      -----
      <gml:Envelope srsName="urn:ogc:def:crs:EPSG::4326">
        <gml:lowerCorner
>2 1</gml:lowerCorner>
        <gml:upperCorner
>20 10</gml:upperCorner>
      </gml:Envelope
>
```

```
-- Polyeder Beispiel --
SELECT ST_AsGML(3, ST_GeomFromEWKT('POLYHEDRALSURFACE( ((0 0 0, 0 0 1, 0 1 1, 0 1 0, 0 0 0) ←
),
((0 0 0, 0 1 0, 1 1 0, 1 0 0, 0 0 0)), ((0 0 0, 1 0 0, 1 0 1, 0 0 1, 0 0 0)),
((1 1 0, 1 1 1, 1 0 1, 1 0 0, 1 1 0)),
((0 1 0, 0 1 1, 1 1 1, 1 1 0, 0 1 0)), ((0 0 1, 1 0 1, 1 1 1, 0 1 1, 0 0 1)) )' ));
      st_asgml
      -----
<gml:PolyhedralSurface>
<gml:polygonPatches>
  <gml:PolygonPatch>
    <gml:exterior>
      <gml:LinearRing>
        <gml:posList srsDimension="3"
>0 0 0 0 0 1 0 1 1 0 1 0 0 0 0</gml:posList>
        </gml:LinearRing>
      </gml:exterior>
    </gml:PolygonPatch>
    <gml:PolygonPatch>
      <gml:exterior>
        <gml:LinearRing>
          <gml:posList srsDimension="3"
>0 0 0 0 1 0 1 1 0 1 0 0 0 0 0</gml:posList>
          </gml:LinearRing>
        </gml:exterior>
      </gml:PolygonPatch>
      <gml:PolygonPatch>
        <gml:exterior>
          <gml:LinearRing>
            <gml:posList srsDimension="3"
>0 0 0 1 0 0 1 0 1 0 0 1 0 0 0</gml:posList>
            </gml:LinearRing>
          </gml:exterior>
        </gml:PolygonPatch>
        <gml:PolygonPatch>
          <gml:exterior>
            <gml:LinearRing>
              <gml:posList srsDimension="3"
>1 1 0 1 1 1 1 0 1 1 0 0 1 1 0</gml:posList>
              </gml:LinearRing>
            </gml:exterior>
          </gml:PolygonPatch>
          <gml:PolygonPatch>
            <gml:exterior>
              <gml:LinearRing>
                <gml:posList srsDimension="3"
>0 1 0 0 1 1 1 1 1 1 1 0 0 1 0</gml:posList>
                </gml:LinearRing>
              </gml:exterior>
            </gml:PolygonPatch>
            <gml:PolygonPatch>
              <gml:exterior>
                <gml:LinearRing>
                  <gml:posList srsDimension="3"
>0 0 1 1 0 1 1 1 1 0 1 1 0 0 1</gml:posList>
                  </gml:LinearRing>
                </gml:exterior>
              </gml:PolygonPatch>
            </gml:polygonPatches>
          </gml:PolyhedralSurface>
        >
```

Siehe auch

[ST_GeomFromGML](#)

8.7.7 ST_AsHEXEWKB

ST_AsHEXEWKB — Gibt eine Geometrie im HEXEWKB Format (als Text) aus; verwendet entweder die Little-Endian (NDR) oder die Big-Endian (XDR) Zeichenkodierung.

Synopsis

```
text ST_AsHEXEWKB(geometry g1, text NDRorXDR);
text ST_AsHEXEWKB(geometry g1);
```

Beschreibung

Gibt eine Geometrie im HEXEWKB Format (als Text) aus; verwendet entweder die Little-Endian (NDR) oder die Big-Endian (XDR) Zeichenkodierung. Wenn keine Zeichenkodierung angegeben wurde, wird NDR verwendet.



Note

Verfügbarkeit: 1.2.2



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



This method supports Circular Strings and Curves

Beispiele

```
SELECT ST_AsHEXEWKB(ST_GeomFromText('POLYGON((0 0,0 1,1 1,1 0,0 0))',4326));
--gibt die selbe Antwort wie

SELECT ST_GeomFromText('POLYGON((0 0,0 1,1 1,1 0,0 0))',4326)::text;

st_ashexewkb
-----
0103000020E6100000010000000500
000000000000000000000000000000
0000000000000000000000000000F03F
000000000000F03F000000000000F03F000000000000F03
F000000000000000000000000000000000000000000000000
```

8.7.8 ST_AsKML

ST_AsKML — Gibt die Geometrie als ein KML Element aus. Mehrere Varianten. Standardmäßig ist version=2 und precision=15

Synopsis

```
text ST_AsKML(geometry geom, integer maxdecimaldigits=15);
text ST_AsKML(geography geog, integer maxdecimaldigits=15);
text ST_AsKML(integer version, geometry geom, integer maxdecimaldigits=15, text nprefix=NULL);
text ST_AsKML(integer version, 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.

Version 1: `ST_AsKML(geom_or_geog, maxdecimaldigits) / version=2 / maxdecimaldigits=15`

Version 2: `ST_AsKML(version, geom_or_geog, maxdecimaldigits, nprefix) maxdecimaldigits=15 / nprefix=NULL`



Note

Setzt voraus, dass PostGIS mit Proj-Unterstützung kompiliert wurde. Verwenden Sie bitte `PostGIS_Full_Version`, um festzustellen ob mit proj kompiliert wurde.



Note

Verfügbarkeit: 1.2.2 - spätere Varianten ab 1.3.2 nehmen den Versionsparameter mit auf



Note

Erweiterung: 2.0.0 - Präfix Namensraum hinzugefügt. Standardmäßig kein Präfix



Note

Änderung: 2.0.0 verwendet Standardargumente und unterstützt benannte Argumente.



Note

Die Ausgabe AsKML funktioniert nicht bei Geometrien ohne SRID



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

Beispiele

```
SELECT ST_AsKML(ST_GeomFromText('POLYGON((0 0,0 1,1 1,1 0,0 0))',4326));
```

```

      st_askml
      -
      <Polygon
><outerBoundaryIs
><LinearRing
><coordinates
>0,0 0,1 1,1 1,0 0,0</coordinates
></LinearRing
></outerBoundaryIs
></Polygon>
```

```
--3D Linienzug
```

```

SELECT ST_AsKML('SRID=4326;LINESTRING(1 2 3, 4 5 6)');
<LineString
><coordinates
>1,2,3 4,5,6</coordinates
></LineString>

```

Siehe auch

[ST_AsSVG](#), [ST_AsGML](#)

8.7.9 ST_AsLatLonText

ST_AsLatLonText — Gibt die "Grad, Minuten, Sekunden"-Darstellung für den angegebenen Punkt aus.

Synopsis

text **ST_AsLatLonText**(geometry pt, text format=');

Beschreibung

Gibt die "Grad, Minuten, Sekunden"-Darstellung des Punktes aus.



Note

Es wird angenommen, dass der Punkt in einer Breite/Länge-Projektion vorliegt. Die X (Länge) und Y (Breite) Koordinaten werden bei der Ausgabe in den "üblichen" Bereich (-180 to +180 für die Länge, -90 to +90 für die Breite) normalisiert.

Der Textparameter ist eine Zeichenkette für die Formatierung der Ausgabe, ähnlich wie die Zeichenkette für die Formatierung der Datumsausgabe. Gültige Zeichen sind "D" für Grad/Degrees, "M" für Minuten, "S" für Sekunden, und "C" für die Himmelsrichtung (NSEW). DMS Zeichen können wiederholt werden, um die gewünschte Zeichenbreite und Genauigkeit anzugeben ("SSS.SSSS" bedeutet z.B. " 1.0023").

"M", "S", und "C" sind optional. Wenn "C" weggelassen wird, werden Grad mit einem "-" Zeichen versehen, wenn Süd oder West. Wenn "S" weggelassen wird, werden die Minuten als Dezimalzahl mit der vorgegebenen Anzahl an Kommastellen angezeigt. Wenn "M" weggelassen wird, werden die Grad als Dezimalzahl mit der vorgegebenen Anzahl an Kommastellen angezeigt.

Wenn die Zeichenkette für das Ausgabeformat weggelassen wird (oder leer ist) wird ein Standardformat verwendet.

Verfügbarkeit: 2.0

Beispiele

Standardformat.

```

SELECT (ST_AsLatLonText('POINT (-3.2342342 -2.32498)'));
      st_aslatlon
-----
2\textdegree{}19'29.928"S 3\textdegree{}14'3.243"W

```

Ein Format angeben (identisch mit Standardformat).

```
SELECT (ST_AsLatLonText('POINT (-3.2342342 -2.32498)', 'D\textdegree{}M'S.SSS'C'));
      st_aslatlontext
-----
2\textdegree{}19'29.928"S 3\textdegree{}14'3.243"W
```

Andere Zeichen als D, M, S, C und "." werden lediglich durchgereicht.

```
SELECT (ST_AsLatLonText('POINT (-3.2342342 -2.32498)', 'D degrees, M minutes, S seconds to the C'));
      st_aslatlontext
-----
2 degrees, 19 minutes, 30 seconds to the S 3 degrees, 14 minutes, 3 seconds to the W
```

Grad mit einem Vorzeichen versehen - anstatt der Himmelsrichtung.

```
SELECT (ST_AsLatLonText('POINT (-3.2342342 -2.32498)', 'D\textdegree{}M'S.SSS''));
      st_aslatlontext
-----
-2\textdegree{}19'29.928" -3\textdegree{}14'3.243"
```

Dezimalgrad.

```
SELECT (ST_AsLatLonText('POINT (-3.2342342 -2.32498)', 'D.DDDD degrees C'));
      st_aslatlontext
-----
2.3250 degrees S 3.2342 degrees W
```

Überhöhte Werte werden normalisiert.

```
SELECT (ST_AsLatLonText('POINT (-302.2342342 -792.32498)'));
      st_aslatlontext
-----
72\textdegree{}19'29.928"S 57\textdegree{}45'56.757"E
```

8.7.10 ST_AsSVG

ST_AsSVG — Gibt ein Geobjekt als SVG-Pfadgeometrie zurück. Unterstützt den geometrischen und den geographischen Datentyp.

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.



Note

Verfügbarkeit: 1.2.2. Änderung: 1.4.0 L-Befehl beim absoluten Pfad aufgenommen, um mit <http://www.w3.org/TR/SVG/paths.html#PathDataBNF> konform zu sein.

Änderung: 2.0.0 verwendet Standardargumente und unterstützt benannte Argumente.


```

----- st_astext -----
POLYGON((0 0,0 1,1 1,1 0,0 0))
(1 row)

```

Siehe auch

[ST_AsBinary](#), [ST_AsEWKB](#), [ST_AsEWKT](#), [ST_GeomFromText](#)

8.7.12 ST_AsTWKB

ST_AsTWKB — Gibt die Geometrie als TWKB, aka "Tiny Well-known Binary" zurück

Synopsis

bytea **ST_AsTWKB**(geometry g1, integer decimaldigits_xy=0, integer decimaldigits_z=0, integer decimaldigits_m=0, boolean include_sizes=false, boolean include_bounding_boxes=false);
 bytea **ST_AsTWKB**(geometry[] geometries, bigint[] unique_ids, integer decimaldigits_xy=0, integer decimaldigits_z=0, integer decimaldigits_m=0, boolean include_sizes=false, boolean include_bounding_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 Einerstelle 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 Sammelgeometrie und eindeutige Identifikatoren in eine TWKB-Collection zu konvertieren, welche die Identifikatoren erhält. Dies ist nützlich für Clients, die davon ausgehen, eine Sammelgeometrie auszupacken, um so auf zusätzliche Information über die internen Objekte zuzugreifen. Sie können das Feld mit der Funktion **array_agg** erstellen. Die anderen Parameter bewirken dasselbe wie bei dem einfachen Format dieser Funktion.



Note

Die Formatspezifikation steht Online unter <https://github.com/TWKB/Specification> zur Verfügung, und Code zum Aufbau eines JavaScript Clients findet sich unter <https://github.com/TWKB/twkb.js>.

Enhanced: 2.4.0 memory and speed improvements.

Verfügbarkeit: 2.2.0

Beispiele

```

SELECT ST_AsTWKB('LINESTRING(1 1,5 5)'::geometry);
----- st_astwkb -----
\x02000202020808

```


Um ein aggregiertes TWKB-Objekt inklusive Identifikatoren zu erzeugen, fassen Sie bitte die gewünschte Geometrie und Objekte zuerst mittels "array_agg()" zusammen und rufen anschließend die passende TWKB Funktion auf.

```
SELECT ST_AsTWKB(array_agg(geom), array_agg(gid)) FROM mytable;
           st_astwkb
-----
\x040402020400000202
```

Siehe auch

[ST_GeomFromTWKB](#), [ST_AsBinary](#), [ST_AsEWKB](#), [ST_AsEWKT](#), [ST_GeomFromText](#)

8.7.13 ST_AsX3D

ST_AsX3D — Gibt eine Geometrie im X3D XML Knotenelement-Format zurück: ISO-IEC-19776-1.2-X3DEncodings-XML

Synopsis

text **ST_AsX3D**(geometry g1, integer maxdecimaldigits=15, integer options=0);

Beschreibung

Gibt eine Geometrie als X3D knotenformatiertes XML Element zurück <http://www.web3d.org/standards/number/19776-1>. Falls maxdecimaldigits (Genauigkeit) nicht angegeben ist, wird sie standardmäßig 15.

Note



Es gibt verschiedene Möglichkeiten eine PostGIS Geometrie in X3D zu übersetzen, da sich der X3D Geometrietyp nicht direkt in den geometrischen Datentyp von PostGIS abbilden lässt. Einige neuere X3D Datentypen, die sich besser abbilden lassen könnten haben wir vermieden, da diese von den meisten Rendering-Tools zurzeit nicht unterstützt werden. Dies sind die Abbildungen für die wir uns entschieden haben. Falls Sie Ideen haben, wie wir es den Anwendern ermöglichen können ihre bevorzugten Abbildungen anzugeben, können Sie gerne ein Bug-Ticket senden.

Im Folgenden wird beschrieben, wie der PostGIS 2D/3D Datentyp derzeit in den X3D Datentyp abgebildet wird

Das Argument 'options' ist ein Bitfeld. Ab PostGIS 2.2+ wird dieses verwendet, um anzuzeigen ob die Koordinaten als X3D geospatiale Knoten in GeoKoordinaten dargestellt werden und auch ob X- und Y-Achse vertauscht werden sollen. Standardmäßig erfolgt die Ausgabe durch ST_AsX3D im Datenbankformat (Länge, Breite oder X,Y), aber es kann auch der X3D Standard mit Breite/Länge oder Y/X bevorzugt werden.

- 0: X/Y in der Datenbankreihenfolge (z.B. ist Länge/Breite = X,Y die standardmäßige Datenbankreihenfolge), Standardwert, und nicht-spatiale Koordinaten (nur der normale alte Koordinaten-Tag).
- 1: X und Y umdrehen. In Verbindung mit der Option für GeoKoordinaten wird bei der Standardausgabe die Breite zuerst/"latitude_first" ausgegeben und die Koordinaten umgedreht.
- 2: Die Koordinaten werden als geospatiale GeoKoordinaten ausgegeben. Diese Option gibt eine Fehlermeldung aus, falls die Geometrie nicht in WGS 84 Länge/Breite (SRID: 4326) vorliegt. Dies ist zurzeit der einzige GeoKoordinaten-Typ der unterstützt wird. [Siehe die X3D Spezifikation für Koordinatenreferenzsysteme](#). Die Standardausgabe ist GeoCoordinate geoSystem=' "GD" "WE" "longitude_first" '. Wenn Sie den X3D Standard bevorzugen GeoCoordinate geoSystem=' "GD" "WE" "latitude_first" ' verwenden Sie bitte (2+1) = 3

PostGIS Datentyp	2D X3D Datentyp	3D X3D Datentyp
PostGIS Datentyp	2D X3D Datentyp	3D X3D Datentyp
LINESTRING	zurzeit nicht implementiert - wird PolyLine2D	LineSet
MULTILINESTRING	zurzeit nicht implementiert - wird PolyLine2D	IndexedLineSet
MULTIPOINT	Polypoint2D	PointSet
POINT	gibt leerzeichengetrennte Koordinaten aus	gibt leerzeichengetrennte Koordinaten aus
(MULTI) POLYGON, POLYHEDRALSURFACE	Ungültiges X3D Markup	IndexedFaceSet (die inneren Ringe werden zurzeit als ein weiteres FaceSet abgebildet)
TIN	TriangleSet2D (zurzeit nicht implementiert)	IndexedTriangleSet

**Note**

Die Unterstützung von 2D-Geometrie ist noch nicht vollständig. Die inneren Ringe werden zur Zeit lediglich als gesonderte Polygone abgebildet. Wir arbeiten daran.

Bezüglich 3D sind viele Weiterentwicklungen im Gange, insbesondere in Bezug auf **X3D Integration mit HTML5**

Es gibt auch einen feinen OpenSource X3D Viewer, den Sie benützen können, um Geometrien darzustellen. Free Wrl <http://freewrl.sourceforge.net/>

Binärdateien sind für Mac, Linux und Windows verfügbar. Sie können den mitgelieferten FreeWRL_Launcher verwenden, um Geometrien darzustellen.

Riskieren Sie auch einen Blick auf **PostGIS minimalist X3D viewer**, der diese Funktionalität einsetzt und auf **x3dDom HTML/JS OpenSource Toolkit**.

Verfügbarkeit: 2.0.0: ISO-IEC-19776-1.2-X3DEncodings-XML

Erweiterung: 2.2.0: Unterstützung für geographische Koordinaten und Vertauschen der Achsen (x/y, Länge/Breite). Für nähere Details siehe Optionen.



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



This function supports Polyhedral surfaces.



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

Beispiel: Erzeugung eines voll funktionsfähigen X3D Dokuments - Dieses erzeugt einen Würfel, den man sich mit FreeWrl und anderen X3D-Viewern ansehen kann.

```
SELECT '<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE X3D PUBLIC "ISO//Web3D//DTD X3D 3.0//EN" "http://www.web3d.org/specifications/x3d
-3.0.dtd">
<X3D>
  <Scene>
    <Transform>
      <Shape>
        <Appearance>
          <Material emissiveColor=''0 0 1''/>
        </Appearance>
      </Shape>
    </Transform>
  </Scene>
</X3D>
> ' ||
```

```

        ST_AsX3D( ST_GeomFromEWKT('POLYHEDRALSURFACE( ((0 0 0, 0 0 1, 0 1 1, 0 1 0, 0 0 0)),
((0 0 0, 0 1 0, 1 1 0, 1 0 0, 0 0 0)), ((0 0 0, 1 0 0, 1 0 1, 0 0 1, 0 0 0)),
((1 1 0, 1 1 1, 1 0 1, 1 0 0, 1 1 0)),
((0 1 0, 0 1 1, 1 1 1, 1 1 0, 0 1 0)), ((0 0 1, 1 0 1, 1 1 1, 0 1 1, 0 0 1)) )' ) ||
        '</Shape>'
    </Transform>
</Scene>
</X3D>
>' As x3ddoc;

        x3ddoc
        -----
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE X3D PUBLIC "ISO//Web3D//DTD X3D 3.0//EN" "http://www.web3d.org/specifications/x3d ←
-3.0.dtd">
<X3D>
  <Scene>
    <Transform>
      <Shape>
        <Appearance>
          <Material emissiveColor='0 0 1' />
        </Appearance>
        <IndexedFaceSet coordIndex='0 1 2 3 -1 4 5 6 7 -1 8 9 10 11 -1 12 13 14 15 -1 16 17 ←
18 19 -1 20 21 22 23'>
          <Coordinate point='0 0 0 0 0 1 0 1 1 0 1 0 0 0 0 0 1 0 1 1 0 1 0 0 0 0 0 1 0 0 ←
1 0 1 0 0 1 1 1 0 1 1 1 1 0 1 1 0 0 0 1 1 1 1 1 1 1 1 0 0 0 1 1 0 1 1 1 ←
1 0 1 1' />
        </IndexedFaceSet>
      </Shape>
    </Transform>
  </Scene>
</X3D>
>

```

Beispiel: Ein Achteck, um 3 Einheiten gehoben und mit einer dezimalen Genauigkeit von 6

```

SELECT ST_AsX3D(
ST_Translate(
  ST_Force_3d(
    ST_Buffer(ST_Point(10,10),5, 'quad_segs=2')), 0,0,
    3)
,6) As x3dfrag;

x3dfrag
-----
<IndexedFaceSet coordIndex="0 1 2 3 4 5 6 7">
  <Coordinate point="15 10 3 13.535534 6.464466 3 10 5 3 6.464466 6.464466 3 5 10 3 ←
6.464466 13.535534 3 10 15 3 13.535534 13.535534 3 " />
</IndexedFaceSet>
>

```

Beispiel: TIN

```

SELECT ST_AsX3D(ST_GeomFromEWKT('TIN (((
    0 0 0,
    0 0 1,
    0 1 0,
    0 0 0

```

```

   )), ((
      0 0 0,
      0 1 0,
      1 1 0,
      0 0 0
    ))
  )'') As x3dfrag;

      x3dfrag
      -----
<IndexedTriangleSet  index='0 1 2 3 4 5'
><Coordinate point='0 0 0 0 0 1 0 1 0 0 0 0 0 1 0 1 1 0' /></IndexedTriangleSet
>

```

Beispiel: Geschlossener MultiLinestring (die Begrenzung eines Polygons mit Lücken)

```

SELECT ST_AsX3D(
      ST_GeomFromEWKT('MULTILINESTRING((20 0 10,16 -12 10,0 -16 10,-12 -12  ←
      10,-20 0 10,-12 16 10,0 24 10,16 16 10,20 0 10),
      (12 0 10,8 8 10,0 12 10,-8 8 10,-8 0 10,-8 -4 10,0 -8 10,8 -4 10,12 0 10)))')
) As x3dfrag;

      x3dfrag
      -----
<IndexedLineSet  coordIndex='0 1 2 3 4 5 6 7 0 -1 8 9 10 11 12 13 14 15 8'>
  <Coordinate point='20 0 10 16 -12 10 0 -16 10 -12 -12 10 -20 0 10 -12 16 10 0 24 10 16  ←
      16 10 12 0 10 8 8 10 0 12 10 -8 8 10 -8 0 10 -8 -4 10 0 -8 10 8 -4 10 ' />
</IndexedLineSet
>

```

8.7.14 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

Gibt die Geometrie in der GeoHash-Darstellung (<http://en.wikipedia.org/wiki/Geohash>) aus. Ein GeoHash codiert einen Punkt in einem Textformat, das über Präfixe sortierbar und durchsuchbar ist. Ein kürzer codierter GeoHash ergibt eine ungenauere Darstellung des Punktes. Man kann sich einen GeoHash auch als eine Box vorstellen, welche den tatsächlichen Punkt enthält.

Wenn `maxchars` nicht angegeben wird, gibt ST_GeoHash einen GeoHash mit der vollen Genauigkeit der Eingabegeometrie zurück. Punkte ergeben so einen GeoHash mit einer Genauigkeit von 20 Zeichen (dies sollte ausreichen um die Eingabe in Double Precision zur Gänze abzuspeichern). Andere Varianten geben einen Geohash, basierend auf der Größe des Geoobjektes, mit veränderlicher Genauigkeit zurück, Größere Geoobjekte werden mit geringerer, kleinere Geoobjekte mit höherer Genauigkeit dargestellt. Die Idee dahinter ist, dass die durch den GeoHash implizierte Box immer das gegebene Geoobjekt beinhaltet.

Wenn `maxchars` angegeben wird, gibt ST_GeoHash einen GeoHash zurück, der maximal die Anzahl dieser Zeichen aufweist. Auf diese Weise ist es möglich die Eingabegeometrie mit einer geringeren Präzision darzustellen. Bei Nicht-Punkten befindet sich der Anfangspunkt der Berechnung im Mittelpunkt des Umgebungsrechtecks der Geometrie.

Verfügbarkeit: 1.4.0

**Note**

ST_GeoHash funktioniert nicht, wenn die Geometrien nicht in geographischen (Länge/Breite) Koordinaten vorliegen.



This method supports Circular Strings and Curves

Beispiele

```
SELECT ST_GeoHash(ST_SetSRID(ST_MakePoint(-126,48),4326));

      st_geohash
-----
c0w3hf1s70w3hf1s70w3

SELECT ST_GeoHash(ST_SetSRID(ST_MakePoint(-126,48),4326),5);

      st_geohash
-----
c0w3h
```

Siehe auch

[ST_GeomFromGeoHash](#)

8.7.15 ST_AsGeobuf

ST_AsGeobuf — Gibt eine Menge an Zeilen in der Geobuf Darstellung aus.

Synopsis

```
bytea ST_AsGeobuf(anyelement set row);
bytea ST_AsGeobuf(anyelement row, text geom_name);
```

Beschreibung

Gibt Zeilen einer FeatureCollection in der Geobuf Darstellung (<https://github.com/mapbox/geobuf>) aus. Von jeder Eingabegeometrie wird die maximale Genauigkeit analysiert, um eine optimale Speicherung zu erreichen. Anmerkung: In der jetzigen Form kann Geobuf nicht "gestreamt" werden, wodurch die gesamte Ausgabe im Arbeitsspeicher zusammengestellt wird.

`row` Datenzeilen mit zumindest einer Geometriespalte.

`geom_name` ist die Bezeichnung der Geometriespalte in den Datenzeilen. Wenn NULL, dann wird standardmäßig die erste aufgefundene Geometriespalte verwendet.

Verfügbarkeit: 2.4.0

Beispiele

```
SELECT encode(ST_AsGeobuf(q, 'geom'), 'base64')
  FROM (SELECT ST_GeomFromText('POLYGON((0 0,0 1,1 1,1 0,0 0))') AS geom) AS q;
st_asgeobuf
-----
GAAiEAoOCgwIBBoIAAAAAgIAAAE=
```

8.7.16 ST_AsMVTGeom

ST_AsMVTGeom — Transformiert eine Geometrie in das Koordinatensystem eines [Mapbox Vector Tiles](#).

Synopsis

geometry **ST_AsMVTGeom**(geometry geom, box2d bounds, integer extent=4096, integer buffer=256, boolean clip_geom=true);

Beschreibung

Transformiert eine Geometrie in das Koordinatensystem eines [Mapbox Vector Tiles](#) aus Zeilen die einem Layer entsprechen. Unternimmt alle Anstrengungen, damit die Geometrie valide bleibt oder korrigiert sie eventuell sogar. Bei diesem Prozess kann es vorkommen, dass die Geometrie in eine niedrigere Dimension übergeführt wird.

geom Die zu Transformierende Geometrie.

bounds ist die geometrische Abgrenzung des Inhalts der Kachel ohne Puffer.

extent ist die Größe der Kachel, angegeben im Koordinatensystem der Vektorkacheln und so wie in der [Spezifikation](#) festgelegt. Wenn dieser Wert NULL ist, wird der Standardwert 4096 angenommen.

buffer ist die Puffergröße im Koordinatensystem der Vektorkacheln, an der die Geometrie optional ausgeschnitten werden kann. Wenn NULL, dann wird der Standardwert 256 angenommen.

clip_geom ist ein boolescher Eingabewert, der bestimmt ob die Geometrie ausgeschnitten werden soll, oder so wie sie vorliegt codiert wird. Wenn der Wert NULL ist, wird der Standardwert TRUE angenommen.

Verfügbarkeit: 2.4.0

Beispiele

```
SELECT ST_AsText(ST_AsMVTGeom(
  ST_GeomFromText('POLYGON ((0 0, 10 0, 10 5, 0 -5, 0 0))'),
  ST_MakeBox2D(ST_Point(0, 0), ST_Point(4096, 4096)),
  4096, 0, false));
st_astext
-----
MULTIPOLYGON(((5 4096,10 4096,10 4091,5 4096)),((5 4096,0 4096,0 4101,5 4096)))
```

8.7.17 ST_AsMVT

ST_AsMVT — Gibt Zeilen in der [Mapbox Vector Tile](#) Darstellung aus.

Synopsis

```
bytea ST_AsMVT(anelement set row);
bytea ST_AsMVT(anelement row, text name);
bytea ST_AsMVT(anelement row, text name, integer extent);
bytea ST_AsMVT(anelement row, text name, integer extent, text geom_name);
```

Beschreibung

Gibt eine Menge an Zeilen, die einem Layer entsprechen, in der **Mapbox Vector Tile** Darstellung aus. Mehrere Aufrufe können aneinandergereiht werden, um eine Kachel mit mehreren Layern zu erstellen. Es wird angenommen, dass die Geometrie im Koordinatensystem der Vektorkacheln vorliegt und entsprechend der **Spezifikation** valide ist. Üblicherweise wird **ST_AsMVTGeom** verwendet, um die Geometrie in das Koordinatensystem der Vektorkacheln zu transformieren. Die übrigen Daten werden als Attribute codiert.

Das **Mapbox Vector Tile** Format kann Geoobjekte mit unterschiedlichen Attributen pro Feature speichern. Um dies zu nutzen, legen Sie bitte eine JSONB-Spalte mit JSON-Objekten in den Rohdaten an. Die Schlüssel und Werte in dem Objekt werden in die Feature-Attribute zerlegt.



Important

Do not call with a `GEOMETRYCOLLECTION` as an element in the row. However you can use **ST_AsMVTGeom** to prep a geometry collection for inclusion.

`row` Datenzeilen mit zumindest einer Geometriespalte.

`name` ist der Name des Layers. Wenn NULL, dann wird die Zeichenfolge "default" verwendet.

`extent` ist die Kachelausdehnung in Bildschirmeneinheiten, so wie in der Spezifikation festgelegt. Wenn NULL, wird der Standardwert 4096 angenommen.

`geom_name` ist die Bezeichnung der Geometriespalte in den Datenzeilen. Wenn NULL, dann wird standardmäßig die erste aufgefundene Geometriespalte verwendet.

Verfügbarkeit: 2.4.0

Beispiele

```
SELECT ST_AsMVT(q, 'test', 4096, 'geom') FROM (SELECT 1 AS c1,
  ST_AsMVTGeom(ST_GeomFromText('POLYGON ((35 10, 45 45, 15 40, 10 20, 35 10), (20 30, 35 30, 30 20, 20 30))'),
  ST_MakeBox2D(ST_Point(0, 0), ST_Point(4096, 4096)), 4096, 0, false) AS geom) AS q;
-----
\ ←
x1a320a0474657374121d12020000180322150946ec3f1a14453b0a09280f091413121e09091e0f1a026331220228012
```

Siehe auch

ST_AsMVTGeom

8.8 Operatoren

8.8.1 &&

&& — Gibt `TRUE` zurück, wenn die 2D Bounding Box von A die 2D Bounding Box von B schneidet.

Synopsis

```
boolean &&( geometry A , geometry B );
boolean &&( geography A , geography B );
```

Beschreibung

Der **&&** Operator gibt `TRUE` zurück, wenn die 2D Bounding Box von Geometrie A die 2D Bounding Box der Geometrie von B schneidet.



Note

Dieser Operand benützt jeden Index, der für die Geometrie zur Verfügung stellt.

Erweiterung: Mit 2.0.0 wurde die Unterstützung für polyedrische Oberflächen eingeführt.

Verfügbarkeit: Mit 1.5.0 wurde die Unterstützung von geographischen Koordinaten eingeführt



This method supports Circular Strings and Curves



This function supports Polyhedral surfaces.

Beispiele

```
SELECT tbl1.column1, tbl2.column1, tbl1.column2 && tbl2.column2 AS overlaps
FROM ( VALUES
      (1, 'LINESTRING(0 0, 3 3)::geometry)',
      (2, 'LINESTRING(0 1, 0 5)::geometry)) AS tbl1,
 ( VALUES
      (3, 'LINESTRING(1 2, 4 6)::geometry)) AS tbl2;
```

column1	column1	overlaps
1	3	t
2	3	f

(2 rows)

Siehe auch

[|&>](#), [&>](#), [&<|](#), [&<](#), [~](#), [@](#)

8.8.2 &&(geometry,box2df)

&&(geometry,box2df) — Gibt `TRUE` zurück, wenn sich die 2D Bounding Box (cached) einer Geometrie mit einer 2D Bounding Box mit Gleitpunktgenauigkeit (BOX2DF) überschneidet.

Synopsis

boolean **&&**(geometry A , box2df B);

Beschreibung

Der **&&** Operator gibt TRUE zurück, wenn die im Cache befindliche 2D Bounding Box der Geometrie A sich mit der 2D Bounding Box von B, unter Verwendung von Gleitpunktgenauigkeit überschneidet. D.h.: falls B eine (double precision) box2d ist, wird diese intern in eine auf Gleitpunkt genaue 2D Bounding Box (BOX2DF) umgewandelt.



Note

Dieser Operand ist eher für die interne Nutzung durch BRIN Indizes, als durch die Anwender, gedacht.

Verfügbarkeit: Mit 2.3.0 wurde die Unterstützung von Block Range INdices (BRIN) eingeführt. Erfordert PostgreSQL 9.5+.



This method supports Circular Strings and Curves



This function supports Polyhedral surfaces.

Beispiele

```
SELECT ST_MakePoint(1,1) && ST_MakeBox2D(ST_MakePoint(0,0), ST_MakePoint(2,2)) AS overlaps;

overlaps
-----
t
(1 row)
```

Siehe auch

[&&\(box2df,geometry\)](#), [&&\(box2df,box2df\)](#), [~\(geometry,box2df\)](#), [~\(box2df,geometry\)](#), [~\(box2df,box2df\)](#), [@\(geometry,box2df\)](#), [@\(box2df,geometry\)](#), [@\(box2df,box2df\)](#)

8.8.3 &&(box2df,geometry)

&&(box2df,geometry) — Gibt TRUE zurück, wenn eine 2D float precision bounding box (BOX2DF) eine Geometrie (cached) 2D bounding box schneidet.

Synopsis

boolean **&&**(box2df A , geometry B);

Beschreibung

Der **&&** Operator gibt TRUE zurück, wenn die 2D Bounding Box A die zwischengespeicherte 2D Bounding Box der Geometrie B, unter Benutzung von Fließpunktgenauigkeit, schneidet. D.h.: wenn A eine (double precision) box2d ist, wird diese intern in eine float precision 2D bounding box (BOX2DF) umgewandelt.

**Note**

Dieser Operand ist eher für die interne Nutzung durch BRIN Indizes, als durch die Anwender, gedacht.

Verfügbarkeit: Mit 2.3.0 wurde die Unterstützung von Block Range INdices (BRIN) eingeführt. Erfordert PostgreSQL 9.5+.



This method supports Circular Strings and Curves



This function supports Polyhedral surfaces.

Beispiele

```
SELECT ST_MakeBox2D(ST_MakePoint(0,0), ST_MakePoint(2,2)) && ST_MakePoint(1,1) AS overlaps;

overlaps
-----
t
(1 row)
```

Siehe auch

[&&\(geometry,box2df\)](#), [&&\(box2df,box2df\)](#), [~\(geometry,box2df\)](#), [~\(box2df,geometry\)](#), [~\(box2df,box2df\)](#), [@\(geometry,box2df\)](#), [@\(box2df,geometry\)](#), [@\(box2df,box2df\)](#)

8.8.4 &&(box2df,box2df)

[&&\(box2df,box2df\)](#) — Gibt TRUE zurück, wenn sich zwei 2D float precision Bounding Boxes (BOX2DF) überschneiden.

Synopsis

boolean **&&**(box2df A , box2df B);

Beschreibung

Der **&&** Operator gibt TRUE zurück, wenn sich zwei 2D Bounding Boxes A und B, unter Benutzung von float precision, gegenseitig überschneiden. D.h.: Wenn A (oder B) eine (double precision) box2d ist, wird diese intern in eine float precision 2D bounding box (BOX2DF) umgewandelt

**Note**

Dieser Operator ist für die interne Nutzung durch BRIN Indizes, und nicht so sehr durch Anwender, vorgesehen.

Verfügbarkeit: Mit 2.3.0 wurde die Unterstützung von Block Range INdices (BRIN) eingeführt. Erfordert PostgreSQL 9.5+.



This method supports Circular Strings and Curves



This function supports Polyhedral surfaces.

Beispiele

```
SELECT ST_MakeBox2D(ST_MakePoint(0,0), ST_MakePoint(2,2)) && ST_MakeBox2D(ST_MakePoint(1,1) ←
, ST_MakePoint(3,3)) AS overlaps;

overlaps
-----
t
(1 row)
```

Siehe auch

[&&\(geometry,box2df\)](#), [&&\(box2df,geometry\)](#), [~\(geometry,box2df\)](#), [~\(box2df,geometry\)](#), [~\(box2df,box2df\)](#), [@\(geometry,box2df\)](#), [@\(box2df,geometry\)](#), [@\(box2df,box2df\)](#)

8.8.5 &&&

&&& — Gibt TRUE zurück, wenn A's n-D bounding box B's n-D bounding box schneidet.

Synopsis

boolean **&&&**(geometry A , geometry B);

Beschreibung

Der **&&&** Operator gibt TRUE zurück, wenn die n-D bounding box der Geometrie A die n-D bounding box der Geometrie B schneidet.



Note

Dieser Operand benützt jeden Index, der für die Geometrie zur Verfügung stellt.

Verfügbarkeit: 2.0.0



This method supports Circular Strings and Curves



This function supports Polyhedral surfaces.



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



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

Beispiele: 3D LineStrings

```
SELECT tbl1.column1, tbl2.column1, tbl1.column2 &&& tbl2.column2 AS overlaps_3d,
      tbl1.column2 && tbl2.column2 AS overlaps_2d
FROM ( VALUES
      (1, 'LINESTRING Z(0 0 1, 3 3 2)::geometry),
      (2, 'LINESTRING Z(1 2 0, 0 5 -1)::geometry)) AS tbl1,
( VALUES
```

```
(3, 'LINESTRING Z(1 2 1, 4 6 1)::geometry)) AS tbl2;
```

column1	column1	overlaps_3d	overlaps_2d
1	3	t	t
2	3	f	t

Beispiele: 3M LineStrings

```
SELECT tbl1.column1, tbl2.column1, tbl1.column2 &&& tbl2.column2 AS overlaps_3zm,
       tbl1.column2 && tbl2.column2 AS overlaps_2d
```

```
FROM ( VALUES
      (1, 'LINESTRING M(0 0 1, 3 3 2)::geometry),
      (2, 'LINESTRING M(1 2 0, 0 5 -1)::geometry)) AS tbl1,
  ( VALUES
      (3, 'LINESTRING M(1 2 1, 4 6 1)::geometry)) AS tbl2;
```

column1	column1	overlaps_3zm	overlaps_2d
1	3	t	t
2	3	f	t

Siehe auch

[&&](#)

8.8.6 &&&(geometry,gidx)

&&&(geometry,gidx) — Gibt `TRUE` zurück, wenn die (cached) n-D bounding box einer Geometrie eine n-D float precision bounding box (GIDX) schneidet.

Synopsis

```
boolean &&&( geometry A , gidx B );
```

Beschreibung

Der **&&&** Operator gibt `TRUE` zurück, wenn die zwischengespeicherte n-D bounding box der Geometrie A die n-D bounding box B, unter Benutzung von float precision, schneidet. D.h.: Wenn B eine (double precision) box3d ist, wird diese intern in eine float precision 3D bounding box (GIDX) umgewandelt



Note

Dieser Operator ist für die interne Nutzung durch BRIN Indizes, und nicht so sehr durch Anwender, vorgesehen.

Verfügbarkeit: Mit 2.3.0 wurde die Unterstützung von Block Range INdices (BRIN) eingeführt. Erfordert PostgreSQL 9.5+.



This method supports Circular Strings and Curves



This function supports Polyhedral surfaces.



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



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

Beispiele

```
SELECT ST_MakePoint(1,1,1) &&& ST_3DMakeBox(ST_MakePoint(0,0,0), ST_MakePoint(2,2,2)) AS overlaps;
overlaps
-----
t
(1 row)
```

Siehe auch

[&&&\(gidx,geometry\)](#), [&&&\(gidx,gidx\)](#)

8.8.7 &&&(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, wird diese intern in eine float precision 3D bounding box (GIDX) umgewandelt



Note

Dieser Operator ist für die interne Nutzung durch BRIN Indizes, und nicht so sehr durch Anwender, vorgesehen.

Verfügbarkeit: Mit 2.3.0 wurde die Unterstützung von Block Range INdices (BRIN) eingeführt. Erfordert PostgreSQL 9.5+.



This method supports Circular Strings and Curves



This function supports Polyhedral surfaces.



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



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

Beispiele

```
SELECT ST_3DMakeBox(ST_MakePoint(0,0,0), ST_MakePoint(2,2,2)) &&& ST_MakePoint(1,1,1) AS overlaps;

overlaps
-----
t
(1 row)
```

Siehe auch

[&&&\(geometry,gidx\), &&&\(gidx,gidx\)](#)

8.8.8 &&&(gidx,gidx)

[&&&\(gidx,gidx\)](#) — Gibt TRUE zurück, wenn sich zwei n-D float precision bounding boxes (GIDX) gegenseitig überschneiden.

Synopsis

boolean **&&&**(gidx A , gidx B);

Beschreibung

Der **&&&** Operator gibt TRUE zurück, wenn sich zwei n-D bounding boxes A und B, unter Benutzung von float precision, gegenseitig überschneiden. D.h.: wenn A (oder B) eine (double precision) box3d ist, wird diese intern in eine float precision 3D bounding box (GIDX) umgewandelt



Note

Dieser Operator ist für die interne Nutzung durch BRIN Indizes, und nicht so sehr durch Anwender, vorgesehen.

Verfügbarkeit: Mit 2.3.0 wurde die Unterstützung von Block Range INDEXes (BRIN) eingeführt. Erfordert PostgreSQL 9.5+.



This method supports Circular Strings and Curves



This function supports Polyhedral surfaces.



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



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

Beispiele

```
SELECT ST_3DMakeBox(ST_MakePoint(0,0,0), ST_MakePoint(2,2,2)) &&& ST_3DMakeBox(ST_MakePoint(1,1,1), ST_MakePoint(3,3,3)) AS overlaps;

overlaps
-----
t
(1 row)
```

Siehe auch

[&&\(geometry,gidx\)](#), [&&\(gidx,geometry\)](#)

8.8.9 &<

&< — Gibt TRUE zurück, wenn die bounding box der Geometrie A, die bounding box der Geometrie B überlagert oder links davon liegt.

Synopsis

```
boolean &<( geometry A , geometry B );
```

Beschreibung

Der **&<** Operator gibt TRUE zurück, wenn die bounding box der Geometrie A die bounding box der Geometrie B überlagert oder links davon liegt, oder präziser, überlagert und NICHT rechts von der bounding box der Geometrie B liegt.

**Note**

Dieser Operand benutzt jeden Index, der für die Geometrie zur Verfügung stellt.

Beispiele

```
SELECT tbl1.column1, tbl2.column1, tbl1.column2 &< tbl2.column2 AS overleft
FROM
  ( VALUES
    (1, 'LINESTRING(1 2, 4 6)::geometry) AS tbl1,
  ( VALUES
    (2, 'LINESTRING(0 0, 3 3)::geometry),
    (3, 'LINESTRING(0 1, 0 5)::geometry),
    (4, 'LINESTRING(6 0, 6 1)::geometry) AS tbl2;

column1 | column1 | overleft
-----+-----+-----
        1 |         2 | f
        1 |         3 | f
        1 |         4 | t
(3 rows)
```

Siehe auch

[&&](#), [!&>](#), [&>](#), [&<|](#)

8.8.10 &<|

&<| — Gibt TRUE zurück, wenn die bounding box von A jene von B überlagert oder unterhalb liegt.

Synopsis

```
boolean &<|( geometry A , geometry B );
```

Beschreibung

Der `<|` Operator gibt `TRUE` zurück, wenn die Bounding Box der Geometrie A die Bounding Box der Geometrie B überlagert oder unterhalb liegt, oder präziser, überlagert oder NICHT oberhalb der Bounding der Geometrie B liegt.



This method supports Circular Strings and Curves



This function supports Polyhedral surfaces.



Note

Dieser Operand benützt jeden Index, der für die Geometrie zur Verfügung stellt.

Beispiele

```
SELECT tbl1.column1, tbl2.column1, tbl1.column2 <| tbl2.column2 AS overbelow
FROM
  ( VALUES
    (1, 'LINESTRING(6 0, 6 4)::geometry) AS tbl1,
  ( VALUES
    (2, 'LINESTRING(0 0, 3 3)::geometry),
    (3, 'LINESTRING(0 1, 0 5)::geometry),
    (4, 'LINESTRING(1 2, 4 6)::geometry) AS tbl2;
```

column1	column1	overbelow
1	2	f
1	3	t
1	4	t

(3 rows)

Siehe auch

`&&`, `|&>`, `&>`, `&<`

8.8.11 `&>`

`&>` — Gibt `TRUE` zurück, wenn die Bounding Box von A jene von B überlagert oder rechts davon liegt.

Synopsis

boolean `&>`(geometry A , geometry B);

Beschreibung

Der `&>` Operator gibt `TRUE` zurück, wenn die Bounding Box der Geometrie A die Bounding Box der Geometrie B überlagert oder rechts von ihr liegt, oder präziser, überlagert und NICHT links von der Bounding Box der Geometrie B liegt.



Note

Dieser Operand benützt jeden Index, der für die Geometrie zur Verfügung stellt.

Beispiele

```
SELECT tbl1.column1, tbl2.column1, tbl1.column2 &> tbl2.column2 AS overright
FROM
  ( VALUES
    (1, 'LINESTRING(1 2, 4 6)::geometry)) AS tbl1,
  ( VALUES
    (2, 'LINESTRING(0 0, 3 3)::geometry),
    (3, 'LINESTRING(0 1, 0 5)::geometry),
    (4, 'LINESTRING(6 0, 6 1)::geometry)) AS tbl2;

column1 | column1 | overright
-----+-----+-----
          1 |          2 | t
          1 |          3 | t
          1 |          4 | f
(3 rows)
```

Siehe auch

[&&](#), [|&>](#), [&<|](#), [&<](#)

8.8.12 <<

<< — Gibt TRUE zurück, wenn die Bounding Box von A zur Gänze links von der von B liegt.

Synopsis

boolean <<(geometry A , geometry B);

Beschreibung

Der << Operator gibt TRUE zurück, wenn die Bounding Box der Geometrie A zur Gänze links der Bounding Box der Geometrie B liegt.



Note

Dieser Operand benützt jeden Index, der für die Geometrie zur Verfügung stellt.

Beispiele

```
SELECT tbl1.column1, tbl2.column1, tbl1.column2 << tbl2.column2 AS left
FROM
  ( VALUES
    (1, 'LINESTRING (1 2, 1 5)::geometry)) AS tbl1,
  ( VALUES
    (2, 'LINESTRING (0 0, 4 3)::geometry),
    (3, 'LINESTRING (6 0, 6 5)::geometry),
    (4, 'LINESTRING (2 2, 5 6)::geometry)) AS tbl2;

column1 | column1 | left
-----+-----+-----
          1 |          2 | f
```

```

      1 |      3 | t
      1 |      4 | t
(3 rows)

```

Siehe auch

>>, |>>, <<|

8.8.13 <<|

<<| — Gibt TRUE zurück, wenn A's Bounding Box zur Gänze unterhalb von der von B liegt.

Synopsis

```
boolean <<|( geometry A , geometry B );
```

Beschreibung

Der <<| Operator gibt TRUE zurück, wenn die Bounding Box der Geometrie A zur Gänze unterhalb der Bounding Box von Geometrie B liegt.

**Note**

Dieser Operand benützt jeden Index, der für die Geometrie zur Verfügung stellt.

Beispiele

```

SELECT tbl1.column1, tbl2.column1, tbl1.column2 <<| tbl2.column2 AS below
FROM
  ( VALUES
    (1, 'LINESTRING (0 0, 4 3)::geometry)) AS tbl1,
  ( VALUES
    (2, 'LINESTRING (1 4, 1 7)::geometry),
    (3, 'LINESTRING (6 1, 6 5)::geometry),
    (4, 'LINESTRING (2 3, 5 6)::geometry)) AS tbl2;

```

```

column1 | column1 | below
-----+-----+-----
      1 |      2 | t
      1 |      3 | f
      1 |      4 | f
(3 rows)

```

Siehe auch

<<, >>, |>>

8.8.14 =

= — Returns TRUE if the coordinates and coordinate order geometry/geography A are the same as the coordinates and coordinate order of geometry/geography B.

Synopsis

```
boolean =( geometry A , geometry B );
boolean =( geography A , geography B );
```

Beschreibung

The = operator returns TRUE if the coordinates and coordinate order geometry/geography A are the same as the coordinates and coordinate order of geometry/geography B. PostgreSQL uses the =, <, and > operators defined for geometries to perform internal orderings and comparison of geometries (ie. in a GROUP BY or ORDER BY clause).



Note

Only geometry/geography that are exactly equal in all respects, with the same coordinates, in the same order, are considered equal by this operator. For "spatial equality", that ignores things like coordinate order, and can detect features that cover the same spatial area with different representations, use [ST_OrderingEquals](#) or [ST_Equals](#)



Caution

This operand will NOT make use of any indexes that may be available on the geometries. For an index assisted exact equality test, combine = with &&.

Changed: 2.4.0, in prior versions this was bounding box equality not a geometric equality. If you need bounding box equality, use ~= instead.



This method supports Circular Strings and Curves



This function supports Polyhedral surfaces.

Beispiele

```
SELECT 'LINESTRING(0 0, 0 1, 1 0)::geometry = 'LINESTRING(1 1, 0 0)::geometry';
?column?
```

```
-----
f
(1 row)
```

```
SELECT ST_AsText(column1)
FROM ( VALUES
      ('LINESTRING(0 0, 1 1)::geometry),
      ('LINESTRING(1 1, 0 0)::geometry)) AS foo;
      st_astext
```

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

```
-- Note: the GROUP BY uses the "=" to compare for geometry equivalency.
```

```
SELECT ST_AsText(column1)
FROM ( VALUES
      ('LINESTRING(0 0, 1 1)::geometry),
      ('LINESTRING(1 1, 0 0)::geometry)) AS foo
GROUP BY column1;
      st_astext
```

```

-----
LINESTRING(0 0,1 1)
LINESTRING(1 1,0 0)
(2 rows)

-- In versions prior to 2.0, this used to return true --
SELECT ST_GeomFromText('POINT(1707296.37 4820536.77)') =
       ST_GeomFromText('POINT(1707296.27 4820536.87)') As pt_intersect;

--pt_intersect --
f

```

Siehe auch

[ST_Equals](#), [ST_OrderingEquals](#), [~=](#)

8.8.15 >>

>> — Gibt TRUE zurück, wenn A's bounding box zur Gänze rechts von der von B liegt.

Synopsis

```
boolean >>( geometry A , geometry B );
```

Beschreibung

Der >> Operator gibt TRUE zurück, wenn die Bounding Box von Geometrie A zur Gänze rechts der Bounding Box von Geometrie B liegt.



Note

Dieser Operand benützt jeden Index, der für die Geometrie zur Verfügung stellt.

Beispiele

```

SELECT tbl1.column1, tbl2.column1, tbl1.column2 >> tbl2.column2 AS right
FROM
  ( VALUES
    (1, 'LINESTRING (2 3, 5 6)::geometry) AS tbl1,
  ( VALUES
    (2, 'LINESTRING (1 4, 1 7)::geometry),
    (3, 'LINESTRING (6 1, 6 5)::geometry),
    (4, 'LINESTRING (0 0, 4 3)::geometry) AS tbl2;

column1 | column1 | right
-----+-----+-----
        1 |        2 | t
        1 |        3 | f
        1 |        4 | f
(3 rows)

```

Siehe auch

<<, >>, <<|

8.8.16 @

@ — 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 vollständig in der Bounding Box der Geometrie B enthalten ist.

**Note**

Dieser Operand benützt jeden Index, der für die Geometrie zur Verfügung stellt.

Beispiele

```
SELECT tbl1.column1, tbl2.column1, tbl1.column2 @ tbl2.column2 AS contained
FROM
  ( VALUES
    (1, 'LINESTRING (1 1, 3 3)::geometry) AS tbl1,
  ( VALUES
    (2, 'LINESTRING (0 0, 4 4)::geometry),
    (3, 'LINESTRING (2 2, 4 4)::geometry),
    (4, 'LINESTRING (1 1, 3 3)::geometry) AS tbl2;
```

column1	column1	contained
1	2	t
1	3	f
1	4	t

(3 rows)

Siehe auch

&, &&

8.8.17 @(geometry,box2df)

@(geometry,box2df) — Gibt TRUE zurück, wenn die 2D Bounding Box einer Geometrie in einer 2D float precision Bounding Box (BOX2DF) enthalten ist.

Synopsis

```
boolean @( geometry A , box2df B );
```

Beschreibung

Der @ Operator gibt TRUE zurück, wenn die 2D Bounding Box der Geometrie A in der 2D Bounding Box der Geometrie B , unter Benutzung von float precision, enthalten ist. D.h.: wenn B eine (double precision) box2d ist, wird diese intern in eine float precision 2D bounding box (BOX2DF) übersetzt.



Note

Dieser Operand ist eher für die interne Nutzung durch BRIN Indizes, als durch die Anwender, gedacht.

Verfügbarkeit: Mit 2.3.0 wurde die Unterstützung von Block Range INdices (BRIN) eingeführt. Erfordert PostgreSQL 9.5+.



This method supports Circular Strings and Curves



This function supports Polyhedral surfaces.

Beispiele

```
SELECT ST_Buffer(ST_GeomFromText('POINT(2 2)'), 1) @ ST_MakeBox2D(ST_MakePoint(0,0), ↔
    ST_MakePoint(5,5)) AS is_contained;
```

```
is_contained
-----
t
(1 row)
```

Siehe auch

[&&\(geometry,box2df\)](#), [&&\(box2df,geometry\)](#), [&&\(box2df,box2df\)](#), [~\(geometry,box2df\)](#), [~\(box2df,geometry\)](#), [~\(box2df,box2df\)](#), [@\(box2df,geometry\)](#), [@\(box2df,box2df\)](#)

8.8.18 @(box2df,geometry)

@(box2df,geometry) — Gibt TRUE zurück, wenn eine 2D float precision bounding box (BOX2DF) in der 2D Bounding Box einer Geometrie enthalten ist..

Synopsis

boolean @(box2df A , geometry B);

Beschreibung

Der @ Operator gibt TRUE zurück, wenn die 2D bounding box A in der 2D bounding box der Geometrie B, unter Verwendung von float precision, enthalten ist. D.h.: wenn B eine (double precision) box2d ist, wird diese intern in eine float precision 2D bounding box (BOX2DF) umgewandelt



Note

Dieser Operand ist eher für die interne Nutzung durch BRIN Indizes, als durch die Anwender, gedacht.

Verfügbarkeit: Mit 2.3.0 wurde die Unterstützung von Block Range INdices (BRIN) eingeführt. Erfordert PostgreSQL 9.5+.



This method supports Circular Strings and Curves



This function supports Polyhedral surfaces.

Beispiele

```
SELECT ST_MakeBox2D(ST_MakePoint(2,2), ST_MakePoint(3,3)) @ ST_Buffer(ST_GeomFromText(' ↵
POINT(1 1)'), 10) AS is_contained;
```

```
is_contained
-----
t
(1 row)
```

Siehe auch

[&&\(geometry,box2df\)](#), [&&\(box2df,geometry\)](#), [&&\(box2df,box2df\)](#), [~\(geometry,box2df\)](#), [~\(box2df,geometry\)](#), [~\(box2df,box2df\)](#), [@\(geometry,box2df\)](#), [@\(box2df,box2df\)](#)

8.8.19 @(box2df,box2df)

@(box2df,box2df) — Gibt TRUE zurück, wenn eine 2D float precision bounding box (BOX2DF) innerhalb einer anderen 2D float precision bounding box enthalten ist.

Synopsis

```
boolean @( box2df A , box2df B );
```

Beschreibung

Der @ Operator gibt TRUE zurück, wenn die 2D bounding box A innerhalb der 2D bounding box B, unter Verwendung von float precision, enthalten ist. D.h.: wenn A (oder B) eine (double precision) box2d ist, wird diese intern in eine float precision 2D bounding box (BOX2DF) umgewandelt.



Note

Dieser Operand ist eher für die interne Nutzung durch BRIN Indizes, als durch die Anwender, gedacht.

Verfügbarkeit: Mit 2.3.0 wurde die Unterstützung von Block Range INdices (BRIN) eingeführt. Erfordert PostgreSQL 9.5+.



This method supports Circular Strings and Curves



This function supports Polyhedral surfaces.

Beispiele

```
SELECT ST_MakeBox2D(ST_MakePoint(2,2), ST_MakePoint(3,3)) @ ST_MakeBox2D(ST_MakePoint(0,0), ←
    ST_MakePoint(5,5)) AS is_contained;

is_contained
-----
t
(1 row)
```

Siehe auch

[&&\(geometry,box2df\)](#), [&&\(box2df,geometry\)](#), [&&\(box2df,box2df\)](#), [~\(geometry,box2df\)](#), [~\(box2df,geometry\)](#), [~\(box2df,box2df\)](#), [@\(geometry,box2df\)](#), [@\(box2df,geometry\)](#)

8.8.20 |&>

|&> — Gibt TRUE zurück, wenn A's bounding box diejenige von B überlagert oder oberhalb von B liegt.

Synopsis

boolean |&>(geometry A , geometry B);

Beschreibung

Der |&> Operator gibt TRUE zurück, wenn die bounding box der Geometrie A die bounding box der Geometrie B überlagert oder oberhalb liegt, oder präziser, überlagert oder NICHT unterhalb der Bounding Box der Geometrie B liegt.



Note

Dieser Operand benützt jeden Index, der für die Geometrie zur Verfügung stellt.

Beispiele

```
SELECT tbl1.column1, tbl2.column1, tbl1.column2 |&> tbl2.column2 AS overabove
FROM
  ( VALUES
    (1, 'LINESTRING(6 0, 6 4)::geometry) ) AS tbl1,
  ( VALUES
    (2, 'LINESTRING(0 0, 3 3)::geometry),
    (3, 'LINESTRING(0 1, 0 5)::geometry),
    (4, 'LINESTRING(1 2, 4 6)::geometry) ) AS tbl2;

column1 | column1 | overabove
-----+-----+-----
        1 |         2 | t
        1 |         3 | f
        1 |         4 | f
(3 rows)
```


Siehe auch**&&, &>, &<|, &<****8.8.21 |>>****|>>** — Gibt TRUE zurück, wenn A's bounding box is zur Gänze oberhalb der von B liegt.**Synopsis**boolean **|>>**(geometry A , geometry B);**Beschreibung**Der Operator **|>>** gibt TRUE zurück, wenn die Bounding Box der Geometrie A zur Gänze oberhalb der Bounding Box von Geometrie B liegt.**Note**

Dieser Operand benützt jeden Index, der für die Geometrie zur Verfügung stellt.

Beispiele

```
SELECT tbl1.column1, tbl2.column1, tbl1.column2 |>> tbl2.column2 AS above
FROM
  ( VALUES
    (1, 'LINESTRING (1 4, 1 7)::geometry') AS tbl1,
    ( VALUES
      (2, 'LINESTRING (0 0, 4 2)::geometry),
      (3, 'LINESTRING (6 1, 6 5)::geometry),
      (4, 'LINESTRING (2 3, 5 6)::geometry') AS tbl2;
```

column1	column1	above
1	2	t
1	3	f
1	4	f

(3 rows)

Siehe auch**<<, >>, <<|****8.8.22 ~****~** — Gibt TRUE zurück, wenn A's bounding box die von B enthält.**Synopsis**boolean **~**(geometry A , geometry B);

Beschreibung

Der `~` Operator gibt `TRUE` zurück, wenn die bounding box der Geometrie A zur Gänze die bounding box der Geometrie B enthält.



Note

Dieser Operand benützt jeden Index, der für die Geometrie zur Verfügung stellt.

Beispiele

```
SELECT tbl1.column1, tbl2.column1, tbl1.column2 ~ tbl2.column2 AS contains
FROM
  ( VALUES
    (1, 'LINESTRING (0 0, 3 3)::geometry) AS tbl1,
  ( VALUES
    (2, 'LINESTRING (0 0, 4 4)::geometry),
    (3, 'LINESTRING (1 1, 2 2)::geometry),
    (4, 'LINESTRING (0 0, 3 3)::geometry) AS tbl2;
```

column1	column1	contains
1	2	f
1	3	t
1	4	t

(3 rows)

Siehe auch

[@](#), [&&](#)

8.8.23 ~(geometry,box2df)

`~(geometry,box2df)` — Gibt `TRUE` zurück, wenn die 2D bounding box einer Geometrie eine 2D float precision bounding box (GIDX) enthält.

Synopsis

```
boolean ~( geometry A , box2df B );
```

Beschreibung

Der `~` Operator gibt `TRUE` zurück, wenn die 2D bounding box einer Geometrie A die 2D bounding box B, unter Verwendung von float precision, enthält. D.h.: wenn B eine (double precision) box2d ist, wird diese intern in eine float precision 2D bounding box (BOX2DF) übersetzt



Note

Dieser Operand ist eher für die interne Nutzung durch BRIN Indizes, als durch die Anwender, gedacht.

Verfügbarkeit: Mit 2.3.0 wurde die Unterstützung von Block Range INdices (BRIN) eingeführt. Erfordert PostgreSQL 9.5+.



This method supports Circular Strings and Curves



This function supports Polyhedral surfaces.

Beispiele

```
SELECT ST_Buffer(ST_GeomFromText('POINT(1 1)'), 10) ~ ST_MakeBox2D(ST_MakePoint(0,0), ↵
    ST_MakePoint(2,2)) AS contains;

contains
-----
t
(1 row)
```

Siehe auch

[&&\(geometry,box2df\)](#), [&&\(box2df,geometry\)](#), [&&\(box2df,box2df\)](#), [~\(box2df,geometry\)](#), [~\(box2df,box2df\)](#), [@\(geometry,box2df\)](#), [@\(box2df,geometry\)](#), [@\(box2df,box2df\)](#)

8.8.24 ~(box2df,geometry)

`~(box2df,geometry)` — Gibt TRUE zurück, wenn eine 2D float precision bounding box (BOX2DF) die 2D Bounding Box einer Geometrie enthält.

Synopsis

boolean `~( box2df A , geometry B );`

Beschreibung

Der `~` Operator gibt TRUE zurück, wenn die 2D bounding box A die Bounding Box der Geometrie B, unter Verwendung von float precision, enthält. D.h.: wenn A eine (double precision) box2d ist, wird diese intern in eine float precision 2D bounding box (BOX2DF) umgewandelt.



Note

Dieser Operand ist eher für die interne Nutzung durch BRIN Indizes, als durch die Anwender, gedacht.

Verfügbarkeit: Mit 2.3.0 wurde die Unterstützung von Block Range INdices (BRIN) eingeführt. Erfordert PostgreSQL 9.5+.



This method supports Circular Strings and Curves



This function supports Polyhedral surfaces.

Beispiele

```
SELECT ST_MakeBox2D(ST_MakePoint(0,0), ST_MakePoint(5,5)) ~ ST_Buffer(ST_GeomFromText(' ↵
    POINT(2 2)'), 1) AS contains;

contains
-----
t
(1 row)
```

Siehe auch

[&&\(geometry,box2df\)](#), [&&\(box2df,geometry\)](#), [&&\(box2df,box2df\)](#), [~\(geometry,box2df\)](#), [~\(box2df,box2df\)](#), [@\(geometry,box2df\)](#), [@\(box2df,geometry\)](#), [@\(box2df,box2df\)](#)

8.8.25 ~(box2df,box2df)

`~(box2df,box2df)` — Gibt TRUE zurück, wenn eine 2D float precision bounding box (BOX2DF) eine andere 2D float precision bounding box (BOX2DF) enthält.

Synopsis

boolean `~( box2df A , box2df B );`

Beschreibung

Der `~` Operator gibt TRUE zurück, wenn die 2D bounding box A die 2D bounding box B, unter Verwendung von float precision, enthält. D.h.: wenn A eine (double precision) box2d ist, wird diese intern in eine float precision 2D bounding box (BOX2DF) umgewandelt



Note

Dieser Operand ist eher für die interne Nutzung durch BRIN Indizes, als durch die Anwender, gedacht.

Verfügbarkeit: Mit 2.3.0 wurde die Unterstützung von Block Range INdices (BRIN) eingeführt. Erfordert PostgreSQL 9.5+.



This method supports Circular Strings and Curves



This function supports Polyhedral surfaces.

Beispiele

```
SELECT ST_MakeBox2D(ST_MakePoint(0,0), ST_MakePoint(5,5)) ~ ST_MakeBox2D(ST_MakePoint(2,2), ↵
    ST_MakePoint(3,3)) AS contains;

contains
-----
t
(1 row)
```

Siehe auch

[&&\(geometry,box2df\)](#), [&&\(box2df,geometry\)](#), [&&\(box2df,box2df\)](#), [~\(geometry,box2df\)](#), [~\(box2df,geometry\)](#), [@\(geometry,box2df\)](#), [@\(box2df,geometry\)](#), [@\(box2df,box2df\)](#)

8.8.26 ~ =

`~ =` — Gibt `TRUE` zurück, wenn die bounding box von A ident mit jener von B ist.

Synopsis

`boolean ~=( geometry A , geometry B );`

Beschreibung

Der `~ =` Operator gibt `TRUE` zurück, wenn die bounding box der Geometrie/Geographie A ident mit der bounding box der Geometrie/Geographie B ist.

**Note**

Dieser Operand benützt jeden Index, der für die Geometrie zur Verfügung stellt.

Verfügbarkeit: 1.5.0 "Verhaltensänderung"



This function supports Polyhedral surfaces.

**Warning**

This operator has changed behavior in PostGIS 1.5 from testing for actual geometric equality to only checking for bounding box equality. To complicate things it also depends on if you have done a hard or soft upgrade which behavior your database has. To find out which behavior your database has you can run the query below. To check for true equality use [ST_OrderingEquals](#) or [ST_Equals](#).

Beispiele

```
select 'LINESTRING(0 0, 1 1)::geometry ~ = 'LINESTRING(0 1, 1 0)::geometry as equality;
equality |
-----+
t       |
```

Siehe auch

[ST_Equals](#), [ST_OrderingEquals](#), [=](#)

8.8.27 <->

`<->` — Gibt die 2D Entfernung zwischen A und B zurück.

Synopsis

```
double precision <->( geometry A , geometry B );
double precision <->( geography A , geography B );
```

Beschreibung

Der <-> Operator gibt die 2D Entfernung zwischen zwei Geometrien zurück. Wird er in einer "ORDER BY" Klausel verwendet, so liefert er Index-unterstützte nearest-neighbor Ergebnismengen. PostgreSQL Versionen unter 9.5 geben jedoch lediglich die Entfernung der Centroide der bounding boxes zurück, während PostgreSQL 9.5+ mittels KNN-Methode die tatsächliche Entfernung zwischen den Geometrien, bei geographischen Koordinaten die Entfernung auf der Späre, wiedergibt.



Note

Dieser Operand verwendet 2D GiST Indizes, falls diese für die Geometrien vorhanden sind. Er unterscheidet sich insofern von anderen Operatoren, die räumliche Indizes verwenden, indem der räumliche Index nur dann verwendet wird, wenn sich der Operator in einer ORDER BY Klausel befindet.



Note

Der Index kommt nur zum Tragen, wenn eine der Geometrien eine Konstante ist (sich nicht in einer Subquery/CTE befindet). Z.B. 'SRID=3005;POINT(1011102 450541)::geometry' und nicht a.geom

Siehe [OpenGeo workshop: Nearest-Neighbour Searching](#) für ein praxisbezogenes Anwendungsbeispiel.

Verbesserung: 2.2.0 -- Echtes KNN ("K nearest neighbor") Verhalten für Geometrie und Geographie ab PostgreSQL 9.5+. Beachten Sie bitte, das KNN für Geographie auf der Späre und nicht auf dem Sphäroid beruht. Für PostgreSQL 9.4 und darunter, wird die Berechnung nur auf Basis des Centroids der Box unterstützt.

Änderung: 2.2.0 -- Da für Anwender von PostgreSQL 9.5 der alte hybride Syntax langsamer sein kann, möchten sie diesen Hack eventuell loswerden, falls der Code nur auf PostGIS 2.2+ 9.5+ läuft. Siehe die unteren Beispiele.

Verfügbarkeit: 2.0.0 -- Weak KNN liefert nearest neighbors, welche sich auf die Entfernung der Centroide der Geometrien, anstatt auf den tatsächlichen Entfernungen, stützen. Genaue Ergebnisse für Punkte, ungenau für alle anderen Geometrietypen. Verfügbar ab PostgreSQL 9.1+.

Beispiele

```
SELECT ST_Distance(geom, 'SRID=3005;POINT(1011102 450541)::geometry') as d,edabbr, vaabbr
FROM va2005
ORDER BY d limit 10;
```

d	edabbr	vaabbr
0	ALQ	128
5541.57712511724	ALQ	129A
5579.67450712005	ALQ	001
6083.4207708641	ALQ	131
7691.2205404848	ALQ	003
7900.75451037313	ALQ	122
8694.20710669982	ALQ	129B
9564.24289057111	ALQ	130
12089.665931705	ALQ	127
18472.5531479404	ALQ	002

(10 rows)

Then the KNN raw answer:

```
SELECT st_distance(geom, 'SRID=3005;POINT(1011102 450541)'::geometry) as d,edabbr, vaabbr
FROM va2005
ORDER BY geom <-> 'SRID=3005;POINT(1011102 450541)'::geometry limit 10;
```

d	edabbr	vaabbr
0	ALQ	128
5541.57712511724	ALQ	129A
5579.67450712005	ALQ	001
6083.4207708641	ALQ	131
7691.2205404848	ALQ	003
7900.75451037313	ALQ	122
8694.20710669982	ALQ	129B
9564.24289057111	ALQ	130
12089.665931705	ALQ	127
18472.5531479404	ALQ	002

(10 rows)

Wenn Sie "EXPLAIN ANALYZE" an den zwei Abfragen ausführen, sollte eine Performance Verbesserung im Ausmaß von einer Sekunde auftreten.

Anwender von PostgreSQL < 9.5 können eine hybride Abfrage erstellen, um die echten nearest neighbors aufzufinden. Zuerst eine CTE-Abfrage, welche die Index-unterstützten KNN-Methode anwendet, dann eine exakte Abfrage um eine korrekte Sortierung zu erhalten:

```
WITH index_query AS (
  SELECT ST_Distance(geom, 'SRID=3005;POINT(1011102 450541)'::geometry) as d,edabbr, vaabbr
  FROM va2005
  ORDER BY geom <-> 'SRID=3005;POINT(1011102 450541)'::geometry LIMIT 100)
SELECT *
  FROM index_query
ORDER BY d limit 10;
```

d	edabbr	vaabbr
0	ALQ	128
5541.57712511724	ALQ	129A
5579.67450712005	ALQ	001
6083.4207708641	ALQ	131
7691.2205404848	ALQ	003
7900.75451037313	ALQ	122
8694.20710669982	ALQ	129B
9564.24289057111	ALQ	130
12089.665931705	ALQ	127
18472.5531479404	ALQ	002

(10 rows)

Siehe auch

[ST_DWithin](#), [ST_Distance](#), [<#>](#)

8.8.28 ||

|| — Gibt die Entfernung zwischen den Trajektorien A und B, am Ort der dichtesten Annäherung, an.

Synopsis

double precision || (geometry A , geometry B);

Beschreibung

Der `|=|` Operator gibt die 3D Entfernung zwischen zwei Trajektorien (Siehe [ST_IsValidTrajectory](#)). Dieser entspricht [ST_DistanceCPA](#), da es sich jedoch um einen Operator handelt, kann dieser für nearest neighbor searches mittels eines N-dimensionalen Index verwendet werden (verlangt PostgreSQL 9.5.0 oder höher).



Note

Dieser Operand verwendet die ND GiST Indizes, welche für Geometrien vorhanden sein können. Er unterscheidet sich insofern von anderen Operatoren, die ebenfalls räumliche Indizes verwenden, als der räumliche Index nur dann angewandt wird, wenn sich der Operand in einer ORDER BY Klausel befindet.



Note

Der Index kommt nur zum Tragen, wenn eine der Geometrien eine Konstante ist (sich nicht in einer Subquery/CTE befindet). Z.B. `'SRID=3005;LINESTRINGM(0 0 0,0 0 1)>::geometry` und nicht `a.geom`

Verfügbarkeit: 2.2.0. Index-unterstützt steht erst ab PostgreSQL 9.5+ zur Verfügung.

Beispiele

```
-- Save a literal query trajectory in a psql variable...
\set qt 'ST_AddMeasure(ST_MakeLine(ST_MakePointM(-350,300,0),ST_MakePointM(-410,490,0)) ↔
,10,20) '
-- Run the query !
SELECT track_id, dist FROM (
  SELECT track_id, ST_DistanceCPA(tr,:qt) dist
  FROM trajectories
  ORDER BY tr |=| :qt
  LIMIT 5
) foo;
 track_id      dist
-----+-----
      395 | 0.576496831518066
      380 |  5.06797130410151
      390 |  7.72262293958322
      385 |  9.8004461358071
      405 | 10.9534397988433
(5 rows)
```

Siehe auch

[ST_DistanceCPA](#), [ST_ClosestPointOfApproach](#), [ST_IsValidTrajectory](#)

8.8.29 <#>

<#> — Gibt die 2D Entfernung zwischen den Bounding Boxes von A und B zurück

Synopsis

double precision <#>(geometry A , geometry B);

Beschreibung

Der `<#>` Operator gibt die Entfernung zwischen zwei floating point bounding boxes zurück, wobei diese eventuell vom räumlichen Index ausgelesen wird (PostgreSQL 9.1+ vorausgesetzt). Praktikabel falls man eine nearest neighbor Abfrage **approximate** nach der Entfernung sortieren will.



Note

Dieser Operand verwendet sämtliche Indizes, welche für die Geometrien vorhanden sind. Er unterscheidet sich insofern von anderen Operatoren, welche ebenfalls räumliche Indizes verwenden, als der räumliche Index nur dann verwendet wird, falls sich der Operand in einer ORDER BY Klausel befindet.



Note

Der Index kommt nur zum Tragen, wenn eine der Geometrien eine Konstante ist; z.B.: ORDER BY (ST_GeomFromText('POINT(1 2)') `<#>` geom) anstatt g1.geom `<#>`.

Verfügbarkeit: 2.0.0 -- KNN steht erst ab PostgreSQL 9.1+ zur Verfügung

Beispiele

```
SELECT *
FROM (
SELECT b.tlid, b.mtfcc,
       b.geom <#
> ST_GeomFromText('LINESTRING(746149 2948672,745954 2948576,
                             745787 2948499,745740 2948468,745712 2948438,
                             745690 2948384,745677 2948319)',2249) As b_dist,
       ST_Distance(b.geom, ST_GeomFromText('LINESTRING(746149 2948672,745954 2948576,
                             745787 2948499,745740 2948468,745712 2948438,
                             745690 2948384,745677 2948319)',2249)) As act_dist
FROM bos_roads As b
ORDER BY b_dist, b.tlid
LIMIT 100) As foo
ORDER BY act_dist, tlid LIMIT 10;
```

tlid	mtfcc	b_dist	act_dist
85732027	S1400	0	0
85732029	S1400	0	0
85732031	S1400	0	0
85734335	S1400	0	0
85736037	S1400	0	0
624683742	S1400	0	128.528874268666
85719343	S1400	260.839270432962	260.839270432962
85741826	S1400	164.759294123275	260.839270432962
85732032	S1400	277.75	311.830282365264
85735592	S1400	222.25	311.830282365264

(10 rows)

Siehe auch

[ST_DWithin](#), [ST_Distance](#), [<->](#)

8.8.30 <<->>

<<->> — Gibt die n-D Entfernung zwischen den geometrischen Schwerpunkten der Begrenzungsrechtecke/Bounding Boxes von A und B zurück.

Synopsis

double precision <<->>(geometry A , geometry B);

Beschreibung

Der <<->> Operator gibt die n-D (euklidische) Entfernung zwischen den geometrischen Schwerpunkten der Begrenzungsrechtecke zweier Geometrien zurück. Praktikabel für nearest neighbor **approximate** distance ordering.



Note

Dieser Operator verwendet n-D GiST Indizes, falls diese für die Geometrien vorhanden sind. Er unterscheidet sich insofern von anderen Operatoren, die räumliche Indizes verwenden, indem der räumliche Index nur dann verwendet wird, wenn sich der Operator in einer ORDER BY Klausel befindet.



Note

Der Index kommt nur zum Tragen, wenn eine der Geometrien eine Konstante ist (sich nicht in einer Subquery/CTE befindet). Z.B. 'SRID=3005;POINT(1011102 450541)::geometry und nicht a.geom

Verfügbarkeit: 2.2.0 -- KNN steht erst ab PostgreSQL 9.1+ zur Verfügung.

Siehe auch

<<#>>, <<->>

8.8.31 <<#>>

<<#>> — Gibt die n-D Entfernung zwischen den Bounding Boxes von A und B zurück.

Synopsis

double precision <<#>>(geometry A , geometry B);

Beschreibung

Der <<#>> Operator gibt die Entfernung zwischen zwei floating point bounding boxes zurück, wobei diese eventuell vom räumlichen Index ausgelesen wird (PostgreSQL 9.1+ vorausgesetzt). Praktikabel falls man eine nearest neighbor Abfrage nach der Entfernung sortieren will / **approximate** distance ordering.



Note

Dieser Operand verwendet sämtliche Indizes, welche für die Geometrien vorhanden sind. Er unterscheidet sich insofern von anderen Operatoren, welche ebenfalls räumliche Indizes verwenden, als der räumliche Index nur dann verwendet wird, falls sich der Operand in einer ORDER BY Klausel befindet.

**Note**

Der Index kommt nur zum Tragen, wenn eine der Geometrien eine Konstante ist; z.B.: ORDER BY (ST_GeomFromText('POINT(1 2)') <<#>> geom) anstatt g1.geom <<#>>.

Verfügbarkeit: 2.2.0 -- KNN steht erst ab PostgreSQL 9.1+ zur Verfügung.

Siehe auch

<<->, <#>

8.9 Räumliche Beziehungen und Messungen

8.9.1 ST_3DClosestPoint

ST_3DClosestPoint — Returns the 3-dimensional point on g1 that is closest to g2. This is the first point of the 3D shortest line.

Synopsis

geometry **ST_3DClosestPoint**(geometry g1, geometry g2);

Beschreibung

Returns the 3-dimensional point on g1 that is closest to g2. This is the first point of the 3D shortest line. The 3D length of the 3D shortest line is the 3D distance.



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



This function supports Polyhedral surfaces.

Verfügbarkeit: 2.0.0

Changed: 2.2.0 - if 2 2D geometries are input, a 2D point is returned (instead of old behavior assuming 0 for missing Z). In case of 2D and 3D, Z is no longer assumed to be 0 for missing Z.

Beispiele

linestring and point -- both 3d and 2d closest point

```
SELECT ST_AsEWKT(ST_3DClosestPoint(line,pt)) AS cp3d_line_pt,
       ST_AsEWKT(ST_ClosestPoint(line,pt)) As cp2d_line_pt
FROM (SELECT 'POINT(100 100 30)'::geometry As pt,
            'LINESTRING (20 80 20, 98 190 1, 110 180 3, 50 75 1000)'::
            geometry As line
      ) As foo;
```

cp3d_line_pt		
cp2d_line_pt		
POINT(54.6993798867619 128.935022917228 11.5475869506606)		POINT(73.0769230769231 115.384615384615)

linestring and multipoint -- both 3d and 2d closest point

```
SELECT ST_AsEWKT(ST_3DClosestPoint(line,pt)) AS cp3d_line_pt,
       ST_AsEWKT(ST_ClosestPoint(line,pt)) As cp2d_line_pt
FROM (SELECT 'MULTIPOINT(100 100 30, 50 74 1000) '::geometry As pt,
            'LINESTRING (20 80 20, 98 190 1, 110 180 3, 50 75 900) '::
geometry As line
      ) As foo;
```

cp3d_line_pt	cp2d_line_pt
POINT(54.6993798867619 128.935022917228 11.5475869506606)	POINT(50 75)

Multilinestring and polygon both 3d and 2d closest point

```
SELECT ST_AsEWKT(ST_3DClosestPoint(poly, mline)) As cp3d,
       ST_AsEWKT(ST_ClosestPoint(poly, mline)) As cp2d
FROM (SELECT ST_GeomFromEWKT('POLYGON((175 150 5, 20 40 5, 35 45 5, 50 60 5,
100 100 5, 175 150 5))') As poly,
       ST_GeomFromEWKT('MULTILINESTRING((175 155 2, 20 40 20, 50 60 -2, 125
100 1, 175 155 1),
       (1 10 2, 5 20 1))') As mline ) As foo;
```

cp3d	cp2d
POINT(39.993580415989 54.1889925532825 5)	POINT(20 40)

Siehe auch

[ST_AsEWKT](#), [ST_ClosestPoint](#), [ST_3DDistance](#), [ST_3DShortestLine](#)

8.9.2 ST_3DDistance

ST_3DDistance — For geometry type Returns the 3-dimensional cartesian minimum distance (based on spatial ref) between two geometries in projected units.

Synopsis

float **ST_3DDistance**(geometry g1, geometry g2);

Beschreibung

For geometry type returns the 3-dimensional minimum cartesian distance between two geometries in projected units (spatial ref units).

-  This function supports 3d and will not drop the z-index.
-  This function supports Polyhedral surfaces.
-  This method implements the SQL/MM specification. SQL-MM ?
-  This method is also provided by SFCGAL backend.

Verfügbarkeit: 2.0.0

Changed: 2.2.0 - In case of 2D and 3D, Z is no longer assumed to be 0 for missing Z.

Beispiele

```
-- Geometry example - units in meters (SRID: 2163 US National Atlas Equal area) (3D point ←
  and line compared 2D point and line)
-- Note: currently no vertical datum support so Z is not transformed and assumed to be same ←
  units as final.
SELECT ST_3DDistance(
    ST_Transform(ST_GeomFromEWKT('SRID=4326;POINT(-72.1235 42.3521 4)') ←
      ,2163),
    ST_Transform(ST_GeomFromEWKT('SRID=4326;LINESTRING(-72.1260 42.45 ←
      15, -72.123 42.1546 20)'),2163)
  ) As dist_3d,
  ST_Distance(
    ST_Transform(ST_GeomFromText('POINT(-72.1235 42.3521)',4326),2163),
    ST_Transform(ST_GeomFromText('LINESTRING(-72.1260 42.45, -72.123 ←
      42.1546)', 4326),2163)
  ) As dist_2d;

    dist_3d      |      dist_2d
-----+-----
127.295059324629 | 126.66425605671
```

```
-- Multilinestring and polygon both 3d and 2d distance
-- Same example as 3D closest point example
SELECT ST_3DDistance(poly, mline) As dist3d,
  ST_Distance(poly, mline) As dist2d
  FROM (SELECT 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;

    dist3d      | dist2d
-----+-----
0.716635696066337 | 0
```

Siehe auch

[ST_Distance](#), [ST_3DClosestPoint](#), [ST_3DDWithin](#), [ST_3DMaxDistance](#), [ST_3DShortestLine](#), [ST_Transform](#)

8.9.3 ST_3DDWithin

ST_3DDWithin — For 3d (z) geometry type Returns true if two geometries 3d distance is within number of units.

Synopsis

boolean **ST_3DDWithin**(geometry g1, geometry g2, double precision distance_of_srid);

Beschreibung

For geometry type returns true if the 3d distance between two objects is within distance_of_srid specified projected units (spatial ref units).



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



This function supports Polyhedral surfaces.



This method implements the SQL/MM specification. SQL-MM ?

Verfügbarkeit: 2.0.0

Beispiele

```
-- Geometry example - units in meters (SRID: 2163 US National Atlas Equal area) (3D point  ←
    and line compared 2D point and line)
-- Note: currently no vertical datum support so Z is not transformed and assumed to be same  ←
    units as final.
SELECT ST_3DDWithin(
    ST_Transform(ST_GeomFromEWKT('SRID=4326;POINT(-72.1235 42.3521 4)')  ←
        ,2163),
    ST_Transform(ST_GeomFromEWKT('SRID=4326;LINESTRING(-72.1260 42.45  ←
        15, -72.123 42.1546 20)'),2163),
    126.8
    ) As within_dist_3d,
ST_DWithin(
    ST_Transform(ST_GeomFromEWKT('SRID=4326;POINT(-72.1235 42.3521 4)')  ←
        ,2163),
    ST_Transform(ST_GeomFromEWKT('SRID=4326;LINESTRING(-72.1260 42.45  ←
        15, -72.123 42.1546 20)'),2163),
    126.8
    ) As within_dist_2d;

within_dist_3d | within_dist_2d
-----+-----
f              | t
```

Siehe auch

[ST_3DDistance](#), [ST_Distance](#), [ST_DWithin](#), [ST_3DMaxDistance](#), [ST_Transform](#)

8.9.4 ST_3DDFullyWithin

ST_3DDFullyWithin — Returns true if all of the 3D geometries are within the specified distance of one another.

Synopsis

boolean **ST_3DDFullyWithin**(geometry g1, geometry g2, double precision distance);

Beschreibung

Returns true if the 3D geometries are fully within the specified distance of one another. The distance is specified in units defined by the spatial reference system of the geometries. For this function to make sense, the source geometries must both be of the same coordinate projection, having the same SRID.



Note

This function call will automatically include a bounding box comparison that will make use of any indexes that are available on the geometries.

Verfügbarkeit: 2.0.0



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



This function supports Polyhedral surfaces.

Beispiele

```
-- This compares the difference between fully within and distance within as well
-- as the distance fully within for the 2D footprint of the line/point vs.  ←
-- the 3d fully within
SELECT ST_3DDFullyWithin(geom_a, geom_b, 10) as D3DFullyWithin10,  ←
       ST_3DDWithin(geom_a, geom_b, 10) as D3DWithin10,
       ST_DFullyWithin(geom_a, geom_b, 20) as D2DFullyWithin20,
       ST_3DDFullyWithin(geom_a, geom_b, 20) as D3DFullyWithin20 from
       (select ST_GeomFromEWKT('POINT(1 1 2)') as geom_a,
        ST_GeomFromEWKT('LINESTRING(1 5 2, 2 7 20, 1 9 100, 14 12 3)') as geom_b)  ←
       t1;
d3dfullywithin10 | d3dwithin10 | d2dfullywithin20 | d3dfullywithin20
-----+-----+-----+-----
f               | t           | t               | f
```

Siehe auch

[ST_3DMaxDistance](#), [ST_3DDWithin](#), [ST_DWithin](#), [ST_DFullyWithin](#)

8.9.5 ST_3DIntersects

ST_3DIntersects — Returns TRUE if the Geometries "spatially intersect" in 3d - only for points, linestrings, polygons, polyhedral surface (area). With SFCGAL backend enabled also supports TINS

Synopsis

boolean **ST_3DIntersects**(geometry geomA , geometry geomB);

Beschreibung

Overlaps, Touches, Within all imply spatial intersection. If any of the aforementioned returns true, then the geometries also spatially intersect. Disjoint implies false for spatial intersection.

Verfügbarkeit: 2.0.0



Note

This function call will automatically include a bounding box comparison that will make use of any indexes that are available on the geometries.



Note

In order to take advantage of support for TINS, you need to enable the SFCGAL backend. This can be done at session time with: `set postgis.backend = sfcgal;` or at the database or system level. Database level can be done with `ALTER DATABASE gisdb SET postgis.backend = sfcgal;`.



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



This function supports Polyhedral surfaces.



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



This method is also provided by SFCGAL backend.



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

Geometrie Beispiele

```
SELECT ST_3DIntersects(pt, line), ST_Intersects(pt,line)
      FROM (SELECT 'POINT(0 0 2)::geometry As pt,
                  'LINESTRING (0 0 1, 0 2 3 )::geometry As line) As foo;
 st_3dintersects | st_intersects
-----+-----
f                | t
(1 row)
```

TIN Beispiele

```
set postgis.backend = sfcgal;
SELECT ST_3DIntersects('TIN(((0 0,1 0,0 1,0 0)))::geometry, 'POINT(.1 .1)::geometry);
 st_3dintersects
-----
t
```

Siehe auch

[ST_Intersects](#)

8.9.6 ST_3DLongestLine

ST_3DLongestLine — Returns the 3-dimensional longest line between two geometries

Synopsis

geometry **ST_3DLongestLine**(geometry g1, geometry g2);

Beschreibung

Returns the 3-dimensional longest line between two geometries. The function will only return the first longest line if more than one. The line returned will always start in g1 and end in g2. The 3D length of the line this function returns will always be the same as [ST_3DMaxDistance](#) returns for g1 and g2.

Verfügbarkeit: 2.0.0

Changed: 2.2.0 - if 2 2D geometries are input, a 2D point is returned (instead of old behavior assuming 0 for missing Z). In case of 2D and 3D, Z is no longer assumed to be 0 for missing Z.



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



This function supports Polyhedral surfaces.

Beispiele

linestring and point -- both 3d and 2d longest line

```
SELECT ST_AsEWKT(ST_3DLongestLine(line,pt)) AS lol3d_line_pt,
       ST_AsEWKT(ST_LongestLine(line,pt)) As lol2d_line_pt
FROM (SELECT 'POINT(100 100 30)'::geometry As pt,
           'LINESTRING (20 80 20, 98 190 1, 110 180 3, 50 75 1000)':: ↵
       geometry As line
      ) As foo;
```

lol3d_line_pt		lol2d_line_pt
-----+-----		
LINESTRING(50 75 1000,100 100 30) LINESTRING(98 190,100 100)		

linestring and multipoint -- both 3d and 2d longest line

```
SELECT ST_AsEWKT(ST_3DLongestLine(line,pt)) AS lol3d_line_pt,
       ST_AsEWKT(ST_LongestLine(line,pt)) As lol2d_line_pt
FROM (SELECT 'MULTIPOINT(100 100 30, 50 74 1000)'::geometry As pt,
           'LINESTRING (20 80 20, 98 190 1, 110 180 3, 50 75 900)':: ↵
       geometry As line
      ) As foo;
```

lol3d_line_pt		lol2d_line_pt
-----+-----		
LINESTRING(98 190 1,50 74 1000) LINESTRING(98 190,50 74)		

Multilinestring and polygon both 3d and 2d longest line

```
SELECT ST_AsEWKT(ST_3DLongestLine(poly, mline)) As lol3d,
       ST_AsEWKT(ST_LongestLine(poly, mline)) As lol2d
FROM (SELECT ST_GeomFromEWKT('POLYGON((175 150 5, 20 40 5, 35 45 5, 50 60 5, ↵
100 100 5, 175 150 5))') As poly,
       ST_GeomFromEWKT('MULTILINESTRING((175 155 2, 20 40 20, 50 60 -2, 125 ↵
100 1, 175 155 1),
       (1 10 2, 5 20 1))') As mline ) As foo;
lol3d      |      lol2d
-----+-----
LINESTRING(175 150 5,1 10 2) | LINESTRING(175 150,1 10)
```

Siehe auch

[ST_3DClosestPoint](#), [ST_3DDistance](#), [ST_LongestLine](#), [ST_3DShortestLine](#), [ST_3DMaxDistance](#)

8.9.7 ST_3DMaxDistance

ST_3DMaxDistance — For geometry type Returns the 3-dimensional cartesian maximum distance (based on spatial ref) between two geometries in projected units.

Synopsis

```
float ST_3DMaxDistance(geometry g1, geometry g2);
```

Beschreibung

For geometry type returns the 3-dimensional maximum cartesian distance between two geometries in projected units (spatial ref units).



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



This function supports Polyhedral surfaces.

Verfügbarkeit: 2.0.0

Changed: 2.2.0 - In case of 2D and 3D, Z is no longer assumed to be 0 for missing Z.

Beispiele

```
-- Geometry example - units in meters (SRID: 2163 US National Atlas Equal area) (3D point  ←
  and line compared 2D point and line)
-- Note: currently no vertical datum support so Z is not transformed and assumed to be same  ←
  units as final.
SELECT ST_3DMaxDistance(
    ST_Transform(ST_GeomFromEWKT('SRID=4326;POINT(-72.1235 42.3521  ←
      10000)'),2163),
    ST_Transform(ST_GeomFromEWKT('SRID=4326;LINESTRING(-72.1260 42.45  ←
      15, -72.123 42.1546 20)'),2163)
  ) As dist_3d,
  ST_MaxDistance(
    ST_Transform(ST_GeomFromEWKT('SRID=4326;POINT(-72.1235 42.3521  ←
      10000)'),2163),
    ST_Transform(ST_GeomFromEWKT('SRID=4326;LINESTRING(-72.1260 42.45  ←
      15, -72.123 42.1546 20)'),2163)
  ) As dist_2d;

  dist_3d      |      dist_2d
-----+-----
24383.7467488441 | 22247.8472107251
```

Siehe auch

[ST_Distance](#), [ST_3DDWithin](#), [ST_3DMaxDistance](#), [ST_Transform](#)

8.9.8 ST_3DShortestLine

ST_3DShortestLine — Gibt die 3-dimensionale kürzeste Strecke zwischen zwei Geometrien zurück

Synopsis

```
geometry ST_3DShortestLine(geometry g1, geometry g2);
```

Beschreibung

Returns the 3-dimensional shortest line between two geometries. The function will only return the first shortest line if more than one, that the function finds. If g1 and g2 intersects in just one point the function will return a line with both start and end in that intersection-point. If g1 and g2 are intersecting with more than one point the function will return a line with start and end in the same point but it can be any of the intersecting points. The line returned will always start in g1 and end in g2. The 3D length of the line this function returns will always be the same as **ST_3DDistance** returns for g1 and g2.

Verfügbarkeit: 2.0.0

Changed: 2.2.0 - if 2 2D geometries are input, a 2D point is returned (instead of old behavior assuming 0 for missing Z). In case of 2D and 3D, Z is no longer assumed to be 0 for missing Z.



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



This function supports Polyhedral surfaces.

Beispiele

linestring and point -- both 3d and 2d shortest line

```
SELECT ST_AsEWKT(ST_3DShortestLine(line,pt)) AS shl3d_line_pt,
       ST_AsEWKT(ST_ShortestLine(line,pt)) As shl2d_line_pt
FROM (SELECT 'POINT(100 100 30)::geometry As pt,
            'LINESTRING (20 80 20, 98 190 1, 110 180 3, 50 75 1000)'):: geometry As line
      ) As foo;
```

shl3d_line_pt	shl2d_line_pt
LINESTRING(54.6993798867619 128.935022917228 11.5475869506606,100 100 30)	LINESTRING(73.0769230769231 115.384615384615,100 100)

linestring and multipoint -- both 3d and 2d shortest line

```
SELECT ST_AsEWKT(ST_3DShortestLine(line,pt)) AS shl3d_line_pt,
       ST_AsEWKT(ST_ShortestLine(line,pt)) As shl2d_line_pt
FROM (SELECT 'MULTIPOINT(100 100 30, 50 74 1000)::geometry As pt,
            'LINESTRING (20 80 20, 98 190 1, 110 180 3, 50 75 900)'):: geometry As line
      ) As foo;
```

shl2d_line_pt	shl3d_line_pt
LINESTRING(54.6993798867619 128.935022917228 11.5475869506606,100 100 30)	LINESTRING(50 75,50 74)

Multilinestring and polygon both 3d and 2d shortest line

```

SELECT ST_AsEWKT(ST_3DShortestLine(poly, mline)) As shl3d,
       ST_AsEWKT(ST_ShortestLine(poly, mline)) As shl2d
  FROM (SELECT ST_GeomFromEWKT('POLYGON((175 150 5, 20 40 5, 35 45 5, 50 60 5, ↵
100 100 5, 175 150 5))') As poly,
       ST_GeomFromEWKT('MULTILINESTRING((175 155 2, 20 40 20, 50 60 -2, 125 ↵
100 1, 175 155 1),
       (1 10 2, 5 20 1))') As mline ) As foo;
       shl3d ↵
                                     | shl2d
-----|-----
LINESTRING(39.993580415989 54.1889925532825 5,40.4078575708294 53.6052383805529 ↵
5.03423778139177) | LINESTRING(20 40,20 40)

```

Siehe auch

[ST_3DClosestPoint](#), [ST_3DDistance](#), [ST_LongestLine](#), [ST_ShortestLine](#), [ST_3DMaxDistance](#)

8.9.9 ST_Area

ST_Area — Returns the area of the surface if it is a Polygon or MultiPolygon. For geometry, a 2D Cartesian area is determined with units specified by the SRID. For geography, area is determined on a curved surface with units in square meters.

Synopsis

```

float ST_Area(geometry g1);
float ST_Area(geography geog, boolean use_spheroid=true);

```

Beschreibung

Returns the area of the geometry if it is a Polygon or MultiPolygon. Return the area measurement of an ST_Surface or ST_MultiSurface value. For geometry, a 2D Cartesian area is determined with units specified by the SRID. For geography, by default area is determined on a spheroid with units in square meters. To measure around the faster but less accurate sphere, use ST_Area(geog,false).

Enhanced: 2.0.0 - support for 2D polyhedral surfaces was introduced.

Enhanced: 2.2.0 - measurement on spheroid performed with GeographicLib for improved accuracy and robustness. Requires Proj >= 4.9.0 to take advantage of the new feature.



This method implements the [OpenGIS Simple Features Implementation Specification for SQL 1.1](#).



This method implements the SQL/MM specification. SQL-MM 3: 8.1.2, 9.5.3



This function supports Polyhedral surfaces.

**Note**

For polyhedral surfaces, only supports 2D polyhedral surfaces (not 2.5D). For 2.5D, may give a non-zero answer, but only for the faces that sit completely in XY plane.



This method is also provided by SFCGAL backend.

Beispiele

Return area in square feet for a plot of Massachusetts land and multiply by conversion to get square meters. Note this is in square feet because EPSG:2249 is Massachusetts State Plane Feet

```
SELECT ST_Area(the_geom) As sqft, ST_Area(the_geom)*POWER(0.3048,2) As sqm
      FROM (SELECT
            ST_GeomFromText('POLYGON((743238 2967416,743238 2967450,
            743265 2967450,743265.625 2967416,743238 2967416))',2249) ) As foo(
            the_geom);
```

sqft		sqm
928.625		86.27208552

Return area square feet and transform to Massachusetts state plane meters (EPSG:26986) to get square meters. Note this is in square feet because 2249 is Massachusetts State Plane Feet and transformed area is in square meters since EPSG:26986 is state plane Massachusetts meters

```
SELECT ST_Area(the_geom) As sqft, ST_Area(ST_Transform(the_geom,26986)) As sqm
      FROM (SELECT
            ST_GeomFromText('POLYGON((743238 2967416,743238 2967450,
            743265 2967450,743265.625 2967416,743238 2967416))',2249) ) As foo(
            the_geom);
```

sqft		sqm
928.625		86.2724304199219

Return area square feet and square meters using geography data type. Note that we transform to our geometry to geography (before you can do that make sure your geometry is in WGS 84 long lat 4326). Geography always measures in meters. This is just for demonstration to compare. Normally your table will be stored in geography data type already.

```
SELECT ST_Area(the_geog)/POWER(0.3048,2) As sqft_spheroid, ST_Area(the_geog,false)/POWER(
0.3048,2) As sqft_sphere, ST_Area(the_geog) As sqm_spheroid
      FROM (SELECT
            geography(
            ST_Transform(
            ST_GeomFromText('POLYGON((743238 2967416,743238 2967450,743265
            2967450,743265.625 2967416,743238 2967416))',
            2249
            ),4326
            )
            ) As foo(the_geog);
```

sqft_spheroid		sqft_sphere		sqm_spheroid
928.684403538925		927.049336105925		86.2776042893529

```
--if your data is in geography already
SELECT ST_Area(the_geog)/POWER(0.3048,2) As sqft, ST_Area(the_geog) As sqm
      FROM somegeogtable;
```

Siehe auch

[ST_GeomFromText](#), [ST_GeographyFromText](#), [ST_SetSRID](#), [ST_Transform](#)

8.9.10 ST_Azimuth

ST_Azimuth — Returns the north-based azimuth as the angle in radians measured clockwise from the vertical on pointA to pointB.

Synopsis

```
float ST_Azimuth(geometry pointA, geometry pointB);
float ST_Azimuth(geography pointA, geography pointB);
```

Beschreibung

Returns the azimuth in radians of the segment defined by the given point geometries, or NULL if the two points are coincident. The azimuth is angle is referenced from north, and is positive clockwise: North = 0; East = $\pi/2$; South = π ; West = $3\pi/2$.

For the geography type, the forward azimuth is solved as part of the inverse geodesic problem.

The azimuth is mathematical concept defined as the angle between a reference plane and a point, with angular units in radians. Units can be converted to degrees using a built-in PostgreSQL function degrees(), as shown in the example.

Verfügbarkeit: 1.1.0

Enhanced: 2.0.0 support for geography was introduced.

Enhanced: 2.2.0 measurement on spheroid performed with GeographicLib for improved accuracy and robustness. Requires Proj >= 4.9.0 to take advantage of the new feature.

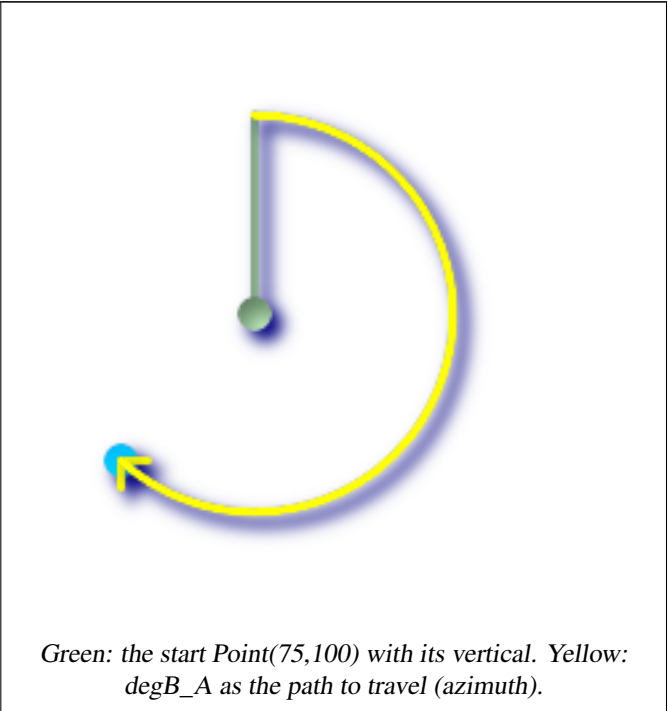
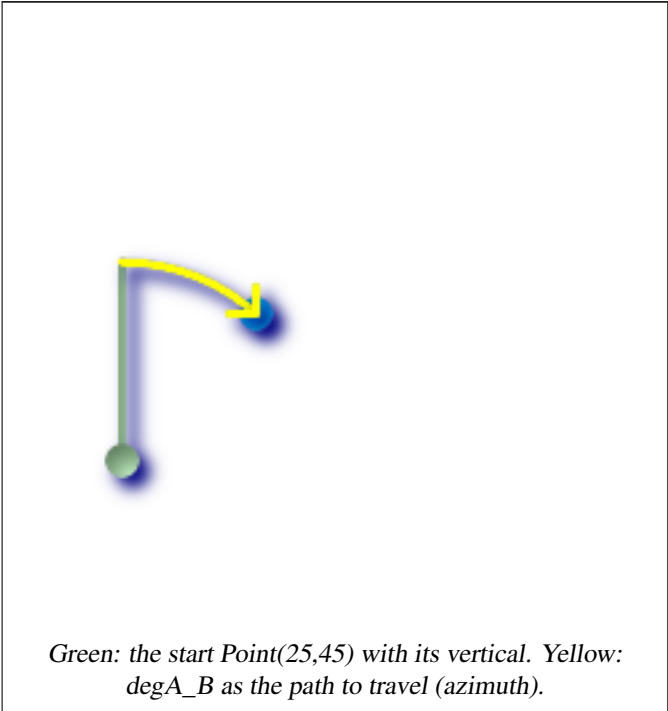
Azimuth is especially useful in conjunction with ST_Translate for shifting an object along its perpendicular axis. See up-[gis_lineshift](#) [Plpgsqlfunctions PostGIS wiki section](#) for example of this.

Beispiele

Geometry Azimuth in degrees

```
SELECT degrees(ST_Azimuth(ST_Point(25, 45), ST_Point(75, 100))) AS degA_B,
       degrees(ST_Azimuth(ST_Point(75, 100), ST_Point(25, 45))) AS degB_A;
```

degA_b	degb_a
42.2736890060937	222.273689006094



Siehe auch

[ST_Point](#), [ST_Translate](#), [ST_Project](#), [PostgreSQL Math Functions](#)

8.9.11 ST_Centroid

ST_Centroid — Gibt den geometrischen Schwerpunkt der Geometrie zurück.

Synopsis

```
geometry ST_Centroid(geometry g1);  
geography ST_Centroid(geography g1, boolean use_spheroid=true);
```

Beschreibung

Computes the geometric center of a geometry, or equivalently, the center of mass of the geometry as a `POINT`. For `[MULTI]POINTS`, this is computed as the arithmetic mean of the input coordinates. For `[MULTI]LINESTRINGS`, this is computed as the weighted length of each line segment. For `[MULTI]POLYGONS`, "weight" is thought in terms of area. If an empty geometry is supplied, an empty `GEOMETRYCOLLECTION` is returned. If `NULL` is supplied, `NULL` is returned. If `CIRCULARSTRING` or `COMPOUNDCURVE` are supplied, they are converted to `linestring` with `CurveToLine` first, then same than for `LINESTRING`.

New in 2.3.0 : support `CIRCULARSTRING` and `COMPOUNDCURVE` (using `CurveToLine`)

Availability: 2.4.0 support for geography was introduced.

The centroid is equal to the centroid of the set of component Geometries of highest dimension (since the lower-dimension geometries contribute zero "weight" to the centroid).



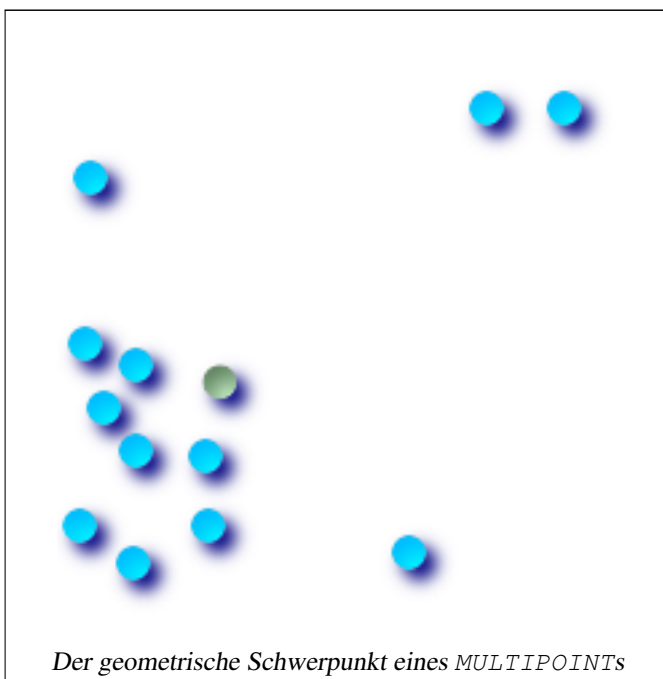
This method implements the [OpenGIS Simple Features Implementation Specification for SQL 1.1](#).

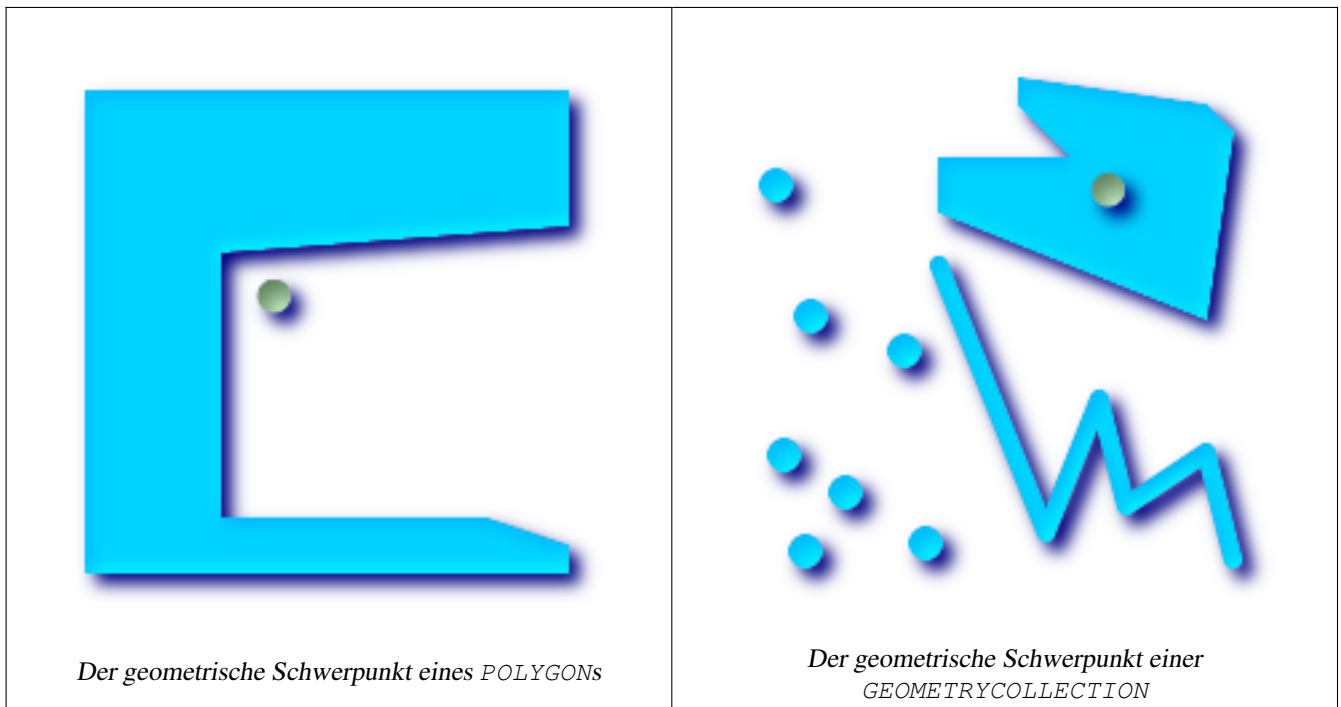


This method implements the SQL/MM specification. SQL-MM 3: 8.1.4, 9.5.5

Beispiele

In each of the following illustrations, the green dot represents the centroid of the source geometry.





```
SELECT ST_AsText(ST_Centroid('MULTIPOINT ( -1 0, -1 2, -1 3, -1 4, -1 7, 0 1, 0 3, 1 1, 2
0, 6 0, 7 8, 9 8, 10 6 )'));
-----
st_astext
POINT(2.30769230769231 3.30769230769231)
(1 row)

SELECT ST_AsText(ST_centroid(g))
FROM ST_GeomFromText('CIRCULARSTRING(0 2, -1 1,0 0, 0.5 0, 1 0, 2 1, 1 2, 0.5 2, 0 2)') AS g ;
-----
POINT(0.5 1)

SELECT ST_AsText(ST_centroid(g))
FROM ST_GeomFromText('COMPOUNDCURVE(CIRCULARSTRING(0 2, -1 1,0 0),(0 0, 0.5 0, 1 0),
CIRCULARSTRING( 1 0, 2 1, 1 2),(1 2, 0.5 2, 0 2))' ) AS g;
-----
POINT(0.5 1)
```

Siehe auch

[ST_PointOnSurface](#)

8.9.12 ST_ClosestPoint

ST_ClosestPoint — Returns the 2-dimensional point on g1 that is closest to g2. This is the first point of the shortest line.

Synopsis

geometry **ST_ClosestPoint**(geometry g1, geometry g2);

Beschreibung

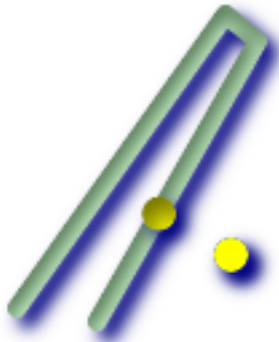
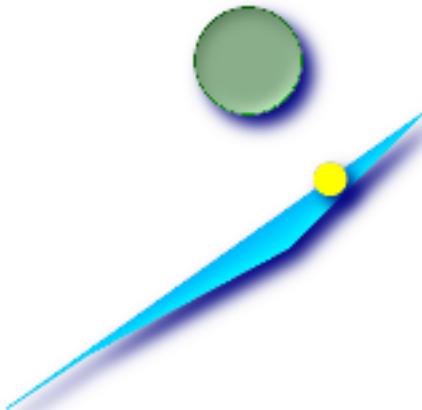
Returns the 2-dimensional point on g1 that is closest to g2. This is the first point of the shortest line.



Note
Falls es sich um eine 3D-Geometrie handelt, sollten Sie **ST_3DClosestPoint** vorziehen.

Verfügbarkeit: 1.5.0

Beispiele

 <i>Closest between point and linestring is the point itself, but closest point between a linestring and point is the point on line string that is closest.</i> <pre>SELECT ST_AsText(ST_ClosestPoint(pt,line) ↔) AS cp_pt_line, ST_AsText(ST_ClosestPoint(line,pt ↔)) As cp_line_pt FROM (SELECT 'POINT(100 100) '::geometry ↔ As pt, 'LINESTRING (20 80, 98 ↔ 190, 110 180, 50 75) '::geometry As line) As foo;</pre> <table><tr><td>cp_pt_line</td><td> </td><td>↔</td></tr><tr><td>cp_line_pt</td><td></td><td></td></tr></table> <pre>POINT(100 100) POINT(73.0769230769231 ↔ 115.384615384615)</pre>	cp_pt_line		↔	cp_line_pt			 <i>Der Punkt des Polygons A der am nächsten beim Polygon B liegt</i> <pre>SELECT ST_AsText (ST_ClosestPoint (ST_GeomFromText (' ↔ POLYGON((175 150, 20 40, 50 60, 125 100, 175 150 ↔ ST_Buffer(↔ ST_GeomFromText('POINT(110 170)'), 20) ↔) As ptwkt;</pre> <table><tr><td>ptwkt</td><td></td><td>↔</td></tr></table> <pre>POINT(140.752120669087 125.695053378061)</pre>	ptwkt		↔
cp_pt_line		↔								
cp_line_pt										
ptwkt		↔								

Siehe auch

[ST_3DClosestPoint](#), [ST_Distance](#), [ST_LongestLine](#), [ST_ShortestLine](#), [ST_MaxDistance](#)

8.9.13 ST_ClusterDBSCAN

`ST_ClusterDBSCAN` — Windowing function that returns integer id for the cluster each input geometry is in based on 2D implementation of Density-based spatial clustering of applications with noise (DBSCAN) algorithm.

Synopsis

integer **ST_ClusterDBSCAN**(geometry winset geom, float8 eps, integer minpoints);

Beschreibung

Returns cluster number for each input geometry, based on a 2D implementation of the [Density-based spatial clustering of applications with noise \(DBSCAN\)](#) algorithm. Unlike [ST_ClusterKMeans](#), it does not require the number of clusters to be specified, but instead uses the desired distance (`eps`) and density(`minpoints`) parameters to construct each cluster.

An input geometry will be added to a cluster if it is either:

- A "core" geometry, that is within `eps` distance (Cartesian) of at least `minpoints` input geometries (including itself) or
- A "border" geometry, that is within `eps` distance of a core geometry.

Note that border geometries may be within `eps` distance of core geometries in more than one cluster; in this case, either assignment would be correct, and the border geometry will be arbitrarily assigned to one of the available clusters. In these cases, it is possible for a correct cluster to be generated with fewer than `minpoints` geometries. When assignment of a border geometry is ambiguous, repeated calls to `ST_ClusterDBSCAN` will produce identical results if an `ORDER BY` clause is included in the window definition, but cluster assignments may differ from other implementations of the same algorithm.

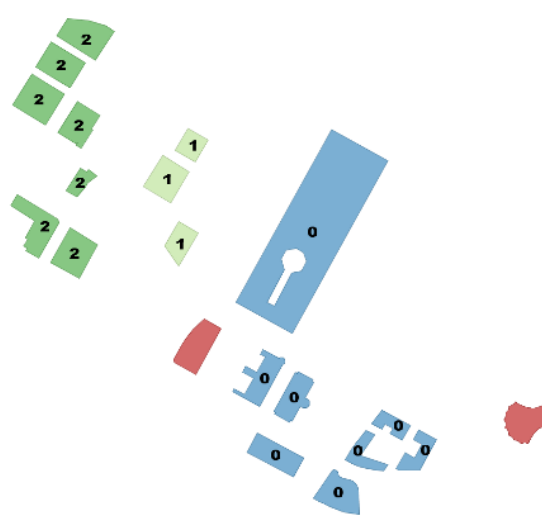
**Note**

Input geometries that do not meet the criteria to join any other cluster will be assigned a cluster number of NULL.

Verfügbarkeit: 2.3.0 - benötigt GEOS

Beispiele

Assigning a cluster number to each polygon within 50 meters of each other. Require at least 2 polygons per cluster

 <i>within 50 meters at least 2 per cluster. singletons have NULL for cid</i> <pre>SELECT name, ST_ClusterDBSCAN(geom, eps ↵ := 50, minpoints := 2) over () AS cid FROM boston_polys WHERE name > '' AND building > '' AND ST_DWithin(geom, ST_Transform(ST_GeomFromText('POINT ↵ (-71.04054 42.35141)', 4326), 26986), 500);</pre>	<table><tr><th>bucket</th><th>name</th><th></th><th>↵</th></tr><tr><td colspan="4">-----+-----</td></tr><tr><td></td><td>Manulife Tower</td><td> </td><td>↵</td></tr><tr><td>0</td><td></td><td></td><td></td></tr><tr><td></td><td>Park Lane Seaport I</td><td> </td><td>↵</td></tr><tr><td>0</td><td></td><td></td><td></td></tr><tr><td></td><td>Park Lane Seaport II</td><td> </td><td>↵</td></tr><tr><td>0</td><td></td><td></td><td></td></tr><tr><td></td><td>Renaissance Boston Waterfront Hotel</td><td> </td><td>↵</td></tr><tr><td>0</td><td></td><td></td><td></td></tr><tr><td></td><td>Seaport Boston Hotel</td><td> </td><td>↵</td></tr><tr><td>0</td><td></td><td></td><td></td></tr><tr><td></td><td>Seaport Hotel & World Trade Center</td><td> </td><td>↵</td></tr><tr><td>0</td><td></td><td></td><td></td></tr><tr><td></td><td>Waterside Place</td><td> </td><td>↵</td></tr><tr><td>0</td><td></td><td></td><td></td></tr><tr><td></td><td>World Trade Center East</td><td> </td><td>↵</td></tr><tr><td>0</td><td></td><td></td><td></td></tr><tr><td></td><td>100 Northern Avenue</td><td> </td><td>↵</td></tr><tr><td>1</td><td></td><td></td><td></td></tr><tr><td></td><td>100 Pier 4</td><td> </td><td>↵</td></tr><tr><td>1</td><td></td><td></td><td></td></tr><tr><td></td><td>The Institute of Contemporary Art</td><td> </td><td>↵</td></tr><tr><td>1</td><td></td><td></td><td></td></tr><tr><td></td><td>101 Seaport</td><td> </td><td>↵</td></tr><tr><td>2</td><td></td><td></td><td></td></tr><tr><td></td><td>District Hall</td><td> </td><td>↵</td></tr><tr><td>2</td><td></td><td></td><td></td></tr><tr><td></td><td>One Marina Park Drive</td><td> </td><td>↵</td></tr><tr><td>2</td><td></td><td></td><td></td></tr><tr><td></td><td>Twenty Two Liberty</td><td> </td><td>↵</td></tr><tr><td>2</td><td></td><td></td><td></td></tr><tr><td></td><td>Vertex</td><td> </td><td>↵</td></tr><tr><td>2</td><td></td><td></td><td></td></tr><tr><td></td><td>Vertex</td><td> </td><td>↵</td></tr><tr><td>2</td><td></td><td></td><td></td></tr><tr><td></td><td>Watermark Seaport</td><td> </td><td>↵</td></tr><tr><td>2</td><td></td><td></td><td></td></tr><tr><td></td><td>Blue Hills Bank Pavilion</td><td> </td><td>↵</td></tr><tr><td>NULL</td><td></td><td></td><td></td></tr><tr><td></td><td>World Trade Center West</td><td> </td><td>↵</td></tr><tr><td>NULL</td><td></td><td></td><td></td></tr><tr><td>(20 rows)</td><td></td><td></td><td></td></tr></table>	bucket	name		↵	-----+-----					Manulife Tower		↵	0					Park Lane Seaport I		↵	0					Park Lane Seaport II		↵	0					Renaissance Boston Waterfront Hotel		↵	0					Seaport Boston Hotel		↵	0					Seaport Hotel & World Trade Center		↵	0					Waterside Place		↵	0					World Trade Center East		↵	0					100 Northern Avenue		↵	1					100 Pier 4		↵	1					The Institute of Contemporary Art		↵	1					101 Seaport		↵	2					District Hall		↵	2					One Marina Park Drive		↵	2					Twenty Two Liberty		↵	2					Vertex		↵	2					Vertex		↵	2					Watermark Seaport		↵	2					Blue Hills Bank Pavilion		↵	NULL					World Trade Center West		↵	NULL				(20 rows)			
bucket	name		↵																																																																																																																																																																										
-----+-----																																																																																																																																																																													
	Manulife Tower		↵																																																																																																																																																																										
0																																																																																																																																																																													
	Park Lane Seaport I		↵																																																																																																																																																																										
0																																																																																																																																																																													
	Park Lane Seaport II		↵																																																																																																																																																																										
0																																																																																																																																																																													
	Renaissance Boston Waterfront Hotel		↵																																																																																																																																																																										
0																																																																																																																																																																													
	Seaport Boston Hotel		↵																																																																																																																																																																										
0																																																																																																																																																																													
	Seaport Hotel & World Trade Center		↵																																																																																																																																																																										
0																																																																																																																																																																													
	Waterside Place		↵																																																																																																																																																																										
0																																																																																																																																																																													
	World Trade Center East		↵																																																																																																																																																																										
0																																																																																																																																																																													
	100 Northern Avenue		↵																																																																																																																																																																										
1																																																																																																																																																																													
	100 Pier 4		↵																																																																																																																																																																										
1																																																																																																																																																																													
	The Institute of Contemporary Art		↵																																																																																																																																																																										
1																																																																																																																																																																													
	101 Seaport		↵																																																																																																																																																																										
2																																																																																																																																																																													
	District Hall		↵																																																																																																																																																																										
2																																																																																																																																																																													
	One Marina Park Drive		↵																																																																																																																																																																										
2																																																																																																																																																																													
	Twenty Two Liberty		↵																																																																																																																																																																										
2																																																																																																																																																																													
	Vertex		↵																																																																																																																																																																										
2																																																																																																																																																																													
	Vertex		↵																																																																																																																																																																										
2																																																																																																																																																																													
	Watermark Seaport		↵																																																																																																																																																																										
2																																																																																																																																																																													
	Blue Hills Bank Pavilion		↵																																																																																																																																																																										
NULL																																																																																																																																																																													
	World Trade Center West		↵																																																																																																																																																																										
NULL																																																																																																																																																																													
(20 rows)																																																																																																																																																																													

Combining parcels with the same cluster number into a single geometry. This uses named argument calling

```
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_ClusterWithin](#)

8.9.14 ST_ClusterIntersecting

ST_ClusterIntersecting — Aggregate. Returns an array with the connected components of a set of geometries

Synopsis

```
geometry[] ST_ClusterIntersecting(geometry set g);
```

Beschreibung

ST_ClusterIntersecting is an aggregate function that returns an array of GeometryCollections, where each GeometryCollection represents an interconnected set of geometries.

Verfügbarkeit: 2.2.0 - benötigt GEOS

Beispiele

```
WITH testdata AS
  (SELECT unnest(ARRAY['LINESTRING (0 0, 1 1)::geometry',
                       'LINESTRING (5 5, 4 4)::geometry',
                       'LINESTRING (6 6, 7 7)::geometry',
                       'LINESTRING (0 0, -1 -1)::geometry',
                       'POLYGON ((0 0, 4 0, 4 4, 0 4, 0 0))::geometry']) AS geom)

SELECT ST_AsText(unnest(ST_ClusterIntersecting(geom))) FROM testdata;

--result

st_astext
-----
GEOMETRYCOLLECTION(LINESTRING(0 0,1 1),LINESTRING(5 5,4 4),LINESTRING(0 0,-1 -1),POLYGON((0 0,4 0,4 4,0 4,0 0)))
GEOMETRYCOLLECTION(LINESTRING(6 6,7 7))
```

Siehe auch

[ST_ClusterDBSCAN](#), [ST_ClusterKMeans](#), [ST_ClusterWithin](#)

8.9.15 ST_ClusterKMeans

ST_ClusterKMeans — Windowing function that returns integer id for the cluster each input geometry is in.

Synopsis

```
integer ST_ClusterKMeans(geometry winset geom, integer number_of_clusters);
```

Beschreibung

Returns 2D distance based **k-means** cluster number for each input geometry. The distance used for clustering is the distance between the centroids of the geometries.

Verfügbarkeit: 2.3.0 - benötigt GEOS

Beispiele

Generate dummy set of parcels for examples

```
CREATE TABLE parcels AS
SELECT lpad((row_number() over())::text,3,'0') As parcel_id, geom,
('{residential, commercial}'::text[])[1 + mod(row_number()OVER(),2)] As type
FROM
  ST_Subdivide(ST_Buffer('LINESTRING(40 100, 98 100, 100 150, 60 90)'::geometry,
    40, 'endcap=square'),12) As geom;
```



Ursprüngliche Parzellen

```
-- Partitioning parcel clusters by type
SELECT ST_ClusterKMeans(geom,3) over (PARTITION BY type) AS cid, parcel_id, type
FROM parcels;
-- result
cid | parcel_id | type
-----+-----+-----
  1 | 005      | commercial
  1 | 003      | commercial
  2 | 007      | commercial
  0 | 001      | commercial
  1 | 004      | residential
  0 | 002      | residential
  2 | 006      | residential
(7 rows)
```

Siehe auch

[ST_ClusterDBSCAN](#), [ST_ClusterIntersecting](#), [ST_ClusterWithin](#), [ST_Subdivide](#)

8.9.16 ST_ClusterWithin

ST_ClusterWithin — Aggregate. Returns an array of GeometryCollections, where each GeometryCollection represents a set of geometries separated by no more than the specified distance.

Synopsis

```
geometry[] ST_ClusterWithin(geometry set g, float8 distance);
```

Beschreibung

ST_ClusterWithin is an aggregate function that returns an array of **GeometryCollections**, where each **GeometryCollection** represents a set of geometries separated by no more than the specified distance. (Distances are Cartesian distances in the units of the SRID.)

Verfügbarkeit: 2.2.0 - benötigt GEOS

Beispiele

```
WITH testdata AS
  (SELECT unnest(ARRAY['LINESTRING (0 0, 1 1)::geometry',
                       'LINESTRING (5 5, 4 4)::geometry',
                       'LINESTRING (6 6, 7 7)::geometry',
                       'LINESTRING (0 0, -1 -1)::geometry',
                       'POLYGON ((0 0, 4 0, 4 4, 0 4, 0 0))::geometry']) AS geom)

SELECT ST_AsText(unnest(ST_ClusterWithin(geom, 1.4))) FROM testdata;

--result

st_astext
-----
GEOMETRYCOLLECTION(LINESTRING(0 0,1 1),LINESTRING(5 5,4 4),LINESTRING(0 0,-1 -1),POLYGON((0 0,4 0,4 4,0 4,0 0)))
GEOMETRYCOLLECTION(LINESTRING(6 6,7 7))
```

Siehe auch

[ST_ClusterDBSCAN](#), [ST_ClusterKMeans](#), [ST_ClusterIntersecting](#)

8.9.17 ST_Contains

ST_Contains — Returns true if and only if no points of B lie in the exterior of A, and at least one point of the interior of B lies in the interior of A.

Synopsis

```
boolean ST_Contains(geometry geomA, geometry geomB);
```

Beschreibung

Geometry A contains Geometry B if and only if no points of B lie in the exterior of A, and at least one point of the interior of B lies in the interior of A. An important subtlety of this definition is that A does not contain its boundary, but A does contain itself. Contrast that to [ST_ContainsProperly](#) where geometry A does not Contain Properly itself.

Returns TRUE if geometry B is completely inside geometry A. For this function to make sense, the source geometries must both be of the same coordinate projection, having the same SRID. **ST_Contains** is the inverse of **ST_Within**. So **ST_Contains**(A,B) implies **ST_Within**(B,A) except in the case of invalid geometries where the result is always false regardless or not defined.

Wird durch das GEOS Modul ausgeführt

Enhanced: 2.3.0 Enhancement to PIP short-circuit extended to support MultiPoints with few points. Prior versions only supported point in polygon.

**Important**

Do not call with a `GEOMETRYCOLLECTION` as an argument

**Important**

Do not use this function with invalid geometries. You will get unexpected results.

This function call will automatically include a bounding box comparison that will make use of any indexes that are available on the geometries. To avoid index use, use the function `_ST_Contains`.

NOTE: this is the "allowable" version that returns a boolean, not an integer.



This method implements the [OpenGIS Simple Features Implementation Specification for SQL 1.1](#). s2.1.1.2 // s2.1.13.3 - ident mit `within(geometry B, geometry A)`

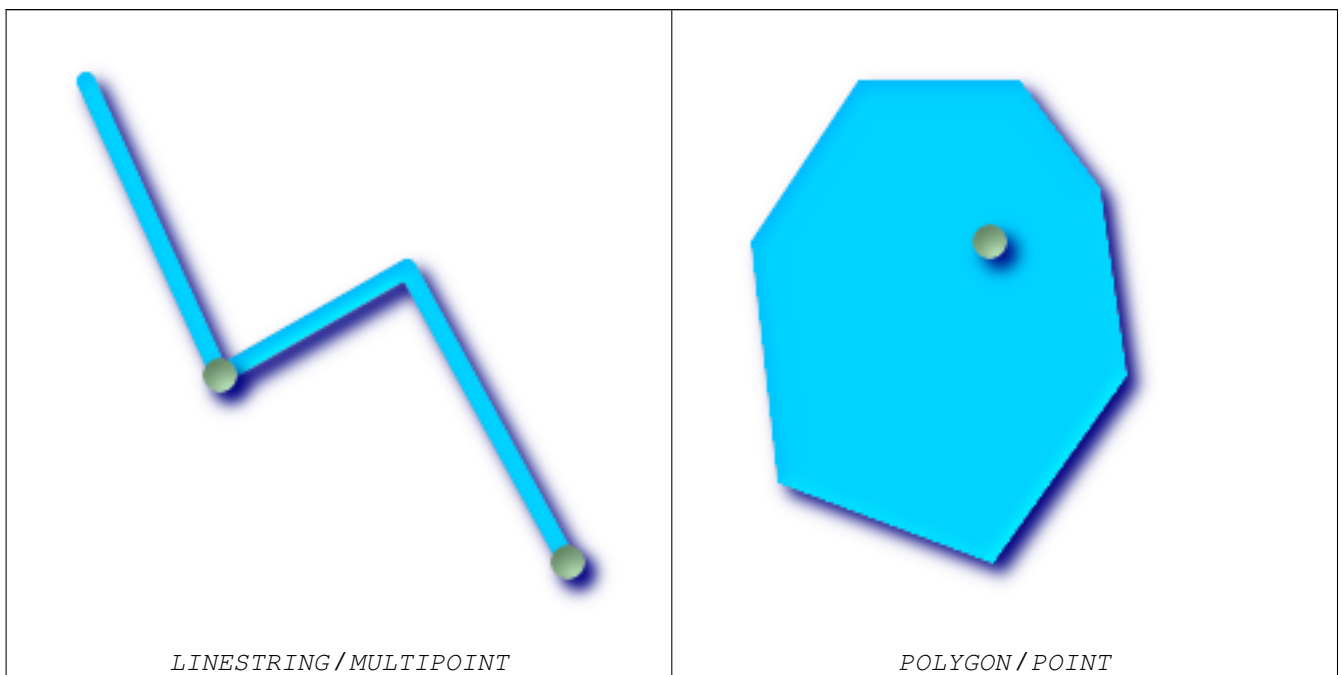


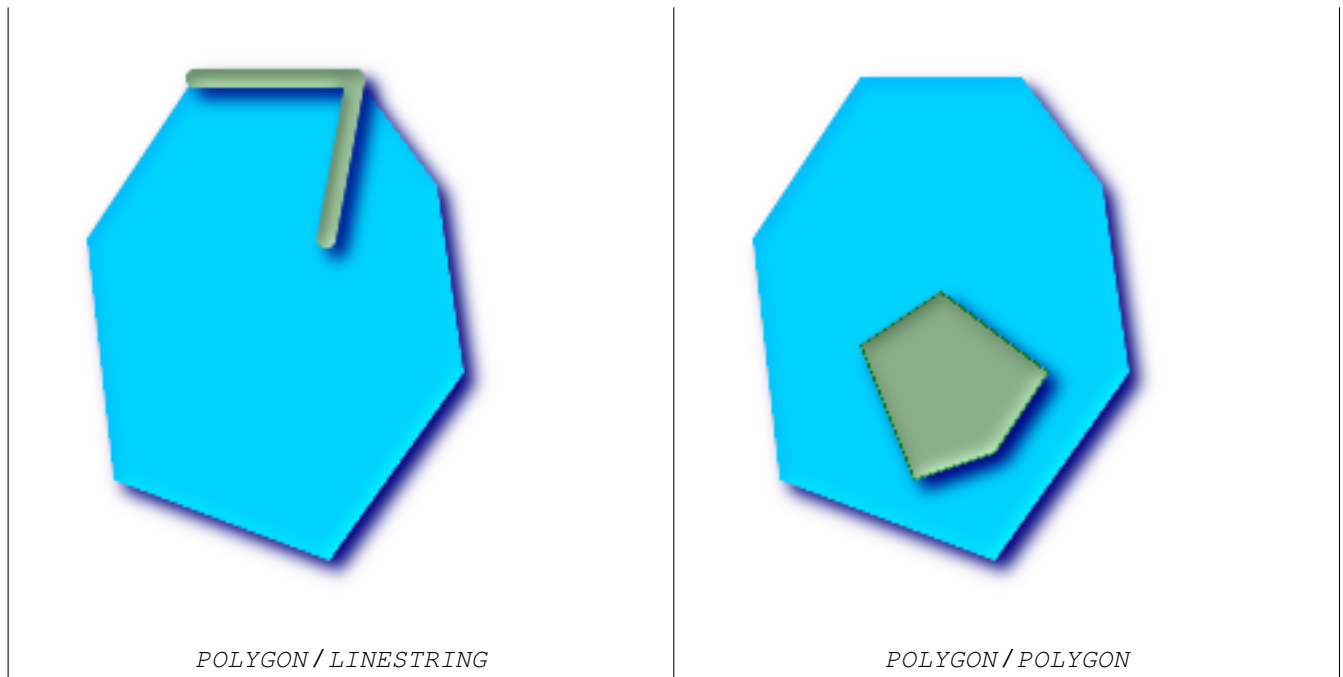
This method implements the SQL/MM specification. SQL-MM 3: 5.1.31

There are certain subtleties to `ST_Contains` and `ST_Within` that are not intuitively obvious. For details check out [Subtleties of OGC Covers, Contains, Within](#)

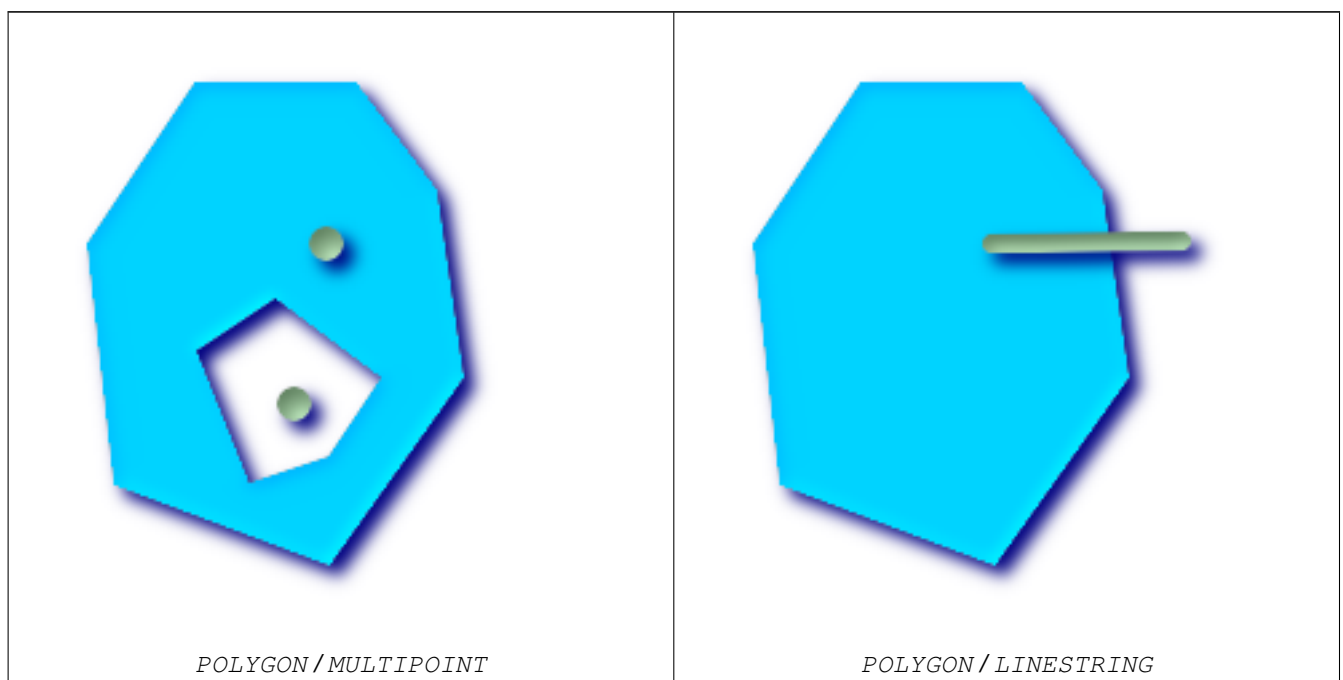
Beispiele

The `ST_Contains` predicate returns `TRUE` in all the following illustrations.





The `ST_Contains` predicate returns `FALSE` in all the following illustrations.



```
-- Ein Kreis innerhalb eines Kreises
SELECT ST_Contains(smallc, bigc) As smallcontainsbig,
       ST_Contains(bigc, smallc) As bigcontainssmall,
       ST_Contains(bigc, ST_Union(smallc, bigc)) as bigcontainsunion,
       ST_Equals(bigc, ST_Union(smallc, bigc)) as bigisunion,
       ST_Covers(bigc, ST_ExteriorRing(bigc)) As bigcoversexterior,
       ST_Contains(bigc, ST_ExteriorRing(bigc)) As bigcontainsexterior
FROM (SELECT ST_Buffer(ST_GeomFromText('POINT(1 2)'), 10) As smallc,
            ST_Buffer(ST_GeomFromText('POINT(1 2)'), 20) As bigc) As foo;

-- Result
```



```

smallcontainsbig | bigcontainssmall | bigcontainsunion | bigisunion | bigcoversexterior | ↩
bigcontainsexterior
-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
f                | t                | t                | t                | t                | f
-- Beispiel für den Unterschied zwischen "contains" und "contains properly"
SELECT ST_GeometryType(geomA) As geomtype, ST_Contains(geomA,geomA) AS acontainsa, ↩
       ST_ContainsProperly(geomA, geomA) AS acontainspropa,
       ST_Contains(geomA, ST_Boundary(geomA)) As acontainsba, ST_ContainsProperly(geomA, ↩
       ST_Boundary(geomA)) As acontainspropba
FROM (VALUES ( ST_Buffer(ST_Point(1,1), 5,1) ),
             ( ST_MakeLine(ST_Point(1,1), ST_Point(-1,-1) ) ),
             ( ST_Point(1,1) )
        ) As foo(geomA);

geomtype      | acontainsa | acontainspropa | acontainsba | acontainspropba
-----+-----+-----+-----+-----+-----+
ST_Polygon    | t          | f              | f           | f
ST_LineString | t          | f              | f           | f
ST_Point      | t          | t              | f           | f

```

Siehe auch

[ST_Boundary](#), [ST_ContainsProperly](#), [ST_Covers](#), [ST_CoveredBy](#), [ST_Equals](#), [ST_Within](#)

8.9.18 ST_ContainsProperly

ST_ContainsProperly — Returns true if B intersects the interior of A but not the boundary (or exterior). A does not contain properly itself, but does contain itself.

Synopsis

boolean **ST_ContainsProperly**(geometry geomA, geometry geomB);

Beschreibung

Returns true if B intersects the interior of A but not the boundary (or exterior).

A does not contain properly itself, but does contain itself.

Every point of the other geometry is a point of this geometry's interior. The DE-9IM Intersection Matrix for the two geometries matches [T**FF*FF*] used in [ST_Relate](#)

Note



From JTS docs slightly reworded: The advantage to using this predicate over [ST_Contains](#) and [ST_Intersects](#) is that it can be computed efficiently, with no need to compute topology at individual points. An example use case for this predicate is computing the intersections of a set of geometries with a large polygonal geometry. Since intersection is a fairly slow operation, it can be more efficient to use containsProperly to filter out test geometries which lie wholly inside the area. In these cases the intersection is known a priori to be exactly the original test geometry.

Verfügbarkeit: 1.4.0 - benötigt GEOS >= 3.1.0.



Important

Do not call with a `GEOMETRYCOLLECTION` as an argument



Important

Do not use this function with invalid geometries. You will get unexpected results.

This function call will automatically include a bounding box comparison that will make use of any indexes that are available on the geometries. To avoid index use, use the function `_ST_ContainsProperly`.

Beispiele

```
--Ein Kreis innerhalb eines Kreises
SELECT ST_ContainsProperly(smallc, bigc) As smallcontainspropbig,
ST_ContainsProperly(bigc, smallc) As bigcontainspropsmall,
ST_ContainsProperly(bigc, ST_Union(smallc, bigc)) as bigcontainspropunion,
ST_Equals(bigc, ST_Union(smallc, bigc)) as bigisunion,
ST_Covers(bigc, ST_ExteriorRing(bigc)) As bigcoversexterior,
ST_ContainsProperly(bigc, ST_ExteriorRing(bigc)) As bigcontainsexterior
FROM (SELECT ST_Buffer(ST_GeomFromText('POINT(1 2)'), 10) As smallc,
ST_Buffer(ST_GeomFromText('POINT(1 2)'), 20) As bigc) As foo;
--Result
smallcontainspropbig | bigcontainspropsmall | bigcontainspropunion | bigisunion | bigcoversexterior | bigcontainsexterior
```

Diagram illustrating the structure of a sequence of operations (f, t) over time steps, with a dashed line above and a solid line below. The sequence is: f, t, f, t, t.

```
--Beispiel für den Unterschied zwischen "contains" und "contains properly"
SELECT ST_GeometryType(geomA) As geomtype, ST_Contains(geomA,geomA) AS acontainsa, ←
       ST_ContainsProperly(geomA, geomA) AS acontainspropa,
       ST_Contains(geomA, ST_Boundary(geomA)) As acontainsba, ST_ContainsProperly(geomA, ←
       ST_Boundary(geomA)) As acontainspropba
FROM (VALUES ( ST_Buffer(ST_Point(1,1), 5,1) ),
             ( ST_MakeLine(ST_Point(1,1), ST_Point(-1,-1) ) ),
             ( ST_Point(1,1) )
       ) As foo(geomA);
```

geomtype	acontainsa	acontainspropa	acontainsba	acontainspropba
ST_Polygon	t	f	f	f
ST_LineString	t	f	f	f
ST_Point	t	t	f	f

Siehe auch

ST GeometryType, ST_Boundary, ST_Contains, ST_Covers, ST_CoveredBy, ST_Equals, ST_Relate, ST_Within

8.9.19 ST Covers

ST_Covers — Gibt 1 (TRUE) zurück, falls kein Punkt der Geometrie B außerhalb von Geometry A liegt

Synopsis

```
boolean ST_Covers(geometry geomA, geometry geomB);
boolean ST_Covers(geography geogpolyA, geography geogpointB);
```

Beschreibung

Gibt 1 (TRUE) zurück, falls kein Punkt der Geometrie/Geographie B außerhalb von Geometry/Geographie A liegt

Wird durch das GEOS Modul ausgeführt



Important

Do not call with a `GEOMETRYCOLLECTION` as an argument



Important

Do not use this function with invalid geometries. You will get unexpected results.

This function call will automatically include a bounding box comparison that will make use of any indexes that are available on the geometries. To avoid index use, use the function `_ST_Covers`.

Enhanced: 2.4.0 Support for polygon in polygon and line in polygon added for geography type

Enhanced: 2.3.0 Enhancement to PIP short-circuit for geometry extended to support MultiPoints with few points. Prior versions only supported point in polygon.

Verfügbarkeit: 1.5 - Unterstützung von geographischen Koordinaten.

Verfügbarkeit: 1.2.2 - benötigt GEOS >= 3.0

NOTE: this is the "allowable" version that returns a boolean, not an integer.

Kein OGC-Standard, aber Oracle hat das auch.

There are certain subtleties to `ST_Contains` and `ST_Within` that are not intuitively obvious. For details check out [Subtleties of OGC Covers, Contains, Within](#)

Beispiele

Geometrie Beispiel

```
--Ein Kreis, der einen anderen Kreis überdeckt
SELECT ST_Covers(smallc,smallc) As smallinsmall,
       ST_Covers(smallc, bigc) As smallcoversbig,
       ST_Covers(bigc, ST_ExteriorRing(bigc)) As bigcoversexterior,
       ST_Contains(bigc, ST_ExteriorRing(bigc)) As bigcontainsexterior
FROM (SELECT ST_Buffer(ST_GeomFromText('POINT(1 2)'), 10) As smallc,
       ST_Buffer(ST_GeomFromText('POINT(1 2)'), 20) As bigc) As foo;
--Result
smallinsmall | smallcoversbig | bigcoversexterior | bigcontainsexterior
-----+-----+-----+-----
t            | f              | t                 | f
(1 row)
```

Geographie Beispiel

```
-- a point with a 300 meter buffer compared to a point, a point and its 10 meter buffer
SELECT ST_Covers(geog_poly, geog_pt) As poly_covers_pt,
       ST_Covers(ST_Buffer(geog_pt,10), geog_pt) As buff_10m_covers_cent
FROM (SELECT ST_Buffer(ST_GeogFromText('SRID=4326;POINT(-99.327 31.4821)'), 300) As ←
      geog_poly,
      ST_GeogFromText('SRID=4326;POINT(-99.33 31.483)') As ←
      geog_pt ) As foo;
```

poly_covers_pt	buff_10m_covers_cent
f	t

Siehe auch

[ST_Contains](#), [ST_CoveredBy](#), [ST_Within](#)

8.9.20 ST_CoveredBy

ST_CoveredBy — Gibt 1 (TRUE) zurück, falls kein Punkt der Geometrie/Geographie A außerhalb von Geometry/Geographie B liegt

Synopsis

```
boolean ST_CoveredBy(geometry geomA, geometry geomB);
boolean ST_CoveredBy(geography geogA, geography geogB);
```

Beschreibung

Gibt 1 (TRUE) zurück, falls kein Punkt der Geometrie/Geographie A außerhalb von Geometry/Geographie B liegt

Wird durch das GEOS Modul ausgeführt



Important

Do not call with a `GEOMETRYCOLLECTION` as an argument



Important

Do not use this function with invalid geometries. You will get unexpected results.

Verfügbarkeit: 1.2.2 - benötigt GEOS >= 3.0

This function call will automatically include a bounding box comparison that will make use of any indexes that are available on the geometries. To avoid index use, use the function `_ST_CoveredBy`.

NOTE: this is the "allowable" version that returns a boolean, not an integer.

Kein OGC-Standard, aber Oracle hat das auch.

There are certain subtleties to `ST_Contains` and `ST_Within` that are not intuitively obvious. For details check out [Subtleties of OGC Covers, Contains, Within](#)

Beispiele

```
--Ein Kreis abgedeckt durch einen Kreis
SELECT ST_CoveredBy(smallc,smallc) As smallinsmall,
       ST_CoveredBy(smallc, bigc) As smallcoveredbybig,
       ST_CoveredBy(ST_ExteriorRing(bigc), bigc) As exteriorcoveredbybig,
       ST_Within(ST_ExteriorRing(bigc),bigc) As exeriorwithinbig
FROM (SELECT ST_Buffer(ST_GeomFromText('POINT(1 2)'), 10) As smallc,
       ST_Buffer(ST_GeomFromText('POINT(1 2)'), 20) As bigc) As foo;
--Result
smallinsmall | smallcoveredbybig | exteriorcoveredbybig | exeriorwithinbig
-----+-----+-----+-----
t           | t                 | t                     | f
(1 row)
```

Siehe auch

[ST_Contains](#), [ST_Covers](#), [ST_ExteriorRing](#), [ST_Within](#)

8.9.21 ST_Crosses

ST_Crosses — Returns TRUE if the supplied geometries have some, but not all, interior points in common.

Synopsis

boolean **ST_Crosses**(geometry g1, geometry g2);

Beschreibung

ST_Crosses takes two geometry objects and returns TRUE if their intersection "spatially cross", that is, the geometries have some, but not all interior points in common. The intersection of the interiors of the geometries must not be the empty set and must have a dimensionality less than the maximum dimension of the two input geometries. Additionally, the intersection of the two geometries must not equal either of the ~~source~~ geometries. Otherwise, it returns FALSE.

In mathematical terms, this is expressed as:

$$a.Crosses(b) \Leftrightarrow (dim(I(a) \cap I(b)) < \max(dim(I(a)), dim(I(b)))) \wedge (a \cap b \neq a) \wedge (a \cap b \neq b)$$

The DE-9IM Intersection Matrix for the two geometries is:

- T*T***** (für die Zustände Punkt/Linie, Punkt/Fläche und Linie/Fläche)
- T*****T** (für die Zustände Linie/Punkt, Fläche/Punkt und Fläche/Linie)
- 0***** (für den Zustand Linie/Linie)

For any other combination of dimensions this predicate returns false.

The OpenGIS Simple Features Specification defines this predicate only for Point/Line, Point/Area, Line/Line, and Line/Area situations. JTS / GEOS extends the definition to apply to Line/Point, Area/Point and Area/Line situations as well. This makes the relation symmetric.



Important

Do not call with a `GEOMETRYCOLLECTION` as an argument

**Note**

This function call will automatically include a bounding box comparison that will make use of any indexes that are available on the geometries.



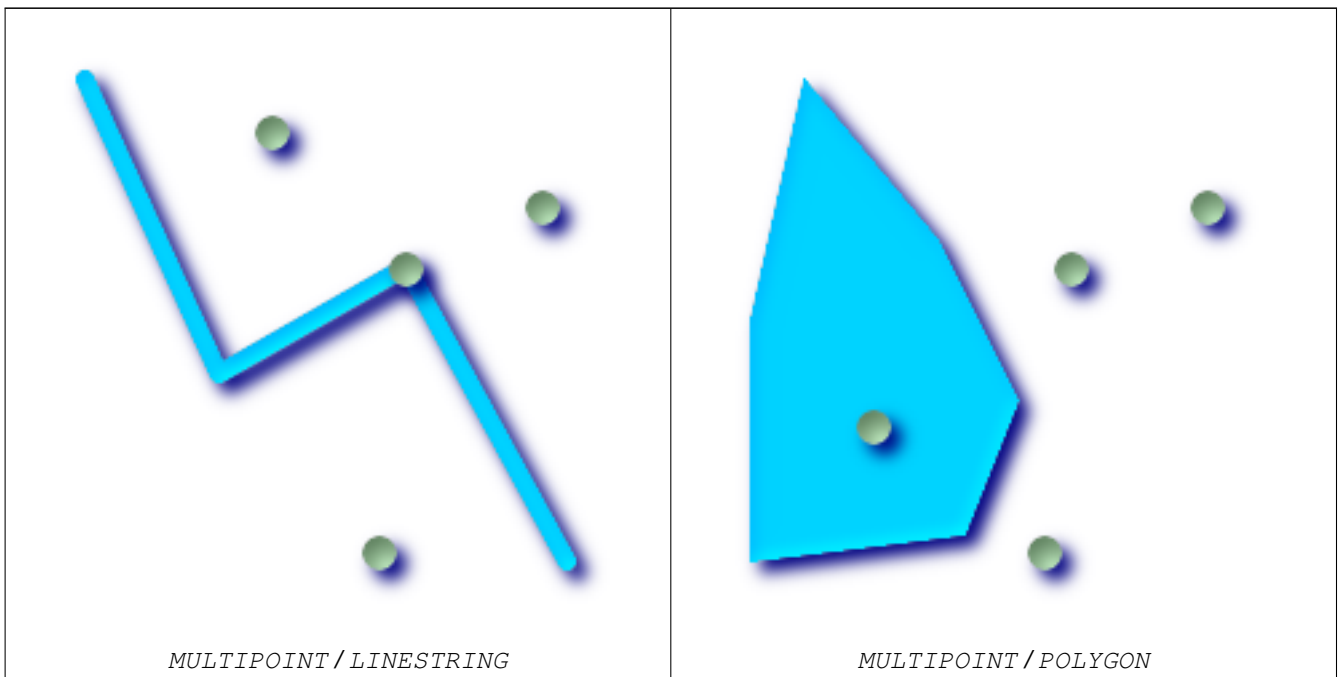
This method implements the [OpenGIS Simple Features Implementation Specification for SQL 1.1](#). s2.1.13.3

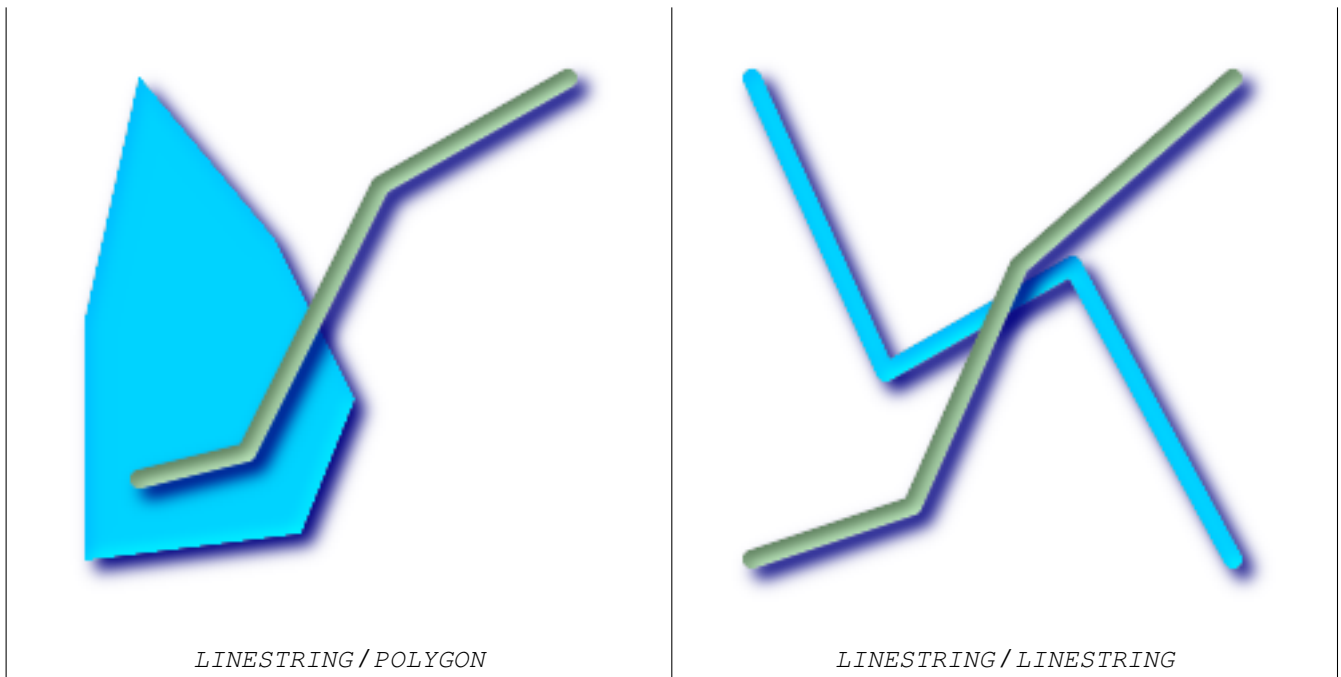


This method implements the SQL/MM specification. SQL-MM 3: 5.1.29

Beispiele

The following illustrations all return TRUE.





Consider a situation where a user has two tables: a table of roads and a table of highways.

```
CREATE TABLE roads (
  id serial NOT NULL,
  the_geom geometry,
  CONSTRAINT roads_pkey PRIMARY KEY ( ↔
    road_id)
);
```

```
CREATE TABLE highways (
  id serial NOT NULL,
  the_gem geometry,
  CONSTRAINT roads_pkey PRIMARY KEY ( ↔
    road_id)
);
```

To determine a list of roads that cross a highway, use a query similiar to:

```
SELECT roads.id
FROM roads, highways
WHERE ST_Crosses(roads.the_geom, highways.the_geom);
```

8.9.22 ST_LineCrossingDirection

ST_LineCrossingDirection — Given 2 linestrings, returns a number between -3 and 3 denoting what kind of crossing behavior. 0 is no crossing.

Synopsis

integer **ST_LineCrossingDirection**(geometry linestringA, geometry linestringB);

Beschreibung

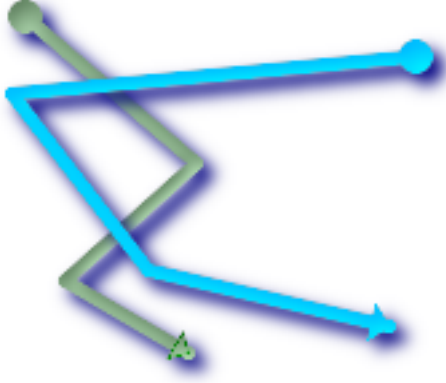
Given 2 linestrings, returns a number between -3 and 3 denoting what kind of crossing behavior. 0 is no crossing. This is only supported for **LINESTRING**

Definition of integer constants is as follows:

- 0: LINIE KEIN KREUZUNGSPUNKT
- -1: LINIE KREUZUNGSPUNKT LINKS
- -1: LINIE KREUZUNGSPUNKT RECHTS
- -2: LINE MULTICROSS END LEFT
- 2: LINE MULTICROSS END RIGHT
- -3: LINE MULTICROSS END SAME FIRST LEFT
- 3: LINE MULTICROSS END SAME FIRST RIGHT

Verfügbarkeit: 1.4

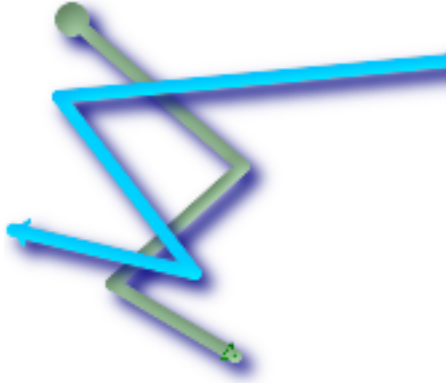
Beispiele



Line 1 (green), Line 2 ball is start point, triangle are end points. Query below.

```
SELECT ST_LineCrossingDirection(foo.line1 ↔  
    , foo.line2) As l1_cross_l2 ,  
       ST_LineCrossingDirection(foo. ↔  
    line2, foo.line1) As l2_cross_l1  
FROM (  
SELECT  
    ST_GeomFromText('LINESTRING(25 169,89 ↔  
    114,40 70,86 43)') As line1,  
    ST_GeomFromText('LINESTRING(171 154,20 ↔  
    140,71 74,161 53)') As line2  
    ) As foo;
```

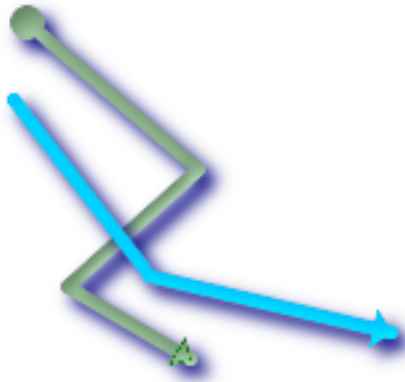
l1_cross_l2	l2_cross_l1
3	-3



Line 1 (green), Line 2 (blue) ball is start point, triangle are end points. Query below.

```
SELECT ST_LineCrossingDirection(foo.line1 ↔  
    , foo.line2) As l1_cross_l2 ,  
       ST_LineCrossingDirection(foo. ↔  
    line2, foo.line1) As l2_cross_l1  
FROM (  
SELECT  
    ST_GeomFromText('LINESTRING(25 169,89 ↔  
    114,40 70,86 43)') As line1,  
    ST_GeomFromText('LINESTRING (171 154, ↔  
    20 140, 71 74, 2.99 90.16)') As line2  
    ) As foo;
```

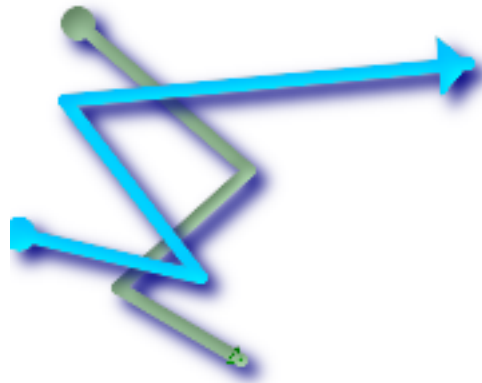
l1_cross_l2	l2_cross_l1
2	-2



Line 1 (green), Line 2 (blue) ball is start point, triangle are end points. Query below.

```
SELECT
    ST_LineCrossingDirection(foo. ↵
        line1, foo.line2) As l1_cross_l2 ,
    ST_LineCrossingDirection(foo. ↵
        line2, foo.line1) As l2_cross_l1
FROM (
    SELECT
        ST_GeomFromText('LINESTRING(25 169,89 ↵
            114,40 70,86 43)') As line1,
        ST_GeomFromText('LINESTRING (20 140, 71 ↵
            74, 161 53)') As line2
    ) As foo;

l1_cross_l2 | l2_cross_l1
-----+-----
-1 | 1
```



Line 1 (green), Line 2 (blue) ball is start point, triangle are end points. Query below.

```
SELECT ST_LineCrossingDirection(foo.line1 ↵
    , foo.line2) As l1_cross_l2 ,
    ST_LineCrossingDirection(foo. ↵
        line2, foo.line1) As l2_cross_l1
FROM (SELECT
    ST_GeomFromText('LINESTRING(25 ↵
        169,89 114,40 70,86 43)') As line1,
    ST_GeomFromText('LINESTRING(2.99 ↵
        90.16,71 74,20 140,171 154)') As line2
    ) As foo;

l1_cross_l2 | l2_cross_l1
-----+-----
-2 | 2
```

```
SELECT s1.gid, s2.gid, ST_LineCrossingDirection(s1.the_geom, s2.the_geom)
    FROM streets s1 CROSS JOIN streets s2 ON (s1.gid != s2.gid AND s1.the_geom && s2. ↵
        the_geom )
WHERE ST_CrossingDirection(s1.the_geom, s2.the_geom)
> 0;
```

Siehe auch

[ST_Crosses](#)

8.9.23 ST_Disjoint

ST_Disjoint — Gibt TRUE zurück, wenn sich die Geometrien nicht "räumlich schneiden" - wenn sie sich keinen gemeinsamen Raum teilen

Synopsis

boolean **ST_Disjoint**(geometry A , geometry B);

Beschreibung

Overlaps, Touches, Within all imply geometries are not spatially disjoint. If any of the aforementioned returns true, then the geometries are not spatially disjoint. Disjoint implies false for spatial intersection.



Important

Do not call with a `GEOMETRYCOLLECTION` as an argument

Wird durch das GEOS Modul ausgeführt



Note

Dieser Funktionsaufruf verwendet keine Indizes



Note

NOTE: this is the "allowable" version that returns a boolean, not an integer.



This method implements the [OpenGIS Simple Features Implementation Specification for SQL 1.1](#). s2.1.1.2 //s2.1.13.3 - a.Relate(b, 'FF*FF****')



This method implements the SQL/MM specification. SQL-MM 3: 5.1.26

Beispiele

```
SELECT ST_Disjoint('POINT(0 0)::geometry, 'LINESTRING ( 2 0, 0 2 ) '::geometry);
st_disjoint
-----
t
(1 row)
SELECT ST_Disjoint('POINT(0 0)::geometry, 'LINESTRING ( 0 0, 0 2 ) '::geometry);
st_disjoint
-----
f
(1 row)
```

Siehe auch

[ST_Intersects](#) [ST_Intersects](#)

8.9.24 ST_Distance

ST_Distance — For geometry type Returns the 2D Cartesian distance between two geometries in projected units (based on spatial ref). For geography type defaults to return minimum geodesic distance between two geographies in meters.

Synopsis

```
float ST_Distance(geometry g1, geometry g2);
float ST_Distance(geography gg1, geography gg2);
float ST_Distance(geography gg1, geography gg2, boolean use_spheroid);
```

Beschreibung

For geometry type returns the minimum 2D Cartesian distance between two geometries in projected units (spatial ref units). For geography type defaults to return the minimum geodesic distance between two geographies in meters. If use_spheroid is false, a faster sphere calculation is used instead of a spheroid.



This method implements the [OpenGIS Simple Features Implementation Specification for SQL 1.1](#).



This method implements the SQL/MM specification. SQL-MM 3: 5.1.23



This method supports Circular Strings and Curves



This method is also provided by SFCGAL backend.

Availability: 1.5.0 geography support was introduced in 1.5. Speed improvements for planar to better handle large or many vertex geometries

Enhanced: 2.1.0 improved speed for geography. See [Making Geography faster](#) for details.

Enhanced: 2.1.0 - support for curved geometries was introduced.

Enhanced: 2.2.0 - measurement on spheroid performed with GeographicLib for improved accuracy and robustness. Requires Proj >= 4.9.0 to take advantage of the new feature.

Standardbeispiele Geometrie

```
--Geometry example - units in planar degrees 4326 is WGS 84 long lat unit=degrees
SELECT ST_Distance(
    ST_GeomFromText('POINT(-72.1235 42.3521)',4326),
    ST_GeomFromText('LINESTRING(-72.1260 42.45, -72.123 42.1546)', 4326)
);
st_distance
-----
0.00150567726382282

-- Geometry example - units in meters (SRID: 26986 Massachusetts state plane meters) (most ←
  accurate for Massachusetts)
SELECT ST_Distance(
    ST_Transform(ST_GeomFromText('POINT(-72.1235 42.3521)',4326),26986) ←
    ,
    ST_Transform(ST_GeomFromText('LINESTRING(-72.1260 42.45, -72.123 ←
    42.1546)', 4326),26986)
);
st_distance
-----
123.797937878454

-- Geometry example - units in meters (SRID: 2163 US National Atlas Equal area) (least ←
  accurate)
SELECT ST_Distance(
    ST_Transform(ST_GeomFromText('POINT(-72.1235 42.3521)',4326),2163),
```

```

        ST_Transform(ST_GeomFromText('LINESTRING(-72.1260 42.45, -72.123 42.1546)', 4326), 2163)
    );

st_distance
-----
126.664256056812

```

Beispiele Geographie

```

-- same as geometry example but note units in meters - use sphere for slightly faster less accurate
SELECT ST_Distance(gg1, gg2) As spheroid_dist, ST_Distance(gg1, gg2, false) As sphere_dist
FROM (SELECT
        ST_GeogFromText('SRID=4326;POINT(-72.1235 42.3521)') As gg1,
        ST_GeogFromText('SRID=4326;LINESTRING(-72.1260 42.45, -72.123 42.1546)') As gg2
    ) As foo ;

spheroid_dist | sphere_dist
-----+-----
123.802076746848 | 123.475736916397

```

Siehe auch

[ST_3DDistance](#), [ST_DWithin](#), [ST_DistanceSphere](#), [ST_DistanceSpheroid](#), [ST_MaxDistance](#), [ST_HausdorffDistance](#), [ST_FrechetDistance](#), [ST_Transform](#)

8.9.25 ST_MinimumClearance

ST_MinimumClearance — Returns the minimum clearance of a geometry, a measure of a geometry's robustness.

Synopsis

```
float ST_MinimumClearance(geometry g);
```

Beschreibung

It is not uncommon to have a geometry that, while meeting the criteria for validity according to `ST_IsValid` (polygons) or `ST_IsSimple` (lines), would become invalid if one of the vertices moved by a slight distance, as can happen during conversion to text-based formats (such as WKT, KML, GML GeoJSON), or binary formats that do not use double-precision floating point coordinates (MapInfo TAB).

A geometry's "minimum clearance" is the smallest distance by which a vertex of the geometry could be moved to produce an invalid geometry. It can be thought of as a quantitative measure of a geometry's robustness, where increasing values of minimum clearance indicate increasing robustness.

If a geometry has a minimum clearance of ϵ , it can be said that:

- No two distinct vertices in the geometry are separated by less than ϵ .
- No vertex is closer than ϵ to a line segment of which it is not an endpoint.

If no minimum clearance exists for a geometry (for example, a single point, or a multipoint whose points are identical), then `ST_MinimumClearance` will return Infinity.

Verfügbarkeit: 2.3.0 - benötigt GEOS $\geq$ 3.6.0

Beispiele

```
SELECT ST_MinimumClearance('POLYGON ((0 0, 1 0, 1 1, 0.5 3.2e-4, 0 0))');
st_minimumclearance
-----
0.00032
```

Siehe auch

[ST_MinimumClearanceLine](#)

8.9.26 ST_MinimumClearanceLine

ST_MinimumClearanceLine — Returns the two-point LineString spanning a geometry's minimum clearance.

Synopsis

Geometry **ST_MinimumClearanceLine**(geometry g);

Beschreibung

Returns the two-point LineString spanning a geometry's minimum clearance. If the geometry does not have a minimum clearance, `LINESTRING EMPTY` will be returned.

Verfügbarkeit: 2.3.0 - benötigt GEOS >= 3.6.0

Beispiele

```
SELECT ST_AsText(ST_MinimumClearanceLine('POLYGON ((0 0, 1 0, 1 1, 0.5 3.2e-4, 0 0))'));
st_astext
-----
LINESTRING(0.5 0.00032,0.5 0)
```

Siehe auch

[ST_MinimumClearance](#)

8.9.27 ST_HausdorffDistance

ST_HausdorffDistance — Returns the Hausdorff distance between two geometries. Basically a measure of how similar or dissimilar 2 geometries are. Units are in the units of the spatial reference system of the geometries.

Synopsis

```
float ST_HausdorffDistance(geometry g1, geometry g2);
float ST_HausdorffDistance(geometry g1, geometry g2, float densifyFrac);
```

Beschreibung

Implements algorithm for computing a distance metric which can be thought of as the "Discrete Hausdorff Distance". This is the Hausdorff distance restricted to discrete points for one of the geometries. [Wikipedia article on Hausdorff distance](#) [Martin Davis note on how Hausdorff Distance calculation was used to prove correctness of the CascadePolygonUnion approach](#).

When densifyFrac is specified, this function performs a segment densification before computing the discrete hausdorff distance. The densifyFrac parameter sets the fraction by which to densify each segment. Each segment will be split into a number of equal-length subsegments, whose fraction of the total length is closest to the given fraction.



Note

The current implementation supports only vertices as the discrete locations. This could be extended to allow an arbitrary density of points to be used.



Note

This algorithm is NOT equivalent to the standard Hausdorff distance. However, it computes an approximation that is correct for a large subset of useful cases. One important part of this subset is Linestrings that are roughly parallel to each other, and roughly equal in length. This is a useful metric for line matching.

Verfügbarkeit: 1.5.0 - benötigt GEOS >= 3.2.0

Beispiele

For each building, find the parcel that best represents it. First we require the parcel intersect with the geometry. DISTINCT ON guarantees we get each building listed only once, the ORDER BY .. ST_HausdorffDistance gives us a preference of parcel that is most similar to the building.

```
SELECT DISTINCT ON(buildings.gid) buildings.gid, parcels.parcel_id
FROM buildings INNER JOIN parcels ON ST_Intersects(buildings.geom, parcels.geom)
ORDER BY buildings.gid, ST_HausdorffDistance(buildings.geom, parcels.geom);
```

```
postgis=# SELECT ST_HausdorffDistance(
              'LINESTRING (0 0, 2 0)::geometry',
              'MULTIPOINT (0 1, 1 0, 2 1)::geometry');
 st_hausdorffdistance
-----
1
(1 row)
```

```
postgis=# SELECT st_hausdorffdistance('LINESTRING (130 0, 0 0, 0 150)::geometry', '↔
LINESTRING (10 10, 10 150, 130 10)::geometry', 0.5);
 st_hausdorffdistance
-----
70
(1 row)
```

Siehe auch

[ST_FrechetDistance](#)

8.9.28 ST_FrechetDistance

ST_FrechetDistance — Returns the Fréchet distance between two geometries. This is a measure of similarity between curves that takes into account the location and ordering of the points along the curves. Units are in the units of the spatial reference system of the geometries.

Synopsis

```
float ST_FrechetDistance(geometry g1, geometry g2, float densifyFrac = -1);
```

Beschreibung

Implements algorithm for computing the Fréchet distance restricted to discrete points for both geometries, based on [Computing Discrete Fréchet Distance](#). The Fréchet distance is a measure of similarity between curves that takes into account the location and ordering of the points along the curves. Therefore it is often better than the Hausdorff distance.

When the optional densifyFrac is specified, this function performs a segment densification before computing the discrete Fréchet distance. The densifyFrac parameter sets the fraction by which to densify each segment. Each segment will be split into a number of equal-length subsegments, whose fraction of the total length is closest to the given fraction.



Note

The current implementation supports only vertices as the discrete locations. This could be extended to allow an arbitrary density of points to be used.



Note

The smaller densifyFrac we specify, the more accurate Fréchet distance we get. But, the computation time and the memory usage increase with the square of the number of subsegments.

Availability: 2.4.0 - requires GEOS >= 3.7.0

Beispiele

```
postgres=# SELECT st_frechetdistance('LINESTRING (0 0, 100 0)::geometry, 'LINESTRING (0 0, ↵
          50 50, 100 0)::geometry');
 st_frechetdistance
-----
          70.7106781186548
(1 row)
```

```
SELECT st_frechetdistance('LINESTRING (0 0, 100 0)::geometry, 'LINESTRING (0 0, 50 50, 100 ↵
          0)::geometry, 0.5);
 st_frechetdistance
-----
                50
(1 row)
```

Siehe auch

[ST_HausdorffDistance](#)

8.9.29 ST_MaxDistance

ST_MaxDistance — Returns the 2-dimensional largest distance between two geometries in projected units.

Synopsis

```
float ST_MaxDistance(geometry g1, geometry g2);
```

Beschreibung



Note

Returns the 2-dimensional maximum distance between two geometries in projected units. If g1 and g2 is the same geometry the function will return the distance between the two vertices most far from each other in that geometry.

Verfügbarkeit: 1.5.0

Beispiele

Basic furthest distance the point is to any part of the line

```
postgis=# SELECT ST_MaxDistance('POINT(0 0)::geometry', 'LINESTRING ( 2 0, 0 2 ) '::geometry ↵
);
      st_maxdistance
-----
2
(1 row)

postgis=# SELECT ST_MaxDistance('POINT(0 0)::geometry', 'LINESTRING ( 2 2, 2 2 ) '::geometry ↵
);
      st_maxdistance
-----
2.82842712474619
(1 row)
```

Siehe auch

[ST_Distance](#), [ST_LongestLine](#), [ST_DFullyWithin](#)

8.9.30 ST_DistanceSphere

ST_DistanceSphere — Returns minimum distance in meters between two lon/lat geometries. Uses a spherical earth and radius derived from the spheroid defined by the SRID. Faster than **ST_DistanceSpheroid** [ST_DistanceSpheroid](#), but less accurate. PostGIS versions prior to 1.5 only implemented for points.

Synopsis

```
float ST_DistanceSphere(geometry geomlonlatA, geometry geomlonlatB);
```

Beschreibung

Returns minimum distance in meters between two lon/lat points. Uses a spherical earth and radius derived from the spheroid defined by the SRID. Faster than [ST_DistanceSpheroid](#), but less accurate. PostGIS Versions prior to 1.5 only implemented for points.

Availability: 1.5 - support for other geometry types besides points was introduced. Prior versions only work with points.

Changed: 2.2.0 In prior versions this used to be called `ST_Distance_Sphere`

Beispiele

```
SELECT round(CAST(ST_DistanceSphere(ST_Centroid(the_geom), ST_GeomFromText('POINT(-118 38) ' ←
',4326)) As numeric),2) As dist_meters,
round(CAST(ST_Distance(ST_Transform(ST_Centroid(the_geom),32611),
ST_Transform(ST_GeomFromText('POINT(-118 38)', 4326),32611)) As numeric),2) As ←
As dist_utm11_meters,
round(CAST(ST_Distance(ST_Centroid(the_geom), ST_GeomFromText('POINT(-118 38)', 4326)) As ←
numeric),5) As dist_degrees,
round(CAST(ST_Distance(ST_Transform(the_geom,32611),
ST_Transform(ST_GeomFromText('POINT(-118 38)', 4326),32611)) As numeric),2) As ←
As min_dist_line_point_meters
FROM
(SELECT ST_GeomFromText('LINESTRING(-118.584 38.374,-118.583 38.5)', 4326) As ←
the_geom) as foo;
dist_meters | dist_utm11_meters | dist_degrees | min_dist_line_point_meters
-----+-----+-----+-----
70424.47 | 70438.00 | 0.72900 | 65871.18
```

Siehe auch

[ST_Distance](#), [ST_DistanceSpheroid](#)

8.9.31 ST_DistanceSpheroid

ST_DistanceSpheroid — Returns the minimum distance between two lon/lat geometries given a particular spheroid. PostGIS versions prior to 1.5 only support points.

Synopsis

```
float ST_DistanceSpheroid(geometry geomlonlatA, geometry geomlonlatB, spheroid measurement_spheroid);
```

Beschreibung

Returns minimum distance in meters between two lon/lat geometries given a particular spheroid. See the explanation of spheroids given for [ST_LengthSpheroid](#). PostGIS version prior to 1.5 only support points.



Note

This function currently does not look at the SRID of a geometry and will always assume its represented in the coordinates of the passed in spheroid. Prior versions of this function only support points.

Availability: 1.5 - support for other geometry types besides points was introduced. Prior versions only work with points.

Changed: 2.2.0 In prior versions this used to be called ST_Distance_Spheroid

Beispiele

```
SELECT round(CAST(
ST_DistanceSpheroid(ST_Centroid(the_geom), ST_GeomFromText('POINT(-118 38) ' ←
',4326), 'SPHEROID["WGS 84",6378137,298.257223563]')
As numeric),2) As dist_meters_spheroid,
```

```

round(CAST(ST_DistanceSphere(ST_Centroid(the_geom), ST_GeomFromText('POINT (-118 38)', 4326)) As numeric), 2) As dist_meters_sphere,
round(CAST(ST_Distance(ST_Transform(ST_Centroid(the_geom), 32611),
ST_Transform(ST_GeomFromText('POINT(-118 38)', 4326), 32611)) As numeric), 2) As dist_utm11_meters
FROM
(SELECT ST_GeomFromText('LINESTRING(-118.584 38.374,-118.583 38.5)', 4326) As
the_geom) as foo;
dist_meters_spheroid | dist_meters_sphere | dist_utm11_meters
-----+-----+-----
70454.92 | 70424.47 | 70438.00

```

Siehe auch

[ST_Distance](#), [ST_DistanceSphere](#)

8.9.32 ST_DFullyWithin

ST_DFullyWithin — Returns true if all of the geometries are within the specified distance of one another

Synopsis

boolean **ST_DFullyWithin**(geometry g1, geometry g2, double precision distance);

Beschreibung

Returns true if the geometries is fully within the specified distance of one another. The distance is specified in units defined by the spatial reference system of the geometries. For this function to make sense, the source geometries must both be of the same coordinate projection, having the same SRID.



Note

This function call will automatically include a bounding box comparison that will make use of any indexes that are available on the geometries.

Verfügbarkeit: 1.5.0

Beispiele

```

postgis=# SELECT ST_DFullyWithin(geom_a, geom_b, 10) as DFullyWithin10, ST_DWithin(geom_a,
geom_b, 10) as DWithin10, ST_DFullyWithin(geom_a, geom_b, 20) as DFullyWithin20 from
(select ST_GeomFromText('POINT(1 1)') as geom_a, ST_GeomFromText('LINESTRING
(1 5, 2 7, 1 9, 14 12)') as geom_b) t1;
-----
DFullyWithin10 | DWithin10 | DFullyWithin20 |
-----+-----+-----+
f              | t         | t               |

```

Siehe auch

[ST_MaxDistance](#), [ST_DWithin](#)

8.9.33 ST_DWithin

ST_DWithin — Returns true if the geometries are within the specified distance of one another. For geometry units are in those of spatial reference and For geography units are in meters and measurement is defaulted to use_spheroid=true (measure around spheroid), for faster check, use_spheroid=false to measure along sphere.

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 ST_DWithin(geography gg1, geography gg2, double precision distance_meters, boolean use_spheroid);
```

Beschreibung

Returns true if the geometries are within the specified distance of one another.

For Geometries: The distance is specified in units defined by the spatial reference system of the geometries. For this function to make sense, the source geometries must both be of the same coordinate projection, having the same SRID.

For geography units are in meters and measurement is defaulted to use_spheroid=true, for faster check, use_spheroid=false to measure along sphere.



Note

This function call will automatically include a bounding box comparison that will make use of any indexes that are available on the geometries.



Note

Prior to 1.3, ST_Expand was commonly used in conjunction with && and ST_Distance to achieve the same effect and in pre-1.3.4 this function was basically short-hand for that construct. From 1.3.4, ST_DWithin uses a more short-circuit distance function which should make it more efficient than prior versions for larger buffer regions.



Note

Use ST_3DDWithin if you have 3D geometries.



This method implements the [OpenGIS Simple Features Implementation Specification for SQL 1.1](#).

Verfügbarkeit: Mit 1.5.0 wurde die Unterstützung von geographischen Koordinaten eingeführt

Enhanced: 2.1.0 improved speed for geography. See [Making Geography faster](#) for details.

Enhanced: 2.1.0 support for curved geometries was introduced.

Beispiele

```
--Find the nearest hospital to each school
--that is within 3000 units of the school.
-- We do an ST_DWithin search to utilize indexes to limit our search list
-- that the non-indexable ST_Distance needs to process
--If the units of the spatial reference is meters then units would be meters
SELECT DISTINCT ON (s.gid) s.gid, s.school_name, s.the_geom, h.hospital_name
```

```

        FROM schools s
            LEFT JOIN hospitals h ON ST_DWithin(s.the_geom, h.the_geom, 3000)
        ORDER BY s.gid, ST_Distance(s.the_geom, h.the_geom);

--The schools with no close hospitals
--Find all schools with no hospital within 3000 units
--away from the school.  Units is in units of spatial ref (e.g. meters, feet, degrees)
SELECT s.gid, s.school_name
        FROM schools s
            LEFT JOIN hospitals h ON ST_DWithin(s.the_geom, h.the_geom, 3000)
        WHERE h.gid IS NULL;

```

Siehe auch

[ST_Distance](#), [ST_Expand](#)

8.9.34 ST_Equals

ST_Equals — Returns true if the given geometries represent the same geometry. Directionality is ignored.

Synopsis

boolean **ST_Equals**(geometry A, geometry B);

Beschreibung

Returns TRUE if the given Geometries are "spatially equal". Use this for a 'better' answer than '='. Note by spatially equal we mean ST_Within(A,B) = true and ST_Within(B,A) = true and also mean ordering of points can be different but represent the same geometry structure. To verify the order of points is consistent, use ST_OrderingEquals (it must be noted ST_OrderingEquals is a little more stringent than simply verifying order of points are the same).



Important

This function will return false if either geometry is invalid except in the case where they are binary equal.



Important

Do not call with a GEOMETRYCOLLECTION as an argument.



This method implements the [OpenGIS Simple Features Implementation Specification for SQL 1.1](#). s2.1.1.2



This method implements the SQL/MM specification. SQL-MM 3: 5.1.24

Changed: 2.2.0 Returns true even for invalid geometries if they are binary equal

Beispiele

```
SELECT ST_Equals(ST_GeomFromText('LINESTRING(0 0, 10 10)'),
                 ST_GeomFromText('LINESTRING(0 0, 5 5, 10 10)'));

 st_equals
-----
t
(1 row)

SELECT ST_Equals(ST_Reverse(ST_GeomFromText('LINESTRING(0 0, 10 10)'),
                        ST_GeomFromText('LINESTRING(0 0, 5 5, 10 10)'));

 st_equals
-----
t
(1 row)
```

Siehe auch

[ST_IsValid](#), [ST_OrderingEquals](#), [ST_Reverse](#), [ST_Within](#)

8.9.35 ST_GeometricMedian

ST_GeometricMedian — Returns the geometric median of a MultiPoint.

Synopsis

geometry **ST_GeometricMedian** (geometry g , float8 tolerance , int max_iter , boolean fail_if_not_converged);

Beschreibung

Computes the approximate geometric median of a MultiPoint geometry using the Weiszfeld algorithm. The geometric median provides a centrality measure that is less sensitive to outlier points than the centroid. The algorithm will iterate until the distance change between successive iterations is less than the supplied *tolerance* parameter. If this condition has not been met after *max_iterations* iterations, the function will produce an error and exit, unless *fail_if_not_converged* is set to false. If a tolerance value is not provided, a default tolerance value will be calculated based on the extent of the input geometry.

Verfügbarkeit: 2.3.0



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

Beispiele



Comparison of the centroid (turquoise point) and geometric median (red point) of a four-point MultiPoint (yellow points).

```
WITH test AS (  
SELECT 'MULTIPOINT((0 0), (1 1), (2 2), (200 200))'::geometry geom)  
SELECT  
  ST_AsText(ST_Centroid(geom)) centroid,  
  ST_AsText(ST_GeometricMedian(geom)) median  
FROM test;  
  centroid      |      median  
-----+-----  
POINT(50.75 50.75) | POINT(1.9761550281255 1.9761550281255)  
(1 row)
```

Siehe auch

[ST_Centroid](#)

8.9.36 ST_HasArc

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

Synopsis

boolean **ST_HasArc**(geometry geomA);

Beschreibung

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

Verfügbarkeit: 1.2.3?

- ✓ This function supports 3d and will not drop the z-index.
- ✓ This method supports Circular Strings and Curves

Beispiele

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

Siehe auch

[ST_CurveToLine](#), [ST_LineToCurve](#)

8.9.37 ST_Intersects

ST_Intersects — Returns TRUE if the Geometries/Geography "spatially intersect in 2D" - (share any portion of space) and FALSE if they don't (they are Disjoint). For geography -- tolerance is 0.00001 meters (so any points that close are considered to intersect)

Synopsis

```
boolean ST_Intersects( geometry geomA , geometry geomB );
boolean ST_Intersects( geography geogA , geography geogB );
```

Beschreibung

If a geometry or geography shares any portion of space then they intersect. For geography -- tolerance is 0.00001 meters (so any points that are close are considered to intersect)

Overlaps, Touches, Within all imply spatial intersection. If any of the aforementioned returns true, then the geometries also spatially intersect. Disjoint implies false for spatial intersection.



Important

Do not call with a `GEOMETRYCOLLECTION` as an argument for geometry version. The geography version supports `GEOMETRYCOLLECTION` since its a thin wrapper around distance implementation.

Enhanced: 2.3.0 Enhancement to PIP short-circuit extended to support MultiPoints with few points. Prior versions only supported point in polygon.

Performed by the GEOS module (for geometry), geography is native

Verfügbarkeit: 1.5 - Unterstützung von geographischen Koordinaten.



Note

This function call will automatically include a bounding box comparison that will make use of any indexes that are available on the geometries.



Note

For geography, this function has a distance tolerance of about 0.00001 meters and uses the sphere rather than spheroid calculation.

**Note**

NOTE: this is the "allowable" version that returns a boolean, not an integer.



This method implements the [OpenGIS Simple Features Implementation Specification for SQL 1.1](#). s2.1.1.2 //s2.1.13.3 - ST_Intersects(g1, g2) --> Not (ST_Disjoint(g1, g2))



This method implements the SQL/MM specification. SQL-MM 3: 5.1.27



This method is also provided by SFCGAL backend.

Geometrie Beispiele

```
SELECT ST_Intersects('POINT(0 0)::geometry, 'LINESTRING ( 2 0, 0 2 ) '::geometry);
st_intersects
-----
f
(1 row)
SELECT ST_Intersects('POINT(0 0)::geometry, 'LINESTRING ( 0 0, 0 2 ) '::geometry);
st_intersects
-----
t
(1 row)
```

Beispiele Geographie

```
SELECT ST_Intersects(
    ST_GeographyFromText('SRID=4326;LINESTRING(-43.23456 72.4567,-43.23456 72.4568)'),
    ST_GeographyFromText('SRID=4326;POINT(-43.23456 72.4567772)')
);

st_intersects
-----
t
```

Siehe auch

[ST_3DIntersects](#), [ST_Disjoint](#)

8.9.38 ST_Length

ST_Length — Returns the 2D length of the geometry if it is a LineString or MultiLineString. geometry are in units of spatial reference and geography are in meters (default spheroid)

Synopsis

```
float ST_Length(geometry a_2dlinestring);
float ST_Length(geography geog, boolean use_spheroid=true);
```


Beschreibung

For geometry: Returns the 2D Cartesian length of the geometry if it is a LineString, MultiLineString, ST_Curve, ST_MultiCurve. 0 is returned for areal geometries. For areal geometries use **ST_Perimeter**. For geometry types, units for length measures are specified by the spatial reference system of the geometry.

For geography types, the calculations are performed using the inverse geodesic problem, where length units are in meters. If PostGIS is compiled with PROJ version 4.8.0 or later, the spheroid is specified by the SRID, otherwise it is exclusive to WGS84. If `use_spheroid=false`, then calculations will approximate a sphere instead of a spheroid.

Currently for geometry this is an alias for ST_Length2D, but this may change to support higher dimensions.



Warning

Changed: 2.0.0 Breaking change -- in prior versions applying this to a MULTI/POLYGON of type geography would give you the perimeter of the POLYGON/MULTIPOLYGON. In 2.0.0 this was changed to return 0 to be in line with geometry behavior. Please use ST_Perimeter if you want the perimeter of a polygon



Note

For geography measurement defaults spheroid measurement. To use the faster less accurate sphere use ST_Length(gg,false);



This method implements the [OpenGIS Simple Features Implementation Specification for SQL 1.1](#). s2.1.5.1



This method implements the SQL/MM specification. SQL-MM 3: 7.1.2, 9.3.4

Verfügbarkeit: 1.5.0 Unterstützung von geographischen Koordinaten.



This method is also provided by SFCGAL backend.

Geometrie Beispiele

Return length in feet for line string. Note this is in feet because EPSG:2249 is Massachusetts State Plane Feet

```
SELECT ST_Length(ST_GeomFromText('LINESTRING(743238 2967416,743238 2967450,743265 2967450,
743265.625 2967416,743238 2967416)',2249));
st_length
-----
122.630744000095

--Koordinatentransformation eines Linienzuges von "WGS 84" nach "Massachusetts state plane ←
meters"
SELECT ST_Length(
    ST_Transform(
        ST_GeomFromEWKT('SRID=4326;LINESTRING(-72.1260 42.45, -72.1240 42.45666, ←
        -72.123 42.1546)'),
        26986
    )
);
st_length
-----
34309.4563576191
```

Beispiele Geographie

Return length of WGS 84 geography line

```
-- default calculation is using a sphere rather than spheroid
SELECT ST_Length(the_geog) As length_spheroid, ST_Length(the_geog,false) As length_sphere
FROM (SELECT ST_GeographyFromText(
'SRID=4326;LINESTRING(-72.1260 42.45, -72.1240 42.45666, -72.123 42.1546)') As the_geog)
As foo;
length_spheroid | length_sphere
-----+-----
34310.5703627288 | 34346.2060960742
```

Siehe auch

[ST_GeographyFromText](#), [ST_GeomFromEWKT](#), [ST_LengthSpheroid](#), [ST_Perimeter](#), [ST_Transform](#)

8.9.39 ST_Length2D

ST_Length2D — Returns the 2-dimensional length of the geometry if it is a linestring or multi-linestring. This is an alias for `ST_Length`

Synopsis

```
float ST_Length2D(geometry a_2dlinestring);
```

Beschreibung

Returns the 2-dimensional length of the geometry if it is a linestring or multi-linestring. This is an alias for `ST_Length`

Siehe auch

[ST_Length](#), [ST_3DLength](#)

8.9.40 ST_3DLength

ST_3DLength — Returns the 3-dimensional or 2-dimensional length of the geometry if it is a linestring or multi-linestring.

Synopsis

```
float ST_3DLength(geometry a_3dlinestring);
```

Beschreibung

Returns the 3-dimensional or 2-dimensional length of the geometry if it is a linestring or multi-linestring. For 2-d lines it will just return the 2-d length (same as `ST_Length` and `ST_Length2D`)



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

Changed: 2.0.0 In prior versions this used to be called `ST_Length3D`

Beispiele

Return length in feet for a 3D cable. Note this is in feet because EPSG:2249 is Massachusetts State Plane Feet

```
SELECT ST_3DLength(ST_GeomFromText('LINESTRING(743238 2967416 1,743238 2967450 1,743265 2967450 3,743265.625 2967416 3,743238 2967416 3)',2249));
ST_3DLength
-----
122.704716741457
```

Siehe auch

[ST_Length](#), [ST_Length2D](#)

8.9.41 ST_LengthSpheroid

ST_LengthSpheroid — Calculates the 2D or 3D length/perimeter of a geometry on an ellipsoid. This is useful if the coordinates of the geometry are in longitude/latitude and a length is desired without reprojection.

Synopsis

```
float ST_LengthSpheroid(geometry a_geometry, spheroid a_spheroid);
```

Beschreibung

Calculates the length/perimeter of a geometry on an ellipsoid. This is useful if the coordinates of the geometry are in longitude/latitude and a length is desired without reprojection. The ellipsoid is a separate database type and can be constructed as follows:

```
SPHEROID [<NAME>, <SEMI-MAJOR AXIS>, <INVERSE FLATTENING>]
```

```
SPHEROID["GRS_1980",6378137,298.257222101]
```

Verfügbarkeit: 1.2.2

Changed: 2.2.0 In prior versions this used to be called `ST_Length_Spheroid` and used to have a `ST_3DLength_Spheroid` alias



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

Beispiele

```
SELECT ST_LengthSpheroid( geometry_column,
                          'SPHEROID["GRS_1980",6378137,298.257222101]' )
FROM geometry_table;

SELECT ST_LengthSpheroid( the_geom, sph_m ) As tot_len,
ST_LengthSpheroid(ST_GeometryN(the_geom,1), sph_m) As len_line1,
ST_LengthSpheroid(ST_GeometryN(the_geom,2), sph_m) As len_line2
FROM (SELECT ST_GeomFromText('MULTILINESTRING((-118.584 38.374,-118.583 38.5),
(-71.05957 42.3589 , -71.061 43))') As the_geom,
CAST('SPHEROID["GRS_1980",6378137,298.257222101]' As spheroid) As sph_m) as foo;
tot_len | len_line1 | len_line2
```

```

-----+-----+-----
85204.5207562955 | 13986.8725229309 | 71217.6482333646

--3D
SELECT ST_LengthSpheroid( the_geom, sph_m ) As tot_len,
ST_LengthSpheroid(ST_GeometryN(the_geom,1), sph_m) As len_line1,
ST_LengthSpheroid(ST_GeometryN(the_geom,2), sph_m) As len_line2
      FROM (SELECT ST_GeomFromEWKT('MULTILINESTRING((-118.584 38.374 ↵
      20,-118.583 38.5 30),
      (-71.05957 42.3589 75, -71.061 43 90))') As the_geom,
CAST('SPHEROID["GRS_1980",6378137,298.257222101]' As spheroid) As sph_m) as foo;

      tot_len      |      len_line1      |      len_line2
-----+-----+-----
85204.5259107402 | 13986.876097711 | 71217.6498130292

```

Siehe auch

[ST_GeometryN](#), [ST_Length](#)

8.9.42 ST_Length2D_Spheroid

ST_Length2D_Spheroid — Calculates the 2D length/perimeter of a geometry on an ellipsoid. This is useful if the coordinates of the geometry are in longitude/latitude and a length is desired without reprojection.

Synopsis

```
float ST_Length2D_Spheroid(geometry a_geometry, spheroid a_spheroid);
```

Beschreibung

Calculates the 2D length/perimeter of a geometry on an ellipsoid. This is useful if the coordinates of the geometry are in longitude/latitude and a length is desired without reprojection. The ellipsoid is a separate database type and can be constructed as follows:

```
SPHEROID [<NAME>, <SEMI-MAJOR AXIS>, <INVERSE FLATTENING>]
```

```
SPHEROID ["GRS_1980", 6378137, 298.257222101]
```



Note

This is much like [ST_LengthSpheroid](#) except it will ignore the Z ordinate in calculations.

Beispiele

```

SELECT ST_Length2D_Spheroid( geometry_column,
                             'SPHEROID["GRS_1980",6378137,298.257222101]' )
      FROM geometry_table;

SELECT ST_Length2D_Spheroid( the_geom, sph_m ) As tot_len,
ST_Length2D_Spheroid(ST_GeometryN(the_geom,1), sph_m) As len_line1,

```

```

ST_Length2D_Spheroid(ST_GeometryN(the_geom,2), sph_m) As len_line2
      FROM (SELECT ST_GeomFromText('MULTILINESTRING((-118.584  ←
      38.374,-118.583 38.5),
      (-71.05957 42.3589 , -71.061 43)))') As the_geom,
CAST('SPHEROID["GRS_1980",6378137,298.257222101]' As spheroid) As sph_m) as foo;
      tot_len      |      len_line1      |      len_line2
-----+-----+-----
85204.5207562955 | 13986.8725229309 | 71217.6482333646

--3D Observe same answer
SELECT ST_Length2D_Spheroid( the_geom, sph_m ) As tot_len,
ST_Length2D_Spheroid(ST_GeometryN(the_geom,1), sph_m) As len_line1,
ST_Length2D_Spheroid(ST_GeometryN(the_geom,2), sph_m) As len_line2
      FROM (SELECT ST_GeomFromEWKT('MULTILINESTRING((-118.584 38.374  ←
      20,-118.583 38.5 30),
      (-71.05957 42.3589 75, -71.061 43 90)))') As the_geom,
CAST('SPHEROID["GRS_1980",6378137,298.257222101]' As spheroid) As sph_m) as foo;

      tot_len      |      len_line1      |      len_line2
-----+-----+-----
85204.5207562955 | 13986.8725229309 | 71217.6482333646

```

Siehe auch

[ST_GeometryN](#), [ST_LengthSpheroid](#)

8.9.43 ST_LongestLine

ST_LongestLine — Returns the 2-dimensional longest line points of two geometries. The function will only return the first longest line if more than one, that the function finds. The line returned will always start in g1 and end in g2. The length of the line this function returns will always be the same as `st_maxdistance` returns for g1 and g2.

Synopsis

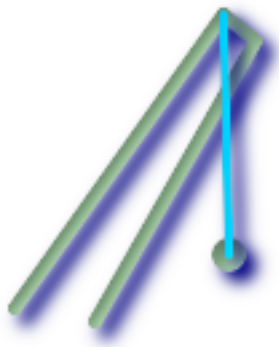
```
geometry ST_LongestLine(geometry g1, geometry g2);
```

Beschreibung

Returns the 2-dimensional longest line between the points of two geometries.

Verfügbarkeit: 1.5.0

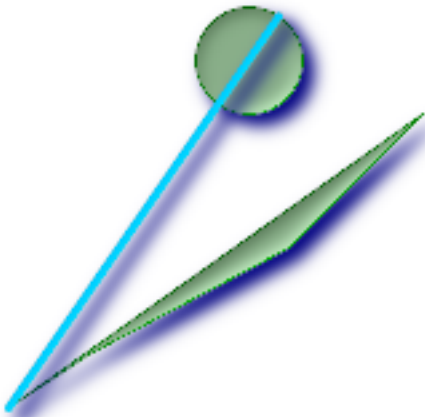
Beispiele



Längste Strecke zwischen Punkt und Linie

```
SELECT ST_AsText (
  ST_LongestLine('POINT(100 100) ':: geometry,
    'LINESTRING (20 80, 98 190, 110 180, 50 75 ) '::geometry)
) As lline;

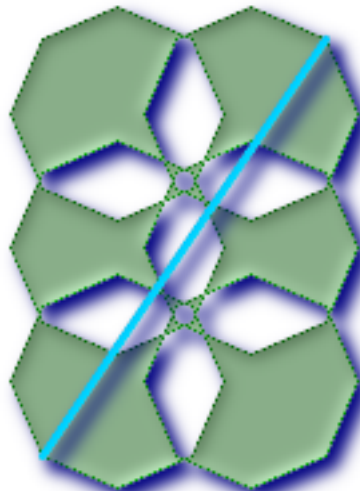
lline
-----
LINESTRING(100 100,98 190)
```



longest line between polygon and polygon

```
SELECT ST_AsText (
  ST_LongestLine(
    ST_GeomFromText ('POLYGON ((175 150, 20 40, 50 60, 125 100, 175 150))'),
    ST_Buffer(ST_GeomFromText ('POINT(110 170)'), 20)
  ) As llinewkt;

lline
-----
LINESTRING(20 40,121.111404660392 186.629392246051)
```



longest straight distance to travel from one part of an elegant city to the other Note the max distance = to the length of the line.

```
SELECT ST_AsText(ST_LongestLine(c.the_geom, c.the_geom)) As llinewkt,
       ST_MaxDistance(c.the_geom,c.the_geom) As max_dist,
       ST_Length(ST_LongestLine(c.the_geom, c.the_geom)) As lenll
FROM (SELECT ST_BuildArea(ST_Collect(the_geom)) As the_geom
      FROM (SELECT ST_Translate(ST_SnapToGrid(ST_Buffer(ST_Point(50 ,generate_series ↵
(50,190, 50)
                               ),40, 'quad_segs=2'),1), x, 0) As the_geom
            FROM generate_series(1,100,50) As x) AS foo
) As c;
```

llinewkt	max_dist	lenll
LINESTRING(23 22,129 178)	188.605408193933	188.605408193933

Siehe auch

[ST_MaxDistance](#), [ST_ShortestLine](#), [ST_LongestLine](#)

8.9.44 ST_OrderingEquals

ST_OrderingEquals — Returns true if the given geometries represent the same geometry and points are in the same directional order.

Synopsis

boolean **ST_OrderingEquals**(geometry A, geometry B);

Beschreibung

ST_OrderingEquals compares two geometries and returns t (TRUE) if the geometries are equal and the coordinates are in the same order; otherwise it returns f (FALSE).

**Note**

This function is implemented as per the ArcSDE SQL specification rather than SQL-MM. http://edndoc.esri.com/arcsde/9.1/sql_api/sqlapi3.htm#ST_OrderingEquals



This method implements the SQL/MM specification. SQL-MM 3: 5.1.43

Beispiele

```
SELECT ST_OrderingEquals(ST_GeomFromText('LINESTRING(0 0, 10 10)'),
                        ST_GeomFromText('LINESTRING(0 0, 5 5, 10 10)'));
 st_orderingequals
-----
f
(1 row)

SELECT ST_OrderingEquals(ST_GeomFromText('LINESTRING(0 0, 10 10)'),
                        ST_GeomFromText('LINESTRING(0 0, 0 0, 10 10)'));
 st_orderingequals
-----
t
(1 row)

SELECT ST_OrderingEquals(ST_Reverse(ST_GeomFromText('LINESTRING(0 0, 10 10)'),
                        ST_GeomFromText('LINESTRING(0 0, 0 0, 10 10)'));
 st_orderingequals
-----
f
(1 row)
```

Siehe auch

[ST_Equals](#), [ST_Reverse](#)

8.9.45 ST_Overlaps

ST_Overlaps — Returns TRUE if the Geometries share space, are of the same dimension, but are not completely contained by each other.

Synopsis

boolean **ST_Overlaps**(geometry A, geometry B);

Beschreibung

Returns TRUE if the Geometries "spatially overlap". By that we mean they intersect, but one does not completely contain another. Wird durch das GEOS Modul ausgeführt

**Note**

Bitte nicht mit dem Argument "GeometryCollection" aufrufen

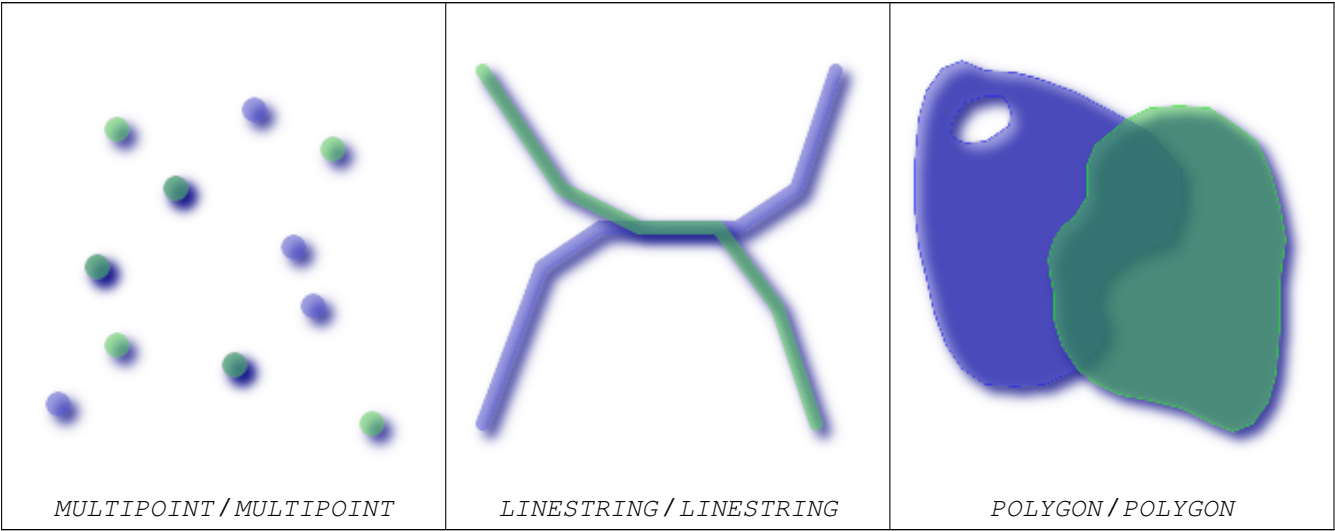
This function call will automatically include a bounding box comparison that will make use of any indexes that are available on the geometries. To avoid index use, use the function `_ST_Overlaps`.

NOTE: this is the "allowable" version that returns a boolean, not an integer.

- ✓ This method implements the [OpenGIS Simple Features Implementation Specification for SQL 1.1](#). s2.1.1.2 // s2.1.13.3
- ✓ This method implements the SQL/MM specification. SQL-MM 3: 5.1.32

Beispiele

The following illustrations all return TRUE.



```
--a point on a line is contained by the line and is of a lower dimension, and therefore ↵
  does not overlap the line
                                nor crosses

SELECT ST_Overlaps(a,b) As a_overlap_b,
       ST_Crosses(a,b) As a_crosses_b,
       ST_Intersects(a, b) As a_intersects_b, ST_Contains(b,a) As b_contains_a
FROM (SELECT ST_GeomFromText('POINT(1 0.5)') As a, ST_GeomFromText('LINESTRING(1 0, 1 1, 3 ↵
  5)') As b)
     As foo

a_overlap_b | a_crosses_b | a_intersects_b | b_contains_a
-----+-----+-----+-----
f           | f           | t              | t

--a line that is partly contained by circle, but not fully is defined as intersecting and ↵
  crossing,
-- but since of different dimension it does not overlap
SELECT ST_Overlaps(a,b) As a_overlap_b, ST_Crosses(a,b) As a_crosses_b,
       ST_Intersects(a, b) As a_intersects_b,
       ST_Contains(a,b) As a_contains_b
FROM (SELECT ST_Buffer(ST_GeomFromText('POINT(1 0.5)'), 3) As a, ST_GeomFromText(' ↵
  LINESTRING(1 0, 1 1, 3 5)') As b)
     As foo;

a_overlap_b | a_crosses_b | a_intersects_b | a_contains_b
-----+-----+-----+-----
```

```
f          | t          | t          | f

-- a 2-dimensional bent hot dog (aka buffered line string) that intersects a circle,
--      but is not fully contained by the circle is defined as overlapping since they ↵
--      are of the same dimension,
--      but it does not cross, because the intersection of the 2 is of the same dimension
--      as the maximum dimension of the 2

SELECT ST_Overlaps(a,b) As a_overlap_b, ST_Crosses(a,b) As a_crosses_b, ST_Intersects(a, b) ↵
      As a_intersects_b,
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 ↵
      dima_intersection_b
FROM (SELECT ST_Buffer(ST_GeomFromText('POINT(1 0.5)'), 3) As a,
      ST_Buffer(ST_GeomFromText('LINESTRING(1 0, 1 1, 3 5)'),0.5) As b)
      As foo;

a_overlap_b | a_crosses_b | a_intersects_b | b_contains_a | dim_a | dim_b | ↵
-----+-----+-----+-----+-----+-----+-----
t          | f          | t          | f          | 2    | 2    | 2
```

Siehe auch

[ST_Contains](#), [ST_Crosses](#), [ST_Dimension](#), [ST_Intersects](#)

8.9.46 ST_Perimeter

ST_Perimeter — Return the length measurement of the boundary of an ST_Surface or ST_MultiSurface geometry or geography. (Polygon, MultiPolygon). geometry measurement is in units of spatial reference and geography is in meters.

Synopsis

```
float ST_Perimeter(geometry g1);
float ST_Perimeter(geography geog, boolean use_spheroid=true);
```

Beschreibung

Returns the 2D perimeter of the geometry/geography if it is a ST_Surface, ST_MultiSurface (Polygon, MultiPolygon). 0 is returned for non-areal geometries. For linear geometries use [ST_Length](#). For geometry types, units for perimeter measures are specified by the spatial reference system of the geometry.

For geography types, the calculations are performed using the inverse geodesic problem, where perimeter units are in meters. If PostGIS is compiled with PROJ version 4.8.0 or later, the spheroid is specified by the SRID, otherwise it is exclusive to WGS84. If use_spheroid=false, then calculations will approximate a sphere instead of a spheroid.

Currently this is an alias for ST_Perimeter2D, but this may change to support higher dimensions.

 This method implements the [OpenGIS Simple Features Implementation Specification for SQL 1.1](#). s2.1.5.1

 This method implements the SQL/MM specification. SQL-MM 3: 8.1.3, 9.5.4

Verfügbarkeit: Mit 2.0.0 wurde die Unterstützung für geographischen Koordinaten eingeführt

Beispiele: Geometrie

Return perimeter in feet for Polygon and MultiPolygon. Note this is in feet because EPSG:2249 is Massachusetts State Plane Feet

```
SELECT ST_Perimeter(ST_GeomFromText('POLYGON((743238 2967416,743238 2967450,743265 2967450,
743265.625 2967416,743238 2967416))', 2249));
st_perimeter
-----
122.630744000095
(1 row)

SELECT ST_Perimeter(ST_GeomFromText('MULTIPOLYGON(((763104.471273676 2949418.44119003,
763104.477769673 2949418.42538203,
763104.189609677 2949418.22343004,763104.471273676 2949418.44119003))),
((763104.471273676 2949418.44119003,763095.804579742 2949436.33850239,
763086.132105649 2949451.46730207,763078.452329651 2949462.11549407,
763075.354136904 2949466.17407812,763064.362142565 2949477.64291974,
763059.953961626 2949481.28983009,762994.637609571 2949532.04103014,
762990.568508415 2949535.06640477,762986.710889563 2949539.61421415,
763117.237897679 2949709.50493431,763235.236617789 2949617.95619822,
763287.718121842 2949562.20592617,763111.553321674 2949423.91664605,
763104.471273676 2949418.44119003)))', 2249));
st_perimeter
-----
845.227713366825
(1 row)
```

Beispiele: Geographie

Return perimeter in meters and feet for Polygon and MultiPolygon. Note this is geography (WGS 84 long lat)

```
SELECT ST_Perimeter(geog) As per_meters, ST_Perimeter(geog)/0.3048 As per_ft
FROM ST_GeogFromText('POLYGON((-71.1776848522251 42.3902896512902,-71.1776843766326 ↵
42.3903829478009,
-71.1775844305465 42.3903826677917,-71.1775825927231 42.3902893647987,-71.1776848522251 ↵
42.3902896512902)))' As geog;

per_meters | per_ft
-----+-----
37.3790462565251 | 122.634666195949

-- Beispiel MultiPolygon --
SELECT ST_Perimeter(geog) As per_meters, ST_Perimeter(geog,false) As per_sphere_meters, ↵
ST_Perimeter(geog)/0.3048 As per_ft
FROM ST_GeogFromText('MULTIPOLYGON(((71.1044543107478 42.340674480411,-71.1044542869917 ↵
42.3406744369506,
-71.1044553562977 42.340673886454,-71.1044543107478 42.340674480411))),
((-71.1044543107478 42.340674480411,-71.1044860600303 42.3407237015564,-71.1045215770124 ↵
42.3407653385914,
-71.1045498002983 42.3407946553165,-71.1045611902745 42.3408058316308,-71.1046016507427 ↵
42.340837442371,
-71.104617893173 42.3408475056957,-71.1048586153981 42.3409875993595,-71.1048736143677 ↵
42.3409959528211,
-71.1048878050242 42.3410084812078,-71.1044020965803 42.3414730072048,
-71.1039672113619 42.3412202916693,-71.1037740497748 42.3410666421308,
-71.1044280218456 42.3406894151355,-71.1044543107478 42.340674480411)))' As geog;

per_meters | per_sphere_meters | per_ft
-----+-----+-----
```

```
257.634283683311 | 257.412311446337 | 845.256836231335
```

Siehe auch

[ST_GeogFromText](#), [ST_GeomFromText](#), [ST_Length](#)

8.9.47 ST_Perimeter2D

ST_Perimeter2D — Returns the 2-dimensional perimeter of the geometry, if it is a polygon or multi-polygon. This is currently an alias for **ST_Perimeter**.

Synopsis

```
float ST_Perimeter2D(geometry geomA);
```

Beschreibung

Returns the 2-dimensional perimeter of the geometry, if it is a polygon or multi-polygon.



Note

This is currently an alias for **ST_Perimeter**. In future versions **ST_Perimeter** may return the highest dimension perimeter for a geometry. This is still under consideration

Siehe auch

[ST_Perimeter](#)

8.9.48 ST_3DPerimeter

ST_3DPerimeter — Returns the 3-dimensional perimeter of the geometry, if it is a polygon or multi-polygon.

Synopsis

```
float ST_3DPerimeter(geometry geomA);
```

Beschreibung

Returns the 3-dimensional perimeter of the geometry, if it is a polygon or multi-polygon. If the geometry is 2-dimensional, then the 2-dimensional perimeter is returned.



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

Changed: 2.0.0 In prior versions this used to be called **ST_Perimeter3D**

Beispiele

Perimeter of a slightly elevated polygon in the air in Massachusetts state plane feet

```
SELECT ST_3DPerimeter(the_geom), ST_Perimeter2d(the_geom), ST_Perimeter(the_geom) FROM
      (SELECT ST_GeomFromEWKT('SRID=2249;POLYGON((743238 2967416 2,743238 ↵
                2967450 1,
743265.625 2967416 1,743238 2967416 2))') As the_geom) As foo;
```

ST_3DPerimeter	st_perimeter2d	st_perimeter
105.465793597674	105.432997272188	105.432997272188

Siehe auch

[ST_GeomFromEWKT](#), [ST_Perimeter](#), [ST_Perimeter2D](#)

8.9.49 ST_PointOnSurface

ST_PointOnSurface — Returns a `POINT` guaranteed to lie on the surface.

Synopsis

geometry **ST_PointOnSurface**(geometry g1);

Beschreibung

Returns a `POINT` guaranteed to intersect a surface.



This method implements the [OpenGIS Simple Features Implementation Specification for SQL 1.1](#). s3.2.14.2 // s3.2.18.2



This method implements the SQL/MM specification. SQL-MM 3: 8.1.5, 9.5.6. According to the specs, `ST_PointOnSurface` works for surface geometries (`POLYGONS`, `MULTIPOLYGONS`, `CURVED POLYGONS`). So PostGIS seems to be extending what the spec allows here. Most databases Oracle, DB II, ESRI SDE seem to only support this function for surfaces. SQL Server 2008 like PostGIS supports for all common geometries.



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

Beispiele

```
SELECT ST_AsText(ST_PointOnSurface('POINT(0 5)::geometry'));
 st_astext
-----
POINT(0 5)
(1 row)

SELECT ST_AsText(ST_PointOnSurface('LINESTRING(0 5, 0 10)::geometry'));
 st_astext
-----
POINT(0 5)
(1 row)
```

```

SELECT ST_AsText(ST_PointOnSurface('POLYGON((0 0, 0 5, 5 5, 5 0, 0 0))'::geometry));
      st_astext
-----
POINT(2.5 2.5)
(1 row)

SELECT ST_AsEWKT(ST_PointOnSurface(ST_GeomFromEWKT('LINESTRING(0 5 1, 0 0 1, 0 10 2)')));
      st_asewkt
-----
POINT(0 0 1)
(1 row)

```

Siehe auch

[ST_Centroid](#), [ST_PointInsideCircle](#)

8.9.50 ST_Project

ST_Project — Returns a `POINT` projected from a start point using a distance in meters and bearing (azimuth) in radians.

Synopsis

geography **ST_Project**(geography g1, float distance, float azimuth);

Beschreibung

Returns a `POINT` projected along a geodesic from a start point using an azimuth (bearing) measured in radians and distance measured in meters. This is also called a direct geodesic problem.

The azimuth is sometimes called the heading or the bearing in navigation. It is measured relative to true north (azimuth zero). East is azimuth 90 ($\pi/2$), south is azimuth 180 (π), west is azimuth 270 ($3\pi/2$).

Die Entfernung wird in Meter angegeben.

Verfügbarkeit: 2.0.0

Enhanced: 2.4.0 Allow negative distance and non-normalized azimuth.

Example: Using degrees - projected point 100,000 meters and bearing 45 degrees

```

SELECT ST_AsText(ST_Project('POINT(0 0)'::geography, 100000, radians(45.0)));
      st_astext
-----
POINT(0.635231029125537 0.639472334729198)
(1 row)

```

Siehe auch

[ST_Azimuth](#), [ST_Distance](#), [PostgreSQL Math Functions](#)

8.9.51 ST_Relate

ST_Relate — Returns true if this Geometry is spatially related to another Geometry, by testing for intersections between the Interior, Boundary and Exterior of the two geometries as specified by the values in the intersectionMatrixPattern. If no intersectionMatrixPattern is passed in, then returns the maximum intersectionMatrixPattern that relates the 2 geometries.

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

Version 1: Takes geomA, geomB, intersectionMatrix and Returns 1 (TRUE) if this Geometry is spatially related to another Geometry, by testing for intersections between the Interior, Boundary and Exterior of the two geometries as specified by the values in the [DE-9IM matrix pattern](#).

This is especially useful for testing compound checks of intersection, crosses, etc in one step.

Bitte nicht mit dem Argument "GeometryCollection" aufrufen



Note

This is the "allowable" version that returns a boolean, not an integer. This is defined in OGC spec



Note

This DOES NOT automagically include an index call. The reason for that is some relationships are anti e.g. Disjoint. If you are using a relationship pattern that requires intersection, then include the && index call.

Version 2: Takes geomA and geomB and returns the [Section 4.3.6](#)

Version 3: same as version 2, but allows to specify a boundary node rule (1:OGC/MOD2, 2:Endpoint, 3:MultivalentEndpoint, 4:MonovalentEndpoint)



Note

Bitte nicht mit dem Argument "GeometryCollection" aufrufen

not in OGC spec, but implied. see s2.1.13.2

Wird durch das GEOS Modul ausgeführt



This method implements the [OpenGIS Simple Features Implementation Specification for SQL 1.1](#). s2.1.1.2 // s2.1.13.3



This method implements the SQL/MM specification. SQL-MM 3: 5.1.25

Enhanced: 2.0.0 - added support for specifying boundary node rule (requires GEOS >= 3.0).

Beispiele

```
--Find all compounds that intersect and not touch a poly (interior intersects)
SELECT l.* , b.name As poly_name
      FROM polys As b
 INNER JOIN compounds As l
    ON (p.the_geom && b.the_geom
 AND ST_Relate(l.the_geom, b.the_geom, 'T*****'));
```

```

SELECT ST_Relate(ST_GeometryFromText('POINT(1 2)'), ST_Buffer(ST_GeometryFromText('POINT(1 2)'),2));
st_relate
-----
0FFFFFF212

SELECT ST_Relate(ST_GeometryFromText('LINESTRING(1 2, 3 4)'), ST_GeometryFromText('LINESTRING(5 6, 7 8)'));
st_relate
-----
FF1FF0102

SELECT ST_Relate(ST_GeometryFromText('POINT(1 2)'), ST_Buffer(ST_GeometryFromText('POINT(1 2)'),2), '0FFFFFF212');
st_relate
-----
t

SELECT ST_Relate(ST_GeometryFromText('POINT(1 2)'), ST_Buffer(ST_GeometryFromText('POINT(1 2)'),2), '*FF*FF212');
st_relate
-----
t

```

Siehe auch

[ST_Crosses](#), [Section 4.3.6](#), [ST_Disjoint](#), [ST_Intersects](#), [ST_Touches](#)

8.9.52 ST_RelateMatch

ST_RelateMatch — Returns true if intersectionMatrixPattern1 implies intersectionMatrixPattern2

Synopsis

boolean **ST_RelateMatch**(text intersectionMatrix, text intersectionMatrixPattern);

Beschreibung

Takes intersectionMatrix and intersectionMatrixPattern and Returns true if the intersectionMatrix satisfies the intersectionMatrixPattern. For more information refer to [Section 4.3.6](#).

Verfügbarkeit: 2.0.0 - benötigt GEOS >= 3.3.0.

Beispiele

```

SELECT ST_RelateMatch('101202FFF', 'TTTTTFFF') ;
-- result --
t
--example of common intersection matrix patterns and example matrices
-- comparing relationships of involving one invalid geometry and ( a line and polygon that intersect at interior and boundary)
SELECT mat.name, pat.name, ST_RelateMatch(mat.val, pat.val) As satisfied
FROM
  ( VALUES ('Equality', 'T1FF1FFF1'),
    ('Overlaps', 'T*T***T**'),

```



```

        ('Within', 'T*F**F***'),
        ('Disjoint', 'FF*FF****') As pat(name,val)
CROSS JOIN
    (
        VALUES ('Self intersections (invalid)', '111111111'),
        ('IE2_BI1_BB0_BE1_EI1_EE2', 'FF2101102'),
        ('IB1_IE1_BB0_BE0_EI2_EI1_EE2', 'F11F00212')
    ) As mat(name,val);

```

Siehe auch

Section [4.3.6](#), [ST_Relate](#)

8.9.53 ST_ShortestLine

ST_ShortestLine — Gibt die 2-dimensionale kürzeste Strecke zwischen zwei Geometrien als Linie zurück

Synopsis

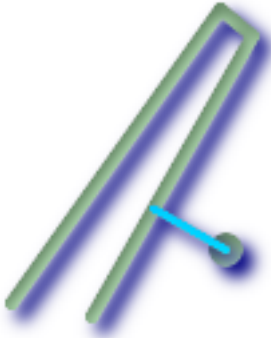
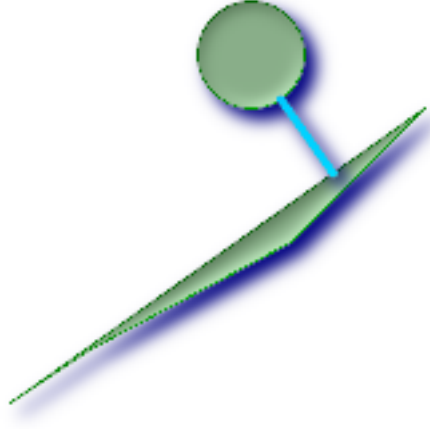
geometry **ST_ShortestLine**(geometry g1, geometry g2);

Beschreibung

Returns the 2-dimensional shortest line between two geometries. The function will only return the first shortest line if more than one, that the function finds. If g1 and g2 intersects in just one point the function will return a line with both start and end in that intersection-point. If g1 and g2 are intersecting with more than one point the function will return a line with start and end in the same point but it can be any of the intersecting points. The line returned will always start in g1 and end in g2. The length of the line this function returns will always be the same as ST_Distance returns for g1 and g2.

Verfügbarkeit: 1.5.0

Beispiele

 <i>Kürzeste Strecke zwischen Punkt und Linienzug</i> <pre> SELECT ST_AsText (ST_ShortestLine('POINT(100 100) ← '::geometry, 'LINESTRING (20 80, 98 ← 190, 110 180, 50 75)'::geometry)) As sline; sline ----- LINESTRING(100 100,73.0769230769231 ← 115.384615384615) </pre>	 <i>kürzeste Strecke zwischen Polygon und Polygon</i> <pre> SELECT ST_AsText (ST_ShortestLine(ST_GeomFromText(' ← POLYGON((175 150, 20 40, 50 60, 125 ST_Buffer(← ST_GeomFromText('POINT(110 170)'), 2)) As slinewkt; LINESTRING(140.752120669087 ← 125.695053378061,121.111404660392 15 </pre>

Siehe auch

[ST_ClosestPoint](#), [ST_Distance](#), [ST_LongestLine](#), [ST_MaxDistance](#)

8.9.54 ST_Touches

ST_Touches — Returns TRUE if the geometries have at least one point in common, but their interiors do not intersect.

Synopsis

boolean **ST_Touches**(geometry g1, geometry g2);

Beschreibung

Returns TRUE if the only points in common between *g1* and *g2* lie in the union of the boundaries of *g1* and *g2*. The **ST_Touches** relation applies to all Area/Area, Line/Line, Line/Area, Point/Area and Point/Line pairs of relationships, but *not* to the Point/Point pair.

In mathematical terms, this predicate is expressed as:

$$a.Touches(b) \Leftrightarrow (I(a) \cap I(b) = \emptyset) \wedge (a \cap b) \neq \emptyset$$

The allowable DE-9IM Intersection Matrices for the two geometries are:

- FT*****
- F**T*****
- F***T*****



Important

Do not call with a `GEOMETRYCOLLECTION` as an argument



Note

This function call will automatically include a bounding box comparison that will make use of any indexes that are available on the geometries. To avoid using an index, use `_ST_Touches` instead.



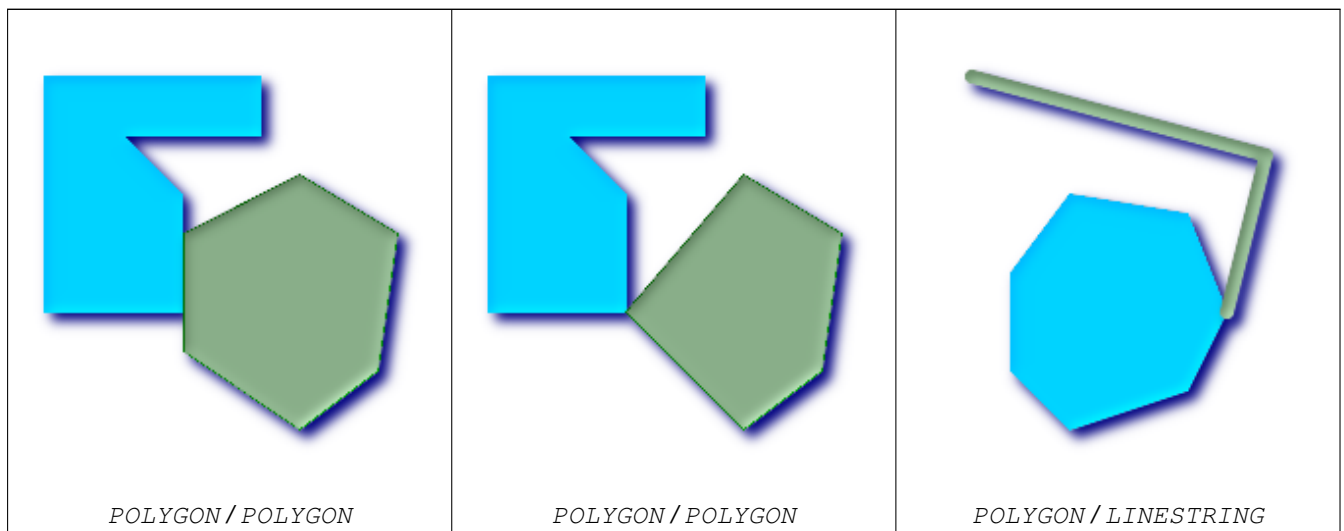
This method implements the [OpenGIS Simple Features Implementation Specification for SQL 1.1](#). s2.1.1.2 // s2.1.13.3

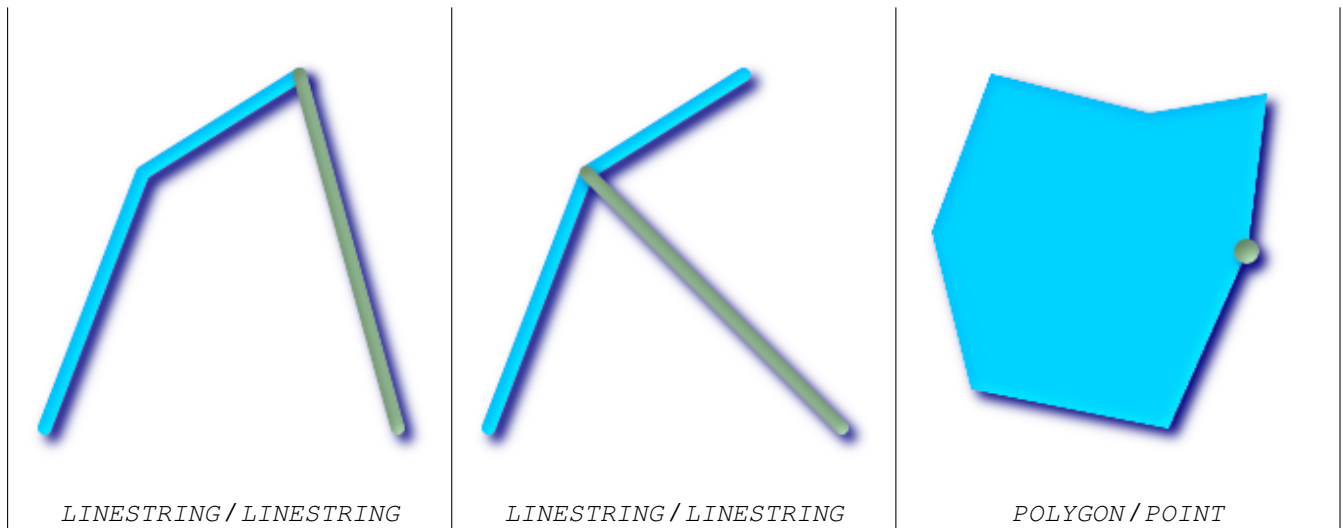


This method implements the SQL/MM specification. SQL-MM 3: 5.1.28

Beispiele

The `ST_Touches` predicate returns `TRUE` in all the following illustrations.





```
SELECT ST_Touches('LINESTRING(0 0, 1 1, 0 2)::geometry, 'POINT(1 1)::geometry');
st_touches
-----
f
(1 row)

SELECT ST_Touches('LINESTRING(0 0, 1 1, 0 2)::geometry, 'POINT(0 2)::geometry');
st_touches
-----
t
(1 row)
```

8.9.55 ST_Within

ST_Within — Gibt TRUE zurück, wenn Geometrie A zur Gänze innerhalb von Geometrie B liegt

Synopsis

boolean **ST_Within**(geometry A, geometry B);

Beschreibung

Returns TRUE if geometry A is completely inside geometry B. For this function to make sense, the source geometries must both be of the same coordinate projection, having the same SRID. It is a given that if **ST_Within**(A,B) is true and **ST_Within**(B,A) is true, then the two geometries are considered spatially equal.

Wird durch das GEOS Modul ausgeführt

Enhanced: 2.3.0 Enhancement to PIP short-circuit for geometry extended to support MultiPoints with few points. Prior versions only supported point in polygon.



Important

Do not call with a `GEOMETRYCOLLECTION` as an argument



Important

Do not use this function with invalid geometries. You will get unexpected results.

This function call will automatically include a bounding box comparison that will make use of any indexes that are available on the geometries. To avoid index use, use the function `_ST_Within`.

NOTE: this is the "allowable" version that returns a boolean, not an integer.



This method implements the [OpenGIS Simple Features Implementation Specification for SQL 1.1](#). s2.1.1.2 // s2.1.13.3 - a.Relate(b, 'T**F**F***')



This method implements the SQL/MM specification. SQL-MM 3: 5.1.30

Beispiele

```
--Ein Kreis innerhalb eines Kreises
SELECT ST_Within(smallc,smallc) As smallinsmall,
       ST_Within(smallc, bigc) As smallinbig,
       ST_Within(bigc,smallc) As biginsmall,
       ST_Within(ST_Union(smallc, bigc), bigc) as unioninbig,
       ST_Within(bigc, ST_Union(smallc, bigc)) as beginunion,
       ST_Equals(bigc, ST_Union(smallc, bigc)) as bigisunion
FROM
(
SELECT ST_Buffer(ST_GeomFromText('POINT(50 50)'), 20) As smallc,
       ST_Buffer(ST_GeomFromText('POINT(50 50)'), 40) As bigc) As foo;
--Result
smallinsmall | smallinbig | biginsmall | unioninbig | beginunion | bigisunion
-----+-----+-----+-----+-----+-----
t            | t          | f          | t          | t          | t
(1 row)
```



Siehe auch

[ST_Contains](#), [ST_Equals](#), [ST_IsValid](#)

8.10 SFCGAL Funktionen

8.10.1 postgis_sfcgal_version

postgis_sfcgal_version — Gibt die verwendete Version von SFCGAL aus

Synopsis

text **postgis_sfcgal_version**(void);

Beschreibung

Verfügbarkeit: 2.1.0



This method needs SFCGAL backend.



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



This function supports Polyhedral surfaces.



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

8.10.2 ST_Extrude

ST_Extrude — Weitet eine Oberfläche auf ein entsprechendes Volumen aus

Synopsis

geometry **ST_Extrude**(geometry geom, float x, float y, float z);

Beschreibung

Verfügbarkeit: 2.1.0



This method needs SFCGAL backend.



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



This function supports Polyhedral surfaces.

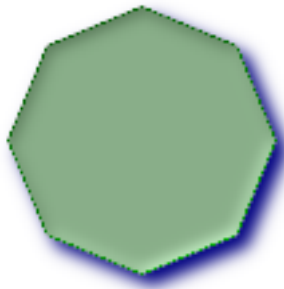


This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).

Beispiele

Die 3D Bilder sind mit PostGIS **ST_AsX3D** erzeugt und das Rendern in HTML mit **X3Dom HTML Javascript rendering library**.

```
SELECT ST_Buffer(ST_GeomFromText('POINT ↵
(100 90)'),
50, 'quad_segs=2'),0,0,30);
```



Ursprüngliches Achteck aus einem gepufferten Punkt gebildet

```
ST_Extrude(ST_Buffer(ST_GeomFromText(' ↵
POINT(100 90)'),
50, 'quad_segs=2'),0,0,30);
```



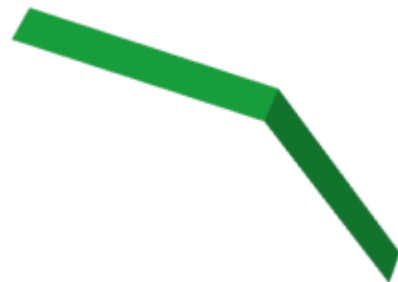
Ein Achteck, um 30 Einheiten entlang der Z Achse ausgeweitet, ergibt ein PolyhedralSurfaceZ

```
SELECT ST_GeomFromText('LINESTRING(50 50, ↵
100 90, 95 150)')
```



Ursprünglicher Linienzug

```
SELECT ST_Extrude(
ST_GeomFromText('LINESTRING(50 50, 100 ↵
90, 95 150)'),0,0,10));
```



Ein Linienzug, entlang der Z Achse ausgeweitet, ergibt ein PolyhedralSurfaceZ

Siehe auch[ST_AsX3D](#)**8.10.3 ST_StraightSkeleton**

ST_StraightSkeleton — Berechnet aus einer Geometrie ein "Gerippe" aus Geraden.

Synopsis

geometry **ST_StraightSkeleton**(geometry geom);

Beschreibung

Verfügbarkeit: 2.1.0



This method needs SFCGAL backend.



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



This function supports Polyhedral surfaces.



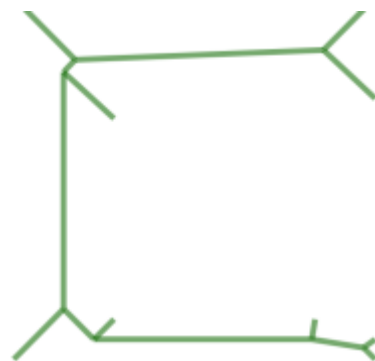
This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).

Beispiele

```
SELECT ST_StraightSkeleton(ST_GeomFromText('POLYGON (( 190 190, 10 190, 10 10, 190 10, 190 ←  
20, 160 30, 60 30, 60 130, 190 140, 190 190 ))'));
```



Ursprüngliches Polygon



Das "Gerippe" des Polygons

8.10.4 ST_ApproximateMedialAxis

ST_ApproximateMedialAxis — Errechnet die genäherte Mediale Achse einer Flächengeometrie.

Synopsis

geometry **ST_ApproximateMedialAxis**(geometry geom);

Beschreibung

Gibt die genäherte mediale Achse einer Flächeneingabe als eine Art Gerippe an Geraden zurück. Wenn mit einer geeigneten Version (1.2.0+) kompiliert wurde, wird die SFCGAL-eigene API ausgeführt. Sonst verhält sich die Funktion nur wie ein Adapter für ST_StraightSkeleton (langsamerer Fall).

Verfügbarkeit: 2.2.0



This method needs SFCGAL backend.



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



This function supports Polyhedral surfaces.



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

Beispiele

```
SELECT ST_ApproximateMedialAxis(ST_GeomFromText('POLYGON (( 190 190, 10 190, 10 10, 190 10, ↵
190 20, 160 30, 60 30, 60 130, 190 140, 190 190 ))'));
```



Ein Polygon und dessen angenäherte mediale Achse

Siehe auch

[ST_StraightSkeleton](#)

8.10.5 ST_IsPlanar

ST_IsPlanar — Überprüft ob es sich um eine ebene Oberfläche handelt oder nicht

Synopsis

boolean **ST_IsPlanar**(geometry geom);

Beschreibung

Verfügbarkeit: 2.2.0: Wurde zwar für 2.1.0 dokumentiert, aber unabsichtlich in der Version 2.1 weggelassen.



This method needs SFCGAL backend.



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



This function supports Polyhedral surfaces.



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

8.10.6 ST_Orientation

ST_Orientation — Bestimmt die Ausrichtung der Fläche

Synopsis

integer **ST_Orientation**(geometry geom);

Beschreibung

Die Funktion ist nur auf Polygone anwendbar. Sie gibt -1 zurück, wenn das Polygon gegen den Uhrzeigersinn ausgerichtet ist und 1, wenn das Polygon im Uhrzeigersinn ausgerichtet ist.

Verfügbarkeit: 2.1.0



This method needs SFCGAL backend.



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

8.10.7 ST_ForceLHR

ST_ForceLHR — Erzwingt LHR Orientierung

Synopsis

geometry **ST_ForceLHR**(geometry geom);

Beschreibung

Verfügbarkeit: 2.1.0



This method needs SFCGAL backend.



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



This function supports Polyhedral surfaces.



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

8.10.8 ST_MinkowskiSum

ST_MinkowskiSum — Berechnet die Minkowski-Summe

Synopsis

```
geometry ST_MinkowskiSum(geometry geom1, geometry geom2);
```

Beschreibung

Diese Funktion errechnet die Minkowski-Summe eines Punktes, einer Linie oder eines Polygons mit einem Polygon in 2D.

Die Minkowski-Summe zweier Geometrien A und B ist die Menge aller Punkte, die die Summe aller Punkte von A und B sind. Minkowski-Summen werden häufig zur Planung von Bewegungsabläufen und im CAD-Bereich eingesetzt. Weitere Einzelheiten finden Sie unter [Wikipedia Minkowski addition](#).

Der erste Parameter kann irgendeine 2D-Geometrie (Punkt, Linienzug, Polygon) sein. Wenn eine 3D-Geometrie eingegeben wird, so wird diese in 2D umgewandelt, indem Z auf 0 gesetzt wird. Dies kann zu ungünstigen Spezialfällen führen. Der zweite Parameter muss ein 2D-Polygon sein.

Die Umsetzung nützt [CGAL 2D Minkowskisum](#).

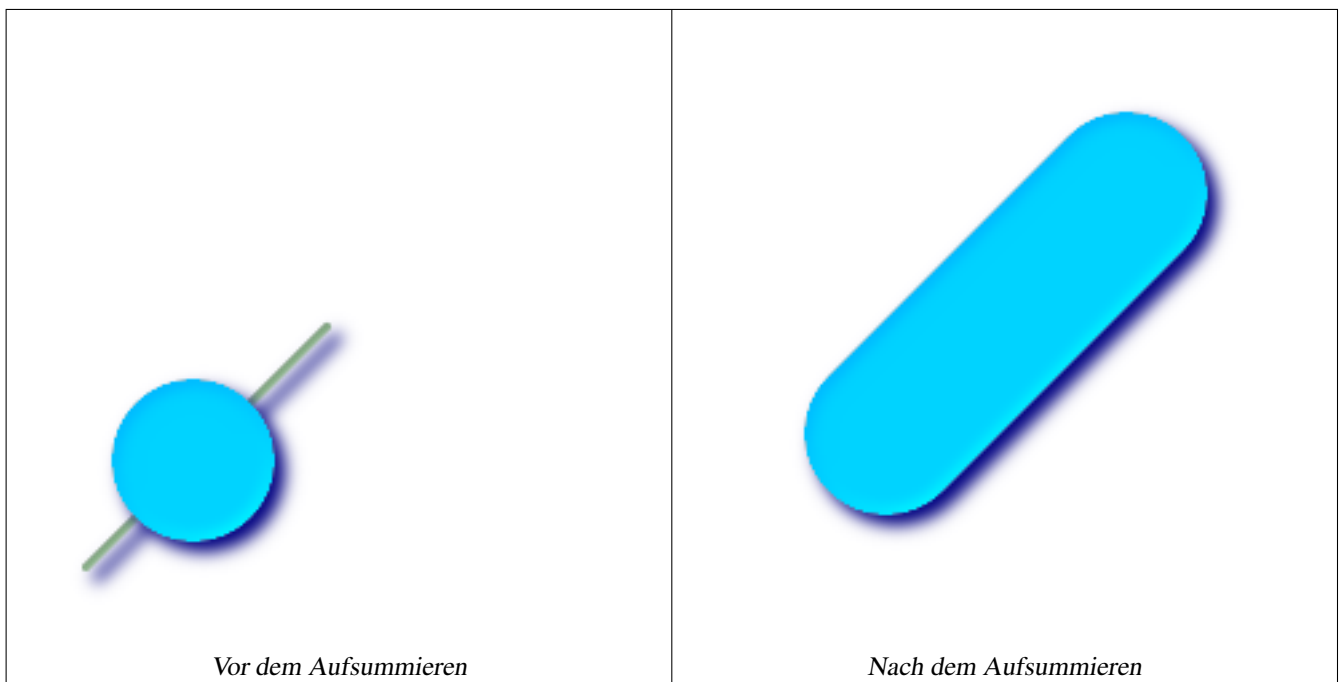
Verfügbarkeit: 2.1.0



This method needs SFCGAL backend.

Beispiele

Die Minkowski-Summe eines LineString's, der ein Kreispolygon schneidet



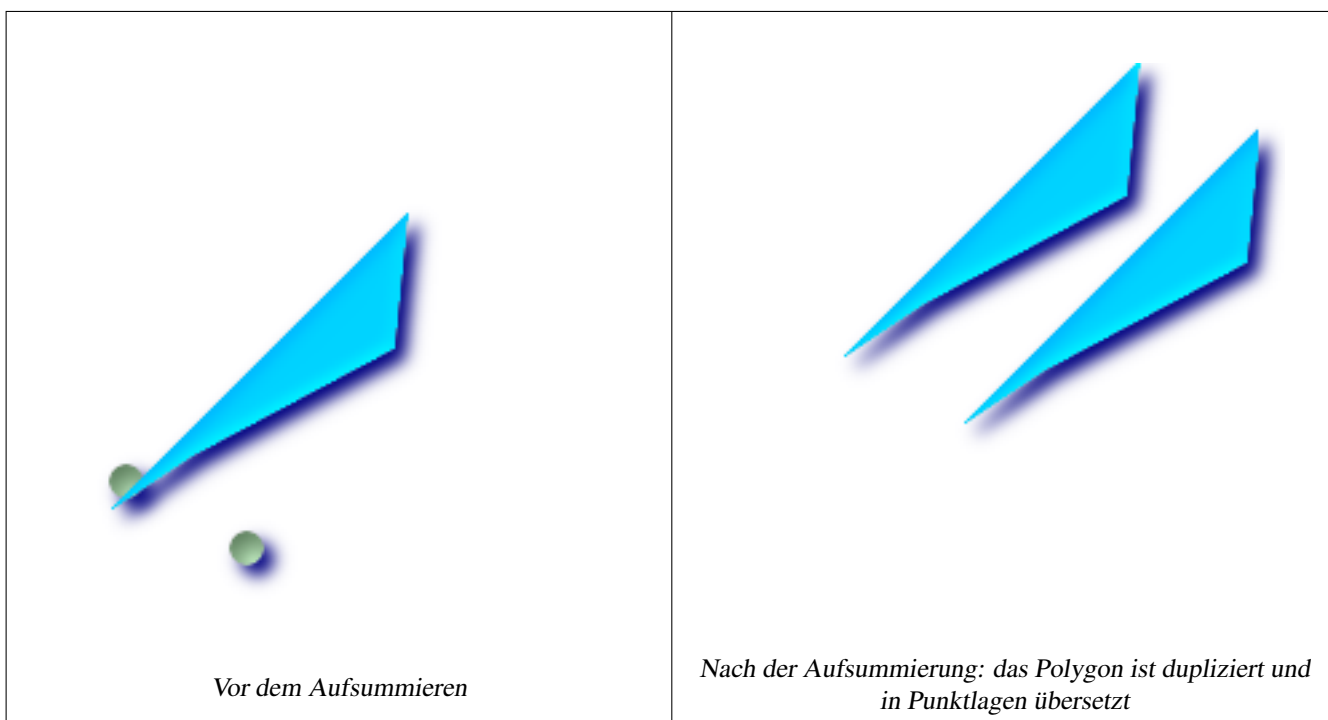
```

SELECT ST_MinkowskiSum(line, circle))
FROM (SELECT
  ST_MakeLine(ST_MakePoint(10, 10),ST_MakePoint(100, 100)) As line,
  ST_Buffer(ST_GeomFromText('POINT(50 50)'), 30) As circle) As foo;

-- wkt --
MULTIPOLYGON(((30 59.9999999999999,30.5764415879031 54.1472903395161,32.2836140246614 ↵
  48.5194970290472,35.0559116309237 43.3328930094119,38.7867965644036 ↵
  38.7867965644035,43.332893009412 35.0559116309236,48.5194970290474 ↵
  32.2836140246614,54.1472903395162 30.5764415879031,60.0000000000001 30,65.8527096604839 ↵
  30.5764415879031,71.4805029709527 32.2836140246614,76.6671069905881 ↵
  35.0559116309237,81.2132034355964 38.7867965644036,171.213203435596 ↵
  128.786796564404,174.944088369076 133.332893009412,177.716385975339 ↵
  138.519497029047,179.423558412097 144.147290339516,180 150,179.423558412097 ↵
  155.852709660484,177.716385975339 161.480502970953,174.944088369076 ↵
  166.667106990588,171.213203435596 171.213203435596,166.667106990588 174.944088369076,
  161.480502970953 177.716385975339,155.852709660484 179.423558412097,150 ↵
  180,144.147290339516 179.423558412097,138.519497029047 177.716385975339,133.332893009412 ↵
  174.944088369076,128.786796564403 171.213203435596,38.7867965644035 ↵
  81.2132034355963,35.0559116309236 76.667106990588,32.2836140246614 ↵
  71.4805029709526,30.5764415879031 65.8527096604838,30 59.9999999999999)))

```

Minkowski Summe von einem Polygon mit einem MultiPoint



```
SELECT ST_MinkowskiSum(mp, poly)
FROM (SELECT 'MULTIPOINT(25 50,70 25)::geometry As mp,
      'POLYGON((130 150, 20 40, 50 60, 125 100, 130 150))::geometry As poly
      ) As foo

-- wkt --
MULTIPOLYGON(
  ((70 115,100 135,175 175,225 225,70 115)),
  ((120 65,150 85,225 125,275 175,120 65))
)
```

8.10.9 ST_3DIntersection

ST_3DIntersection — Führt eine Verschneidung in 3D aus

Synopsis

geometry **ST_3DIntersection**(geometry geom1, geometry geom2);

Beschreibung

Gibt die Schnittmenge von geom1 und geom2 als Geometrie zurück.

Verfügbarkeit: 2.1.0



This method needs SFCGAL backend.



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



This function supports Polyhedral surfaces.

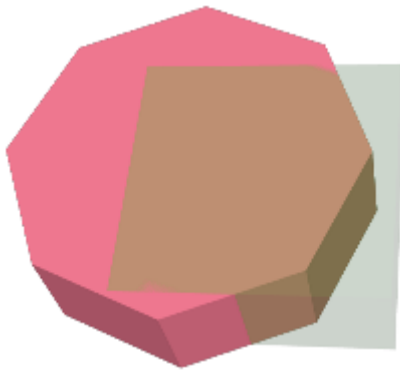


This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).

Beispiele

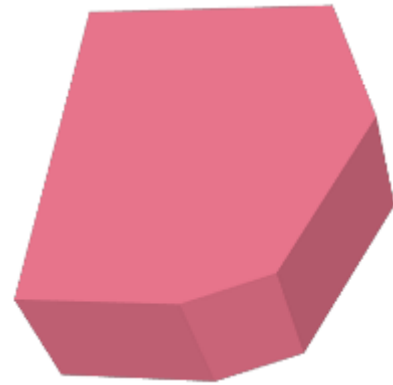
Die 3D Bilder sind mit PostGIS **ST_AsX3D** erzeugt und das Rendern in HTML mit **X3Dom HTML Javascript rendering library**.

```
SELECT ST_Extrude(ST_Buffer( ←
    ST_GeomFromText('POINT(100 90)'),
    50, 'quad_segs=2'),0,0,30) AS geom1,
    ST_Extrude(ST_Buffer( ←
    ST_GeomFromText('POINT(80 80)'),
    50, 'quad_segs=1'),0,0,30) AS geom2;
```



Ursprüngliche 3D-Geometrien überlagert. geom2 wird halbtransparent angezeigt

```
SELECT ST_3DIntersection(geom1,geom2)
FROM ( SELECT ST_Extrude(ST_Buffer( ←
    ST_GeomFromText('POINT(100 90)'),
    50, 'quad_segs=2'),0,0,30) AS geom1,
    ST_Extrude(ST_Buffer( ←
    ST_GeomFromText('POINT(80 80)'),
    50, 'quad_segs=1'),0,0,30) AS geom2 ) As ←
    t;
```



Schnittmenge von geom1 und geom2

Linienzüge und Polygone in 3D

```
SELECT ST_AsText(ST_3DIntersection(linestring, polygon)) As wkt
FROM ST_GeomFromText('LINESTRING Z (2 2 6,1.5 1.5 7,1 1 8,0.5 0.5 8,0 0 10)') AS ←
    linestring
CROSS JOIN ST_GeomFromText('POLYGON((0 0 8, 0 1 8, 1 1 8, 1 0 8, 0 0 8))') AS polygon;

wkt
-----
LINESTRING Z (1 1 8,0.5 0.5 8)
```

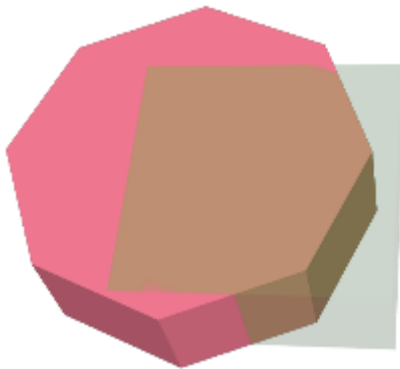
Würfel (geschlossene polyedrische Oberfläche) und Polygon Z

```
SELECT ST_AsText(ST_3DIntersection(
    ST_GeomFromText('POLYHEDRALSURFACE Z( ((0 0 0, 0 0 1, 0 1 1, 0 1 0, 0 0 0)) ←
    ,
    ((0 0 0, 0 1 0, 1 1 0, 1 0 0, 0 0 0)), ((0 0 0, 1 0 0, 1 0 1, 0 0 1, 0 0 0)),
    ((1 1 0, 1 1 1, 1 0 1, 1 0 0, 1 1 0)),
    ((0 1 0, 0 1 1, 1 1 1, 1 1 0, 0 1 0)), ((0 0 1, 1 0 1, 1 1 1, 0 1 1, 0 0 1)) )'),
    'POLYGON Z ((0 0 0, 0 0 0.5, 0 0.5 0.5, 0 0.5 0, 0 0 0))'::geometry))
```

```
TIN Z (((0 0 0,0 0 0.5,0 0.5 0.5,0 0 0)),((0 0.5 0,0 0 0,0 0.5 0.5,0 0.5 0)))
```

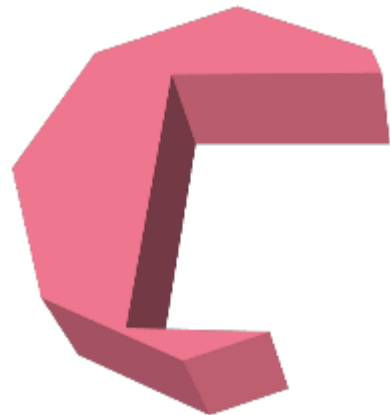


```
SELECT ST_Extrude(ST_Buffer(↵
    ST_GeomFromText('POINT(100 90)'),
    50, 'quad_segs=2'),0,0,30) AS geom1,
    ST_Extrude(ST_Buffer(↵
    ST_GeomFromText('POINT(80 80)'),
    50, 'quad_segs=1'),0,0,30) AS geom2;
```



Ursprüngliche 3D-Geometrien überlagert. geom2 ist jener Teil der nicht entfernt wird.

```
SELECT ST_3DDifference(geom1,geom2)
FROM ( SELECT ST_Extrude(ST_Buffer(↵
    ST_GeomFromText('POINT(100 90)'),
    50, 'quad_segs=2'),0,0,30) AS geom1,
    ST_Extrude(ST_Buffer(↵
    ST_GeomFromText('POINT(80 80)'),
    50, 'quad_segs=1'),0,0,30) AS geom2 ) As ↵
t;
```



Was bleibt zurück nachdem geom2 entfernt wurde

Siehe auch

[ST_Extrude](#), [ST_AsX3D](#), [ST_3DIntersection](#) [ST_3DUnion](#)

8.10.11 ST_3DUnion

ST_3DUnion — Führt eine Vereinigung/Union in 3D aus

Synopsis

geometry **ST_3DUnion**(geometry geom1, geometry geom2);

Beschreibung

Verfügbarkeit: 2.2.0



This method needs SFCGAL backend.



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



This function supports Polyhedral surfaces.

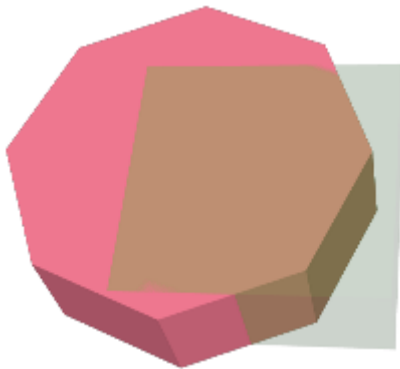


This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).

Beispiele

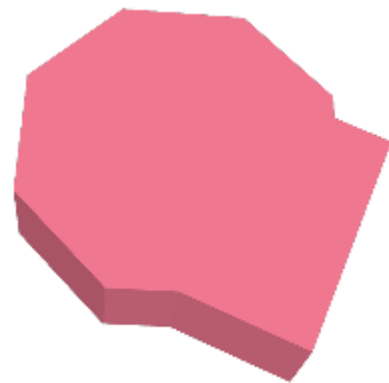
Die 3D Bilder sind mit PostGIS **ST_AsX3D** erzeugt und das Rendern in HTML mit **X3Dom HTML Javascript rendering library**.

```
SELECT ST_Extrude(ST_Buffer( ←
    ST_GeomFromText('POINT(100 90)'),
    50, 'quad_segs=2'),0,0,30) AS geom1,
    ST_Extrude(ST_Buffer( ←
    ST_GeomFromText('POINT(80 80)'),
    50, 'quad_segs=1'),0,0,30) AS geom2;
```



*Überlagerung der ursprünglichen Geometrien in 3D.
geom2 ist transparent dargestellt.*

```
SELECT ST_3DUnion(geom1,geom2)
FROM ( SELECT ST_Extrude(ST_Buffer( ←
    ST_GeomFromText('POINT(100 90)'),
    50, 'quad_segs=2'),0,0,30) AS geom1,
    ST_Extrude(ST_Buffer( ←
    ST_GeomFromText('POINT(80 80)'),
    50, 'quad_segs=1'),0,0,30) AS geom2 ) As ←
t;
```



Vereinigung von geom1 und geom2

Siehe auch

[ST_Extrude](#), [ST_AsX3D](#), [ST_3DIntersection](#) [ST_3DDifference](#)

8.10.12 ST_3DArea

ST_3DArea — Berechnet die Fläche von 3D-Oberflächengeometrien. Gibt 0 für Solids zurück.

Synopsis

```
floatST_3DArea(geometry geom1);
```

Beschreibung

Verfügbarkeit: 2.1.0



This method needs SFCGAL backend.



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



This function supports Polyhedral surfaces.



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

Beispiele

Anmerkung: Standardmäßig ist ein aus WKT erzeugtes PolyhedralSurface eine Oberflächengeometrie und kein Solid. Es hat daher eine Flächenausdehnung. In ein Solid umgewandelt, keine Fläche.

```
SELECT ST_3DArea(geom) As cube_surface_area,
       ST_3DArea(ST_MakeSolid(geom)) As solid_surface_area
FROM (SELECT 'POLYHEDRALSURFACE( ((0 0 0, 0 0 1, 0 1 1, 0 1 0, 0 0 0)),
  ((0 0 0, 0 1 0, 1 1 0, 1 0 0, 0 0 0)),
  ((0 0 0, 1 0 0, 1 0 1, 0 0 1, 0 0 0)),
  ((1 1 0, 1 1 1, 1 0 1, 1 0 0, 1 1 0)),
  ((0 1 0, 0 1 1, 1 1 1, 1 1 0, 0 1 0)),
  ((0 0 1, 1 0 1, 1 1 1, 0 1 1, 0 0 1)) )'::geometry) As f(geom);
```

cube_surface_area	solid_surface_area
6	0

Siehe auch

[ST_Area](#), [ST_MakeSolid](#), [ST_IsSolid](#), [ST_Area](#)

8.10.13 ST_Tessellate

ST_Tessellate — Erzeugt ein Oberflächen-Mosaik aus einem Polygon oder einer polyedrischen Oberfläche und gibt dieses als TIN oder als TIN-Kollektion zurück

Synopsis

geometry **ST_Tessellate**(geometry geom);

Beschreibung

Nimmt als Eingabe eine Fläche, wie ein Multi(Polygon) oder eine polyedrische Oberfläche, und gibt, mittels Mosaikierung in Dreiecke, eine TIN-Darstellung der Geometrie zurück.

Verfügbarkeit: 2.1.0



This method needs SFCGAL backend.



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

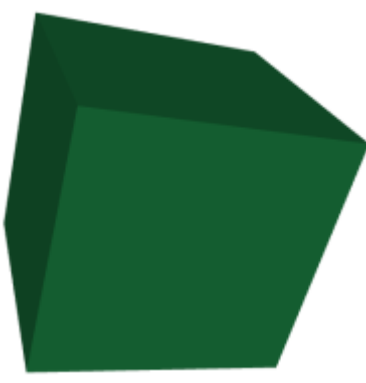


This function supports Polyhedral surfaces.



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

Beispiele

<pre>SELECT ST_GeomFromText('POLYHEDRALSURFACE Z(((0 0 0, 0 0 1, 0 1 1, 0 1 0, 0 0 0), ((0 0 0, 0 1 0, 1 1 0, 1 1 0, 0 0 0), ((1 1 0, 1 1 1, 1 0 1, 1 0 0, 1 1 0), ((0 1 0, 0 1 1, 1 1 1, 1 1 0, 0 1 1, 0 1 0, 0 1 0)), ((0 0 1, 1 0 1, 1 1 1, 0 1 0, 0 0 0)))', 0)</pre>  <i>Ursprünglicher Würfel</i>	<pre>SELECT ST_Tessellate(ST_GeomFromText('POLYHEDRALSURFACE Z(((0 0 0, 0 0 1, 0 1 1, 0 1 0, 0 0 0), ((0 0 0, 0 1 0, 1 1 0, 1 1 0, 0 0 0), ((1 1 0, 1 1 1, 1 0 1, 1 0 0, 1 1 0), ((0 1 0, 0 1 1, 1 1 1, 1 1 0, 0 1 1, 0 1 0, 0 1 0)), ((0 0 1, 1 0 1, 1 1 1, 0 1 0, 0 0 0)))', 0), 0.001)</pre> ST_AsText Ausgabe: <pre>TEXT '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 1 0, 0 0 0), ((1 1 0, 1 1 1, 1 0 1, 1 0 0, 1 1 0), ((0 1 0, 0 1 1, 1 1 1, 1 1 0, 0 1 1, 0 1 0, 0 1 0), ((0 0 1, 1 0 1, 1 1 1, 0 1 0, 0 0 0)))', 0)</pre>  <i>Mosaikierter Würfel mit eingefärbten Dreiecken</i>

```
SELECT 'POLYGON (( 10 190, 10 70, 80 70, ↵
                        80 130, 50 160, 120 160,
```



Ursprüngliches Polygon

```
SELECT
    ST_Tessellate('POLYGON (( 10 190, ↵
                        10 70, 80 70, 80 130, 50 160, 120 190, 10 190 ))',:geometry);
ST_AsText Ausgabe
TEXT('POLYGON ((80 130,50 160,80 70,80 130)),((50 ↵
                        160,10 190,10 70,50 160)),
                        ((80 70,50 160,10 70,80 70)) ↵
                        ,((120 160,120 190,50 160,120 1
                        ((120 190,10 190,50 160,120 190)))
```



Mosaikiertes Polygon

8.10.14 ST_Volume

ST_Volume — Berechnet das Volumen eines 3D-Solids. Auf Oberflächengeometrien (auch auf geschlossene) angewandt wird 0 zurückgegeben.

Synopsis

float **ST_Volume**(geometry geom1);

Beschreibung

Verfügbarkeit: 2.2.0

- ✓ This method needs SFCGAL backend.
- ✓ This function supports 3d and will not drop the z-index.
- ✓ This function supports Polyhedral surfaces.
- ✓ This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).

Beispiel

Wenn geschlossene Oberflächen über WKT erzeugt werden, so werden diese wie eine Flächen und nicht wie ein Solid behandelt. Um sie in ein Solid umzuwandeln, müssen Sie **ST_MakeSolid** verwenden. Flächeneometrien besitzen kein Volumen. Das folgende Beispiel demonstriert dies.

```
SELECT ST_Volume(geom) As cube_surface_vol,
       ST_Volume(ST_MakeSolid(geom)) As solid_surface_vol
FROM (SELECT 'POLYHEDRALSURFACE( ((0 0 0, 0 0 1, 0 1 1, 0 1 0, 0 0 0)),
  ((0 0 0, 0 1 0, 1 1 0, 1 0 0, 0 0 0)),
  ((0 0 0, 1 0 0, 1 0 1, 0 0 1, 0 0 0)),
  ((1 1 0, 1 1 1, 1 0 1, 1 0 0, 1 1 0)),
  ((0 1 0, 0 1 1, 1 1 1, 1 1 0, 0 1 0)),
  ((0 0 1, 1 0 1, 1 1 1, 0 1 1, 0 0 1)) )'::geometry) As f(geom);
```

cube_surface_vol	solid_surface_vol
0	1

Siehe auch

ST_3DArea, **ST_MakeSolid**, **ST_IsSolid**

8.10.15 ST_MakeSolid

ST_MakeSolid — Wandelt die Geometrie in ein Solid um. Es wird keine Überprüfung durchgeführt. Um ein gültiges Solid zu erhalten muss die eingegebene Geometrie entweder eine geschlossene polyedrische Oberfläche oder ein geschlossenes TIN sein.

Synopsis

```
geometry ST_MakeSolid(geometry geom1);
```

Beschreibung

Verfügbarkeit: 2.2.0



This method needs SFCGAL backend.



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



This function supports Polyhedral surfaces.



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

8.10.16 ST_IsSolid

ST_IsSolid — Überprüft ob die Geometrie ein Solid ist. Es wird keine Plausibilitätsprüfung durchgeführt.

Synopsis

```
boolean ST_IsSolid(geometry geom1);
```

Beschreibung

Verfügbarkeit: 2.2.0



This method needs SFCGAL backend.



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



This function supports Polyhedral surfaces.



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

8.11 Geometrieverarbeitung

8.11.1 ST_Buffer

ST_Buffer — (T) Gibt eine Geometrie zurück, welche alle Punkte innerhalb einer gegebenen Entfernung von der Eingabegeometrie beinhaltet.

Synopsis

```
geometry ST_Buffer(geometry g1, float radius_of_buffer);  
geometry ST_Buffer(geometry g1, float radius_of_buffer, integer num_seg_quarter_circle);  
geometry ST_Buffer(geometry g1, float radius_of_buffer, text buffer_style_parameters);  
geography ST_Buffer(geography g1, float radius_of_buffer_in_meters);  
geography ST_Buffer(geography g1, float radius_of_buffer, integer num_seg_quarter_circle);  
geography ST_Buffer(geography g1, float radius_of_buffer, text buffer_style_parameters);
```

Beschreibung

Gibt eine Geometrie/Geographie zurück, die alle Punkte repräsentiert, deren Entfernung von dieser Geometrie/Geographie kleiner oder gleich der gegebenen Entfernung ist.

Geometrie: Berechnungen werden im Koordinatenreferenzsystem der Geometrie durchgeführt. Mit 1.5 wurde die Unterstützung unterschiedlicher Abschlussstücke/end-cap und Gehrungen/mitre eingeführt, um die Gestalt zu kontrollieren.



Note

Negative Radien: Bei Polygonen kann ein negativer Radius vergewendet werden, wodurch das Polygon geschrumpft anstatt vergrößert wird.



Note

Geographie: Beim geographischen Datentyp handelt es sich lediglich um einen schlanken Adapter, der um die geometrische Implementation herumgelegt wurde. Zuerst wird die passende SRID für das Umgebungsrechteck des geographischen Objektes bestimmt (bevorzugt UTM, Lambert Azimuthal Equal Area (LAEA) Nord/Süd Pol, im schlimmsten Fall wird auf Mercator zurückgegriffen), dann im planaren Koordinatenreferenzsystem gepuffert und anschließend nach WGS84 Geographie zurück transformiert.



Beim geographischen Datentyp kann sich dies anders verhalten als erwartet, nämlich dann, wenn das Objekt entsprechend groß ist und zwischen zwei UTM-Zonen fällt oder eine Datumsgrenze überschreitet.

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 rechtwinkligen Abschlüssen anstatt mit runden Enden umzuwandeln. Ein schlanker Adapter für den geographischen Datentyp wurde hinzugefügt. - benötigt GEOS >= 3.2 um diese erweiterte geometrische Funktionalität auszunutzen.

Der optionale dritte Parameter (zurzeit nur auf den geometrischen Datentyp anwendbar) ermöglicht es die Anzahl der Segmente zur Näherung eines Viertelkreises (Ganzzahl, standardmäßig 8) oder eine Liste von leerzeichengetrennten key=value Paaren (string case) festzulegen, um die Berechnungen wie folgt zu optimieren:

- 'quad_segs=#' : Anzahl der Segmente die verwendet werden um einen Viertelkreis anzunähern (standardmäßig 8).
- 'endcap=round|flat|square' : endcap style (standardmäßig "round", benötigt GEOS-3.2 oder höher für andere Werte). 'butt' kann auch als Synonym für 'flat' verwendet werden.
- 'join=round|mitre|bevel' : join style (standardmäßig "round", benötigt GEOS-3.2 oder höher für andere Werte). '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.

Die Einheiten des Radius werden in den Einheiten des Koordinatenreferenzsystems gemessen.

Es können POINTS, MULTIPOINTS, LINESTRINGS, MULTILINESTRINGS, POLYGONS, MULTIPOLYGONS, und Sammelgeometrien/GeometryCollections eingegeben werden.



Note

Diese Funktion ignoriert die dritte Dimension (z) und gibt immer einen 2-D Buffer zurück, sogar dann wenn eine 3D-Geometrie überreicht wird.

Wird vom GEOS Modul ausgeführt



This method implements the [OpenGIS Simple Features Implementation Specification for SQL 1.1](#). s2.1.1.3



This method implements the SQL/MM specification. SQL-MM 3: 5.1.17



Note

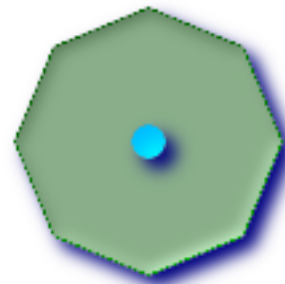
Fälschlicherweise wird diese Funktion oft zur Umkreissuche verwendet. Die Erzeugung eines Puffers zur Umkreissuche ist langsam und witzlos. Benutzen Sie bitte stattdessen [ST_DWithin](#).

Beispiele



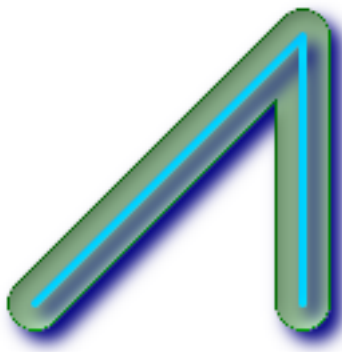
quad_segs=8 (Standardwert)

```
SELECT ST_Buffer(  
  ST_GeomFromText('POINT(100 90)'),  
  50, 'quad_segs=8');
```



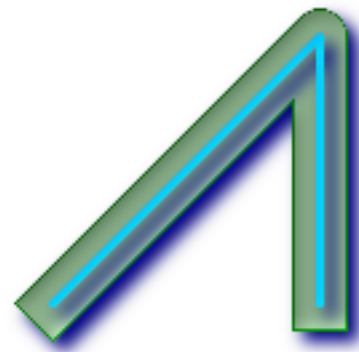
quad_segs=2 (lahme Ente)

```
SELECT ST_Buffer(  
  ST_GeomFromText('POINT(100 90)'),  
  50, 'quad_segs=2');
```



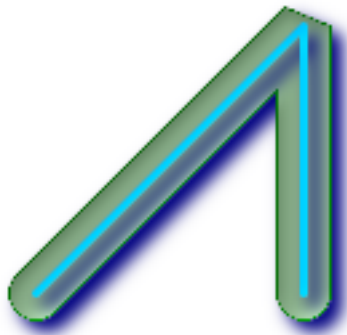
endcap=round join=round (Standardwert)

```
SELECT ST_Buffer(  
  ST_GeomFromText(  
    'LINESTRING(50 50,150 150,150 50)'  
  ), 10, 'endcap=round join=round');
```



endcap=square

```
SELECT ST_Buffer(  
  ST_GeomFromText(  
    'LINESTRING(50 50,150 150,150 50)'  
  ), 10, 'endcap=square join=round');
```


*join=bevel*

```
SELECT ST_Buffer(
  ST_GeomFromText(
    'LINESTRING(50 50,150 150,150 50)'
  ), 10, 'join=bevel');
```

*join=mitre mitre_limit=5.0 (default mitre limit)*

```
SELECT ST_Buffer(
  ST_GeomFromText(
    'LINESTRING(50 50,150 150,150 50)'
  ), 10, 'join=mitre mitre_limit=5.0');
```

```
-- Ein gepufferter Punkt nähert sich einem Kreis an (sh. Abbildung)
-- Ein gepufferter Punkt der die Annäherung mit 2 Punkten pro Viertelkreis erzwingt
-- ist ein Polygon mit 8 Seiten (sh. Abbildung)
SELECT ST_NPoints(ST_Buffer(ST_GeomFromText('POINT(100 90)'), 50)) As ←
  promisingcircle_pcount,
ST_NPoints(ST_Buffer(ST_GeomFromText('POINT(100 90)'), 50, 2)) As lamecircle_pcount;
```

```
promisingcircle_pcount | lamecircle_pcount
-----+-----
33 | 9
```

```
-- Ein leichterer aber lahmer Kreis mit 2 Punkten pro Viertelkreis ist ein Achteck
-- Unterhalb ein 100 Meter Achteck
-- Die Koordinaten sind in NAD 83 Länge/Breite und werden nach
-- Mass state plane Meter transformiert um Messungen in Meter zu bekommen
-- und anschließend gepuffert
```

```
SELECT ST_AsText(ST_Buffer(
  ST_Transform(
    ST_SetSRID(ST_MakePoint(-71.063526, 42.35785), 4269), 26986)
  ,100,2)) As octagon;
```

```
-----
POLYGON((236057.59057465 900908.759918696,236028.301252769 900838.049240578,235
957.59057465 900808.759918696,235886.879896532 900838.049240578,235857.59057465
900908.759918696,235886.879896532 900979.470596815,235957.59057465 901008.759918
696,236028.301252769 900979.470596815,236057.59057465 900908.759918696))
```

Siehe auch

[ST_Collect](#), [ST_DWithin](#), [ST_SetSRID](#), [ST_Transform](#), [ST_Union](#)

8.11.2 ST_BuildArea

ST_BuildArea — Erzeugt eine Flächengeometrie aus den einzelnen Linien einer gegebenen Geometrie

Synopsis

geometry **ST_BuildArea**(geometry A);

Beschreibung

Erzeugt eine Flächengeometrie aus den einzelnen Linien einer gegebenen Geometrie. Der zurückgegebene Datentyp ist, abhängig von der Eingabe, ein Polygon oder ein MultiPolygon. Wenn das Eingabe-Liniennetz keine Polygone bildet, wird NULL zurückgegeben. Die Eingabe können LineStrings, MultiLineStrings, Polygons, MultiPolygons und GeometryCollections sein.

Diese Funktion nimmt an, dass alle inneren Geometrien Lücken/Inseln darstellen.



Note

Damit diese Funktion korrekt arbeitet, müssen die Knoten des eingegebenen Liniennetzes richtig angeordnet sein

Verfügbarkeit: 1.1.0 - benötigt GEOS >= 2.1.0.

Beispiele



Erzeugt einen Donut

```
SELECT ST_BuildArea(ST_Collect(smallc,bigc))
FROM (SELECT
      ST_Buffer(
        ST_GeomFromText('POINT(100 90)'), 25) As smallc,
      ST_Buffer(ST_GeomFromText('POINT(100 90)'), 50) As bigc) As foo;
```



Dies erzeugt eine auseinanderklaffende Lücke innerhalb des Kreises mit herausstehenden Zacken

```
SELECT ST_BuildArea(ST_Collect(line,circle))
FROM (SELECT
  ST_Buffer(
    ST_MakeLine(ST_MakePoint(10, 10),ST_MakePoint(190, 190)),
    5) As line,
  ST_Buffer(ST_GeomFromText('POINT(100 90)'), 50) As circle) As foo;

--dies erzeugt dieselbe auseinanderklaffende Lücke
--allerdings aus Linienzügen anstelle von Polygonen
SELECT ST_BuildArea(
  ST_Collect(ST_ExteriorRing(line),ST_ExteriorRing(circle))
)
FROM (SELECT ST_Buffer(
  ST_MakeLine(ST_MakePoint(10, 10),ST_MakePoint(190, 190))
  ,5) As line,
  ST_Buffer(ST_GeomFromText('POINT(100 90)'), 50) As circle) As foo;
```

Siehe auch

[ST_Node](#), [ST_MakePolygon](#), [ST_BdPolyFromText](#), [ST_BdMPolyFromText](#) Adapter für diese Funktion mit der OGC-Standardschnittstelle

8.11.3 ST_ClipByBox2D

ST_ClipByBox2D — Gibt jenen Teil der Geometrie zurück, der innerhalb eines Rechteckes liegt.

Synopsis

geometry **ST_ClipByBox2D**(geometry geom, box2d box);

Beschreibung

Schneidet eine Geometrie mittels einer 2D-Box aus; eine schnelle aber möglicherweise unsaubere Methode. Es ist nicht garantiert, dass die Ausgabegeometrie valide ist (bei Polygonen kann es zu Selbstüberschneidungen kommen). Topologisch

invalide Eingabegeometrien führen zu keiner Fehlermeldung.

Wird vom GEOS Modul ausgeführt



Note

Benötigt GEOS 3.5.0+

Verfügbarkeit: 2.2.0 - benötigt GEOS >= 3.5.0.

Beispiele

```
-- Der zweite Eingabeparameter baut auf die implizite Typumwandlung von Geometrie nach ↔
Box2D auf
SELECT ST_ClipByBox2D(the_geom, ST_MakeEnvelope(0,0,10,10)) FROM mytab;
```

Siehe auch

[ST_Intersection](#), [ST_MakeBox2D](#), [ST_MakeEnvelope](#)

8.11.4 ST_Collect

ST_Collect — Gibt einen festgelegten ST_Geometry Wert aus einer Sammlung anderer Geometrien zurück.

Synopsis

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

Beschreibung

Der Ausgabetypp kann eine MULTI* oder eine GEOMETRYCOLLECTION sein. Hat 2 Varianten. Variante 1 sammelt 2 Geometrien ein. Variante 2 ist eine Aggregatfunktion, die einen Satz an Geometrien entgegennimmt und diese in einer einzelnen ST_Geometry versammelt.

Aggregat Version:: Diese Funktion gibt eine GEOMETRYCOLLECTION oder ein MULTI Objekt aus einem Satz an Geometrien. Die Funktion ST_Collect() ist in der Terminologie von PostgreSQL eine Aggregatfunktion. Dies bedeutet, dass sie so wie die SUM() und AVG() Funktionen mit Datenzeilen arbeitet. Zum Beispiel, "SELECT ST_Collect(GEOM) FROM GEOMTABLE GROUP BY ATTRCOLUMN" gibt eine einzelne GEOMETRYCOLLECTION für jeden eindeutigen Wert von ATTRCOLUMN zurück.

Nicht-Aggregat Version: Diese Funktion gibt eine Geometrie zurück, die aus der Ansammlung zweier Eingabegeometrien besteht. Der Ausgabetypp kann eine Mehrfach-/MULTI* oder eine Sammelgeometrie/GEOMETRYCOLLECTION sein.



Note

ST_Collect und ST_Union sind oftmals gegeneinander austauschbar, außer das ST_Collect immer eine GeometryCollection oder MULTI-Geometrie zurückgibt und ST_Union kann Einzelgeometrien zurückgeben, wenn es die Grenzlinien auflöst. ST_Union teilt Linienzüge an Überschneidungsknoten, während ST_Collect Linienzüge niemals auftrennt und nur einen MULTILINESTRING zurückgibt. Um ST_Collect von der Ausgabe einer GeometryCollection bei der Sammlung von MULTI-Geometrien abzuhalten, kann man den unteren Trick anwenden, der [ST_Dump](#) nützt, um die MULTIs in Einzelgeometrien zu zerlegen und diese anschließend neu gruppiert.

Verfügbarkeit: 1.4.0 - ST_Collect(geomarray) wurde eingeführt. ST_Collect wurde verbessert, um mehrere Geometrien schneller handhaben zu können.



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



This method supports Circular Strings and Curves Diese Methode unterstützt Kreisbögen und Kurven, gibt aber niemals eine MULTICURVE oder MULTI* zurück, so wie man es sich erwarten würde und PostGIS unterstützt diese auch nicht.

Beispiele

Aggregat Beispiel

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

Nicht-Aggregat Beispiel

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

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

--2D-Punkte sammeln
SELECT ST_AsText(ST_Collect(ST_GeomFromText('POINT(1 2)'),
                          ST_GeomFromText('POINT(1 2)')));

st_astext
-----
MULTIPOINT(1 2,1 2)

--3D-Punkte sammeln
SELECT ST_AsEWKT(ST_Collect(ST_GeomFromEWKT('POINT(1 2 3)'),
                          ST_GeomFromEWKT('POINT(1 2 4)')));

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

--Beispiel mit Kurven
SELECT ST_AsText(ST_Collect(ST_GeomFromText('CIRCULARSTRING(220268 150415,220227 150505,220227 150406)'),
                          ST_GeomFromText('CIRCULARSTRING(220227 150406,220227 150407,220227 150406)')));

-----
GEOMETRYCOLLECTION(CIRCULARSTRING(220268 150415,220227 150505,220227 150406),
CIRCULARSTRING(220227 150406,220227 150407,220227 150406))

--Neues ST_Collect-Array Konstrukt
SELECT ST_Collect(ARRAY(SELECT the_geom FROM sometable));

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

--wkt collect --
```

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

Siehe auch

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

8.11.5 ST_ConcaveHull

ST_ConcaveHull — Die konkave Hülle einer Geometrie stellt eine möglicherweise konkave Geometrie dar, welche alle Geometrien der Menge einschließt. Sie können es sich wie in Folie einpacken vorstellen.

Synopsis

geometry **ST_ConcaveHull**(geometry geomA, float target_percent, boolean allow_holes=false);

Beschreibung

Die konkave Hülle einer Geometrie stellt eine möglicherweise konkave Geometrie dar, welche alle Geometrien der Menge einschließt. Ob Polygone Lücken aufweisen dürfen ist standardmäßig auf FALSE gesetzt. Das Ergebnis ist niemals mehr als ein einzelnes Polygon.

Der Eingabeparameter "target_percent" ist der Flächenanteil der konvexen Hülle für den PostGIS eine Lösung annähert bevor es aufgibt oder beendet. Man kann sich eine konkave Hülle als eine Geometrie vorstellen, die man erhält wenn man einen Satz an Geometrien vakuumversiegelt. Ein target_percent von 1 führt zum selben Ergebnis wie die konvexe Hülle. Ein target_percent zwischen 0 und 0.99 ergibt eine kleinere Fläche als die konvexe Hülle. Dies unterscheidet sich von der konvexen Hülle, welche eher einem Gummiband entspricht, das den Satz an Geometrien umwickelt.

Wird üblicherweise auf Mehrfach/MULTI- und Sammelgeometrien/GeometryCollections angewandt. Obwohl es sich nicht um eine Aggregatfunktion handelt, können Sie es in Verbindung mit ST_Collect oder ST_Union verwenden um die konkave Hülle eines Satzes an Points/Linestrings/Polygons zu erhalten - ST_ConcaveHull(ST_Collect(somepointfield), 0.80).

Die Berechnung der konkaven Hülle ist wesentlich langsamer als bei der konvexen Hülle, aber die Geometrie wird besser umhüllt und ist auch nützlich bei der Bilderkennung.

Wird vom GEOS Modul ausgeführt



Note

Anmerkung - Wenn Sie es auf Punkte, Linienzüge oder Sammelgeometrien anwenden, verwenden Sie bitte ST_Collect. Bei Polygonen verwenden Sie bitte ST_Union, da es bei invaliden Geometrien fehlschlagen kann.



Note

Anmerkung - Umso kleiner Sie die target_percent ansetzen, desto länger dauert die Berechnung der konkaven Hülle und desto wahrscheinlicher ist es, dass topologische Fehler auftreten. Dies gilt auch umso größer die Anzahl der Kommastellen und die Anzahl der Punkte ist. Versuchen Sie zuerst eine 0.99, dies ist üblicherweise sehr schnell, manchmal so schnell wie die Berechnung der konvexen Hülle und ergibt meist einen viel besseren Wert als 99% Schrumpfung, da es fast immer über das Ziel hinausschießt. Als nächstes versuchen Sie 0.98; üblicherweise verlangsamt sich die Berechnung quadratisch. Um die Präzision und Kommastellen zu verringern, verwenden Sie bitte [ST_SimplifyPreserveTopology](#) oder [ST_SnapToGrid](#) nach ST_ConcaveHull. ST_SnapToGrid ist ein wenig schneller, kann allerdings zu invaliden Geometrien führen, während ST_SimplifyPreserveTopology meist die Validität der Geometrie erhält.

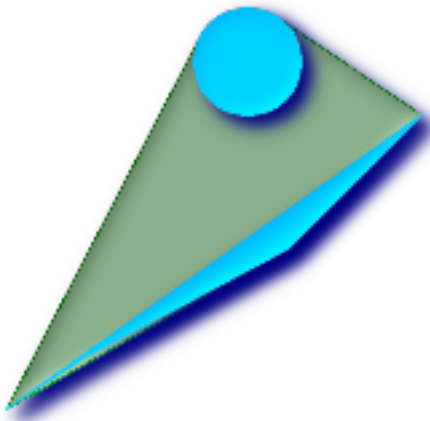
Ein konkreteres Beispiel und eine kurze Erklärung der Technik finden Sie unter http://www.bostongis.com/postgis_concavehull.snippet

Siehe auch Simon Greeners Artikel über ConcaveHull, die in Oracle 11G-R2 eingeführt wurde. http://www.spatialdbadvisor.com/oracle_spatial_tips_tricks/172/concave-hull-geometries-in-oracle-11gr2. Die Lösung, die wir mit 0.75 target_percent der konvexen Hülle erhalten ähnelt der Geometrieform, die Simon mit Oracle SDO_CONCAVEHULL_BOUNDARY erhält.

Verfügbarkeit: 2.0.0

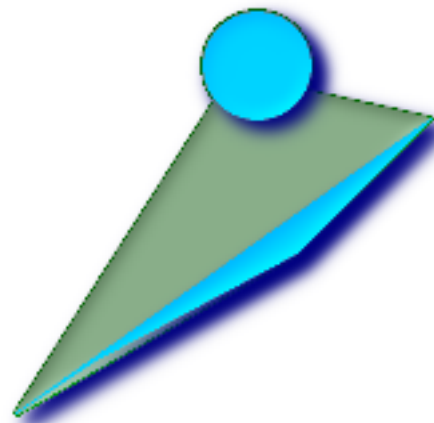
Beispiele

```
--Abschätzung der infizierten Fläche aus Punktbeobachtungen
SELECT d.disease_type,
       ST_ConcaveHull(ST_Collect(d.pnt_geom), 0.99) As geom
FROM disease_obs As d
GROUP BY d.disease_type;
```



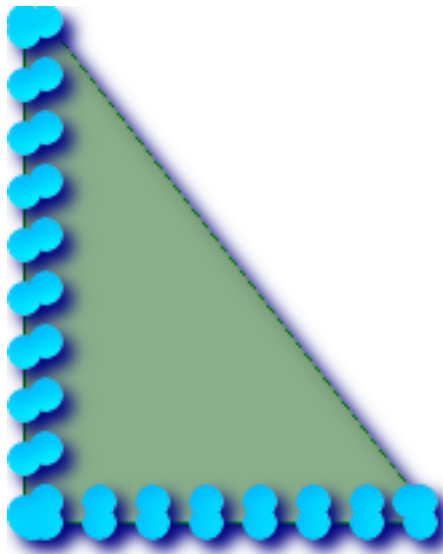
ST_ConcaveHull von 2 Polygonen die von einer konkaven Hülle mit 100% Schrumpfung umhüllt werden.

```
-- Geometrien mit konkaver Hülle ↔
-- überlagert
-- Ziel 100% Schrumpfung (dies entspricht ↔
-- der konvexen Hülle - keine Schrumpfung)
SELECT
  ST_ConcaveHull(
    ST_Union(ST_GeomFromText ↔
      ('POLYGON((175 150, 20 40,
                50 60, 125 100,
                175 150))'),
    ST_Buffer(ST_GeomFromText ↔
      ('POINT(110 170)'), 20)
    ), 1)
  As convexhull;
```



-- Geometrien überlagert mit der konkaven Hülle mit einem Zielwert von 90% der konvexen Hülle

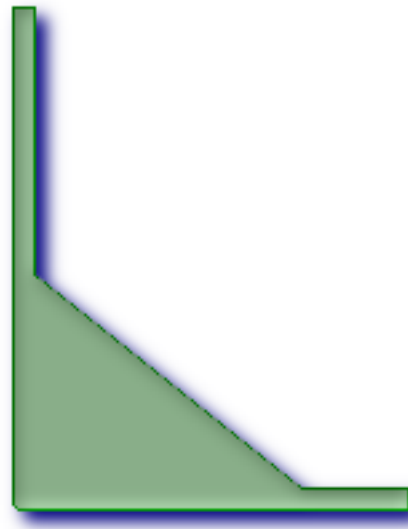
```
-- Geometrien überlagert mit der konkaven ↔
-- Hülle, die einen Zielwert von 90% Schru
SELECT
  ST_ConcaveHull(
    ST_Union(ST_GeomFromText ↔
      ('POLYGON((175 150, 20 40,
                50 60, 125 100,
                175 150))'),
    ST_Buffer(ST_GeomFromText ↔
      ('POINT(110 170)'), 20)
    ), 0.9)
  As target_90;
```



Als L angeordnete Punkte mit der konvexen Hülle überlagert.

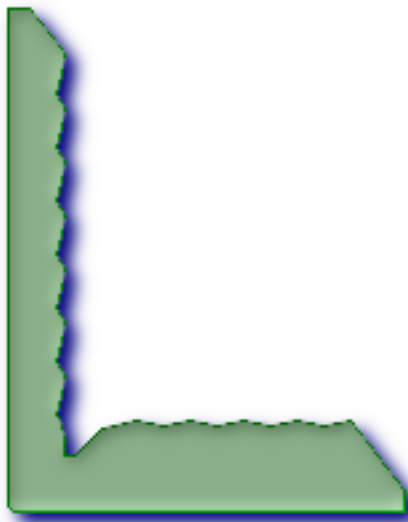
```
-- Erzeugt eine Tabelle mit 42 Punkten,  ↳
    die eine L-Form bilden
SELECT (ST_DumpPoints(ST_GeomFromText(
'MULTIPOINT(14 14,34 14,54 14,74 14,94  ↳
    14,114 14,134 14,
150 14,154 14,154 6,134 6,114 6,94 6,74  ↳
    6,54 6,34 6,
14 6,10 6,8 6,7 7,6 8,6 10,6 30,6 50,6  ↳
    70,6 90,6 110,6 130,
6 150,6 170,6 190,6 194,14 194,14 174,14  ↳
    154,14 134,14 114,
14 94,14 74,14 54,14 34,14 14)'))).geom
    INTO TABLE l_shape;

SELECT ST_ConvexHull(ST_Collect(geom))
FROM l_shape;
```



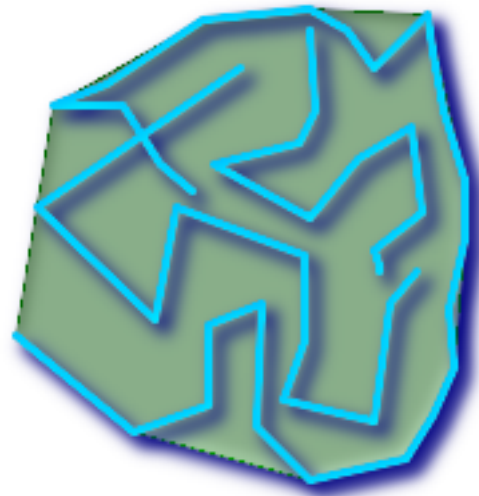
ST_ConcaveHull der als L angeordneten Punkte mit einem Zielwert von 99% der konvexen Hülle

```
SELECT ST_ConcaveHull(ST_Collect(geom),  ↳
    0.99)
FROM l_shape;
```

ST_ConcaveHull der als L angeordneten Punkte mit einem Zielwert von 80% der Fläche der konvexen Hülle

```
-- Konkave Hülle der als L angeordneten Punkte
-- mit einem Zielwert von 80% der konvexen Hülle
SELECT ST_ConcaveHull(ST_Collect(
  geom), 0.80)
FROM l_shape;
```



MultiLineString überlagert mit konvexer Hülle



MultiLineString überlagert mit der konkaven Hülle der LineStrings mit einem Zielwert von 99% - erster Versuch

```
SELECT ST_ConcaveHull(ST_GeomFromText('
  MULTILINESTRING((106 164,30 112,74 70,82
    130 62,122 40,156 32,162 76,172
      88),
    (132 178,134 148,128 136,96 128,132
      108,150 130,
    170 142,174 110,156 96,158 90,158 88),
    (22 64,66 28,94 38,94 68,114 76,112 30,
    132 10,168 18,178 34,186 52,184 74,190
      100,
    190 122,182 148,178 170,176 184,156
      164,146 178,
    132 186,92 182,56 158,36 150,62 150,76
      128,88 118))'),0.99)
```

Siehe auch

[ST_Collect](#), [ST_ConvexHull](#), [ST_SimplifyPreserveTopology](#), [ST_SnapToGrid](#)

8.11.6 ST_ConvexHull

ST_ConvexHull — The convex hull of a geometry represents the minimum convex geometry that encloses all geometries within the set.

Synopsis

geometry **ST_ConvexHull**(geometry geomA);

Beschreibung

Die konvexe Hülle einer Geometrie stellt die kleinste konvexe Geometrie dar, welche den Satz von Geometrien umschließt.

Man kann sich die konvexe Hülle als eine Geometrie vorstellen, die man erhält wenn man ein elastisches Band um einen Satz von Geometrien wickelt. Dies unterscheidet sich von der konkaven Hülle, die analog dem Vakuumverpacken von Geometrien ist.

Wird üblicherweise auf Mehrfach/MULTI- und Sammelgeometrien/GeometryCollections angewandt. Obwohl es sich nicht um eine Aggregatfunktion handelt, können Sie es in Verbindung mit `ST_Collect` verwenden um die konvexe Hülle einer Punktmenge zu erhalten. `ST_ConvexHull(ST_Collect(somepointfield))`.

Wird oft zur Bestimmung einer beeinträchtigten Fläche aus einem Satz von Beobachtungspunkten verwendet.

Wird vom GEOS Modul ausgeführt



This method implements the [OpenGIS Simple Features Implementation Specification for SQL 1.1](#). s2.1.1.3



This method implements the SQL/MM specification. SQL-MM 3: 5.1.16



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

Beispiele

```
-- Abschätzung des infizierten Gebietes aus Punktbeobachtungen
SELECT d.disease_type,
       ST_ConvexHull(ST_Collect(d.the_geom)) As the_geom
FROM disease_obs As d
GROUP BY d.disease_type;
```



Konvexe Hülle eines MultiLineString und eines MultiPoint gemeinsam mit dem MultiLineString und dem MultiPoint

```
SELECT ST_AsText(ST_ConvexHull(
    ST_Collect(
        ST_GeomFromText('MULTILINESTRING((100 190,10 8),(150 10, 20 30))'),
        ST_GeomFromText('MULTIPOINT(50 5, 150 30, 50 10, 10 10)')
    )));
---st_astext--
POLYGON((50 5,10 8,10 10,100 190,150 30,150 10,50 5))
```

Siehe auch

[ST_Collect](#), [ST_ConcaveHull](#), [ST_MinimumBoundingCircle](#)

8.11.7 ST_CurveToLine

ST_CurveToLine — Wandelt einen CIRCULARSTRING/CURVEPOLYGON in ein LINESTRING/POLYGON um

Synopsis

```
geometry ST_CurveToLine(geometry curveGeom);
geometry ST_CurveToLine(geometry curveGeom, integer segments_per_qtr_circle);
geometry ST_CurveToLine(geometry curveGeom, float tolerance, integer tolerance_type, integer flags);
```

Beschreibung

Konvertiert einen Kreisbogen/CircularString in einen normalen LineString oder ein CurvePolygon in ein Polygon. Nützlich zur Ausgabe an Geräten, die Kreisbögen nicht 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: 1.2.2?

Erweiterung: ab 2.4.0 kann die Toleranz über die 'maximale Abweichung' und den 'maximalen Winkel' angegeben werden. Die symmetrische Ausgabe wurde hinzugefügt.



This method implements the [OpenGIS Simple Features Implementation Specification for SQL 1.1](#).



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



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



This method supports Circular Strings and Curves

Beispiele

```
SELECT ST_AsText(ST_CurveToLine(ST_GeomFromText('CIRCULARSTRING(220268 150415,220227 150505,220227 150406)')));

--Result --
LINESTRING(220268 150415,220269.95064912 150416.539364228,220271.823415575 150418.17258804,220273.613787707 150419.895736857,
220275.317452352 150421.704659462,220276.930305234 150423.594998003,220278.448460847 150425.562198489,
220279.868261823 150427.60152176,220281.186287736 150429.708054909,220282.399363347 150431.876723113,
220283.50456625 150434.10230186,220284.499233914 150436.379429536,220285.380970099 150438.702620341,220286.147650624 150441.066277505,
220286.797428488 150443.464706771,220287.328738321 150445.892130112,220287.740300149 150448.342699654,
220288.031122486 150450.810511759,220288.200504713 150453.289621251,220288.248038775 150455.77405574,
220288.173610157 150458.257830005,220287.977398166 150460.734960415,220287.659875492 150463.199479347,
220287.221807076 150465.64544956,220286.664248262 150468.066978495,220285.988542259 150470.458232479,220285.196316903 150472.81345077,
220284.289480732 150475.126959442,220283.270218395 150477.39318505,220282.140985384 150479.606668057,
220280.90450212 150481.762075989,220279.5637474 150483.85421628,220278.12195122 150485.87804878,
220276.582586992 150487.828697901,220274.949363179 150489.701464356,220273.226214362 150491.491836488,
220271.417291757 150493.195501133,220269.526953216 150494.808354014,220267.559752731 150496.326509628,
220265.520429459 150497.746310603,220263.41389631 150499.064336517,220261.245228106 150500.277412127,
220259.019649359 150501.38261503,220256.742521683 150502.377282695,220254.419330878 150503.259018879,
```

```

220252.055673714 150504.025699404,220249.657244448 150504.675477269,220247.229821107 ↵
150505.206787101,
220244.779251566 150505.61834893,220242.311439461 150505.909171266,220239.832329968 ↵
150506.078553494,
220237.347895479 150506.126087555,220234.864121215 150506.051658938,220232.386990804 ↵
150505.855446946,
220229.922471872 150505.537924272,220227.47650166 150505.099855856,220225.054972724 ↵
150504.542297043,
220222.663718741 150503.86659104,220220.308500449 150503.074365683,
220217.994991777 150502.167529512,220215.72876617 150501.148267175,
220213.515283163 150500.019034164,220211.35987523 150498.7825509,
220209.267734939 150497.441796181,220207.243902439 150496,
220205.293253319 150494.460635772,220203.420486864 150492.82741196,220201.630114732 ↵
150491.104263143,
220199.926450087 150489.295340538,220198.313597205 150487.405001997,220196.795441592 ↵
150485.437801511,
220195.375640616 150483.39847824,220194.057614703 150481.291945091,220192.844539092 ↵
150479.123276887,220191.739336189 150476.89769814,
220190.744668525 150474.620570464,220189.86293234 150472.297379659,220189.096251815 ↵
150469.933722495,
220188.446473951 150467.535293229,220187.915164118 150465.107869888,220187.50360229 ↵
150462.657300346,
220187.212779953 150460.189488241,220187.043397726 150457.710378749,220186.995863664 ↵
150455.22594426,
220187.070292282 150452.742169995,220187.266504273 150450.265039585,220187.584026947 ↵
150447.800520653,
220188.022095363 150445.35455044,220188.579654177 150442.933021505,220189.25536018 ↵
150440.541767521,
220190.047585536 150438.18654923,220190.954421707 150435.873040558,220191.973684044 ↵
150433.60681495,
220193.102917055 150431.393331943,220194.339400319 150429.237924011,220195.680155039 ↵
150427.14578372,220197.12195122 150425.12195122,
220198.661315447 150423.171302099,220200.29453926 150421.298535644,220202.017688077 ↵
150419.508163512,220203.826610682 150417.804498867,
220205.716949223 150416.191645986,220207.684149708 150414.673490372,220209.72347298 ↵
150413.253689397,220211.830006129 150411.935663483,
220213.998674333 150410.722587873,220216.22425308 150409.61738497,220218.501380756 ↵
150408.622717305,220220.824571561 150407.740981121,
220223.188228725 150406.974300596,220225.586657991 150406.324522731,220227 150406)

--3D Beispiel
SELECT ST_AsEWKT(ST_CurveToLine(ST_GeomFromEWKT('CIRCULARSTRING(220268 150415 1,220227 ↵
150505 2,220227 150406 3)')));
Output
-----
LINESTRING(220268 150415 1,220269.95064912 150416.539364228 1.0181172856673,
220271.823415575 150418.17258804 1.03623457133459,220273.613787707 150419.895736857 ↵
1.05435185700189,...AD INFINITUM ....
220225.586657991 150406.324522731 1.32611114201132,220227 150406 3)

--nur 2 Segmente zur Annäherung des Viertelkreises
SELECT ST_AsText(ST_CurveToLine(ST_GeomFromText('CIRCULARSTRING(220268 150415,220227 ↵
150505,220227 150406)'),2));
st_astext
-----
LINESTRING(220268 150415,220287.740300149 150448.342699654,220278.12195122 ↵
150485.87804878,
220244.779251566 150505.61834893,220207.243902439 150496,220187.50360229 150462.657300346,
220197.12195122 150425.12195122,220227 150406)

-- Sicherstellen, dass die angenäherte Linie nicht weiter als 20 Einheiten
-- von der ursprünglichen Kurve entfernt liegt, und das Ergebnis richtungsunabhängig machen

```

```
SELECT ST_AsText(ST_CurveToLine(
  'CIRCULARSTRING(0 0,100 -100,200 0) '::geometry,
    20, -- Tolerance
    1, -- oberhalb die maximale Entfernung zwischen Kurve und Linie
    1 -- Symmetrie Flag
));
st_astext
-----
LINESTRING(0 0,50 -86.6025403784438,150 -86.6025403784439,200 -1.1331077795296e-13,200 0)
```

Siehe auch

[ST_LineToCurve](#)

8.11.8 ST_DelaunayTriangles

ST_DelaunayTriangles — Gibt die Delaunay-Triangulierung für gegebene Punkte zurück.

Synopsis

geometry **ST_DelaunayTriangles**(geometry g1, float tolerance, int4 flags);

Beschreibung

Gibt eine **Delaunay-Triangulierung** rund um die Knoten der Eingabegeometrie zurück. Die Ausgabe ist eine COLLECTION von Polygonen (flags=0), ein MultiLineString (flags=1) oder ein TIN (flags=2). Die Toleranz, sofern angegeben, wird zum Zusammenfangen von Knoten verwendet.

Verfügbarkeit: 2.1.0 - benötigt GEOS >= 3.4.0.

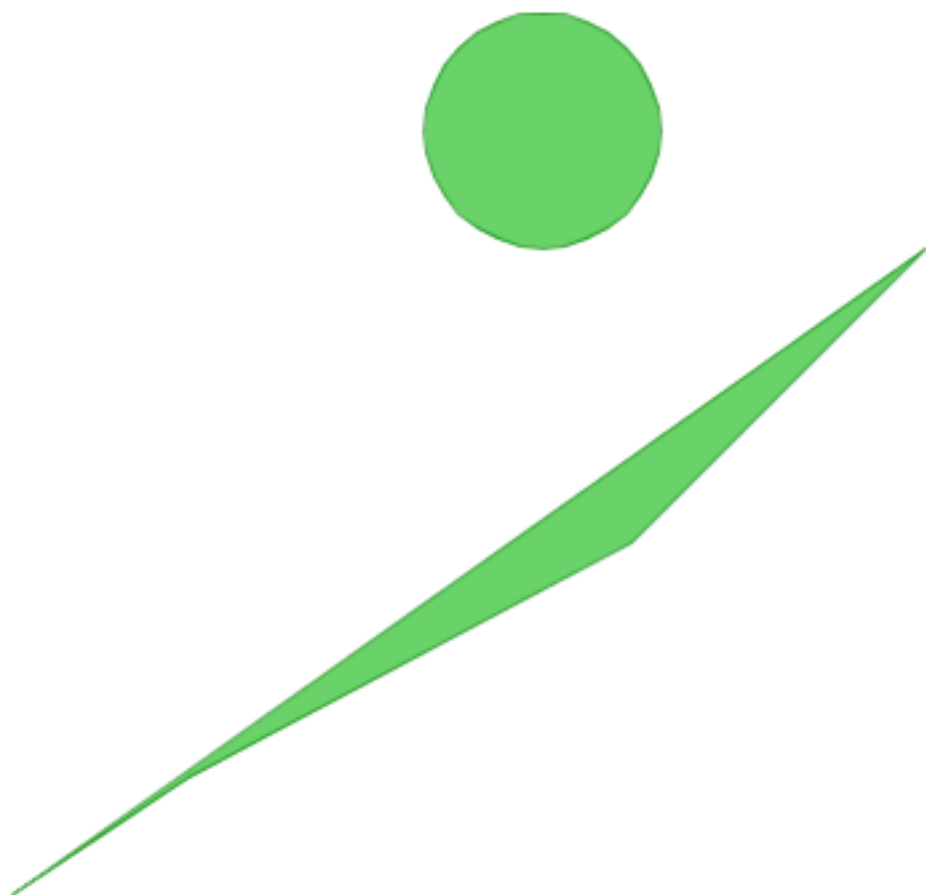


This function supports 3d and will not drop the z-index.



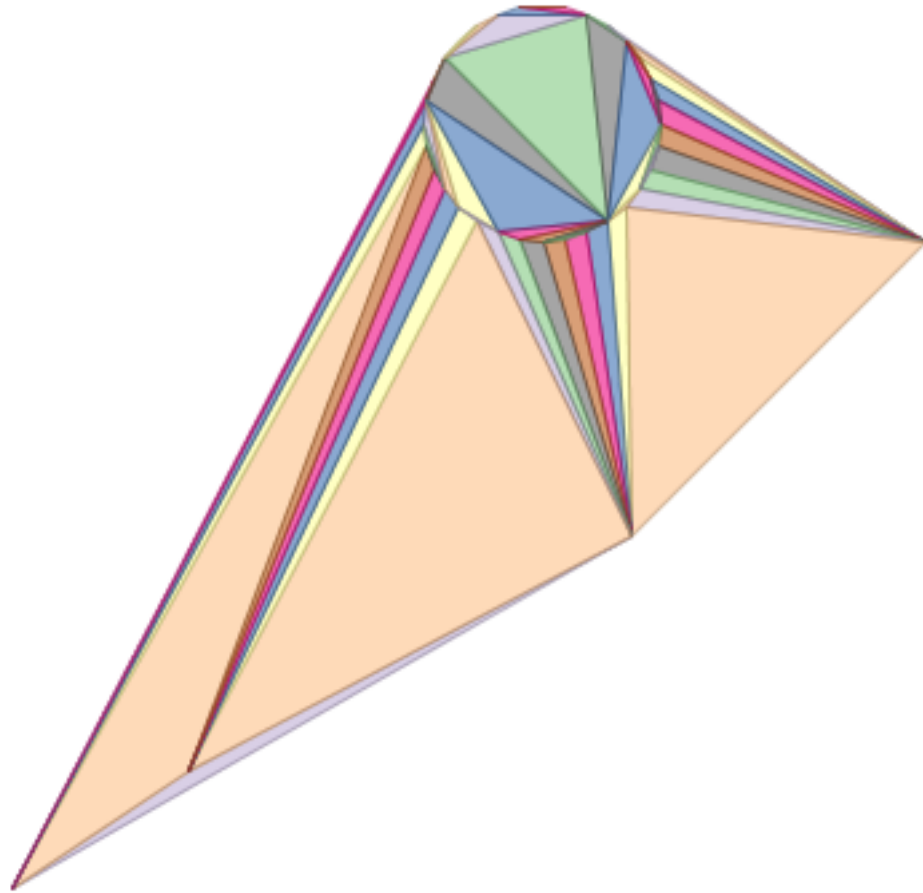
This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).

2D Beispiele



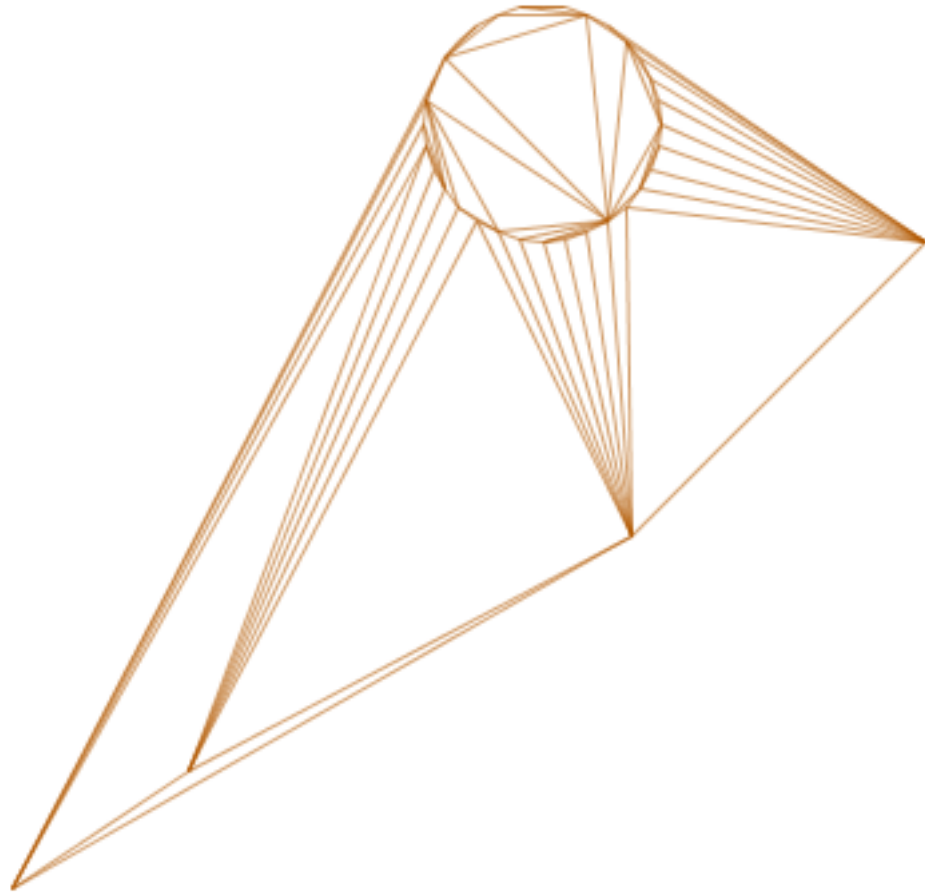
Ursprüngliche Polygone

```
-- Unsere ursprüngliche Geometrie --
ST_Union(ST_GeomFromText('POLYGON((175 150, 20 40,
                    50 60, 125 100, 175 150))'),
ST_Buffer(ST_GeomFromText('POINT(110 170)'), 20)
)
```



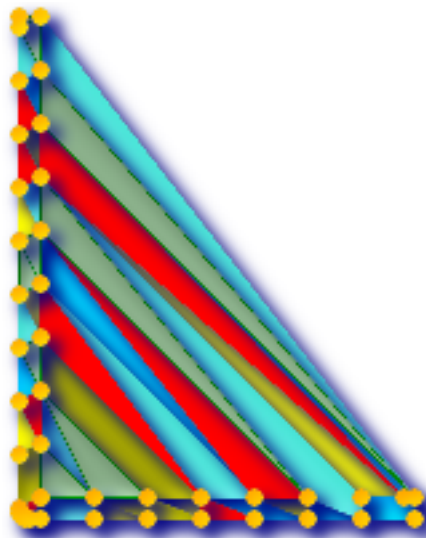
ST_DelaunayTriangles von 2 Polygonen: delaunay triangulierte Polygone, jedes der Dreiecke ist in einer eigenen Farbe dargestellt

```
-- geometries overlaid multilinestring triangles
SELECT
  ST_DelaunayTriangles(
    ST_Union(ST_GeomFromText('POLYGON((175 150, 20 40,
      50 60, 125 100, 175 150))'),
    ST_Buffer(ST_GeomFromText('POINT(110 170)'), 20)
  )
  As dtriag;
```

-- Delaunay-Dreiecke als MultiLinestring

```
SELECT
  ST_DelaunayTriangles(
    ST_Union(ST_GeomFromText('POLYGON((175 150, 20 40,
      50 60, 125 100, 175 150))'),
    ST_Buffer(ST_GeomFromText('POINT(110 170)'), 20)
  ), 0.001, 1)
As dtriag;
```



-- Delaunay Dreiecke von 45 Punkten als 55 Dreieckspolygone

```
-- erzeugt eine Tabelle mit 42 Punkten die eine L-Form bilden
SELECT (ST_DumpPoints(ST_GeomFromText (
'MULTIPOINT(14 14,34 14,54 14,74 14,94 14,114 14,134 14,
150 14,154 14,154 6,134 6,114 6,94 6,74 6,54 6,34 6,
14 6,10 6,8 6,7 7,6 8,6 10,6 30,6 50,6 70,6 90,6 110,6 130,
6 150,6 170,6 190,6 194,14 194,14 174,14 154,14 134,14 114,
14 94,14 74,14 54,14 34,14 14)'))).geom
      INTO TABLE l_shape;
-- Ausgabe als einzelne Dreieckspolygone
SELECT ST_AsText((ST_Dump(geom)).geom) As wkt
FROM ( SELECT ST_DelaunayTriangles(ST_Collect(geom)) As geom
FROM l_shape) As foo;

---wkt ---
POLYGON((6 194,6 190,14 194,6 194))
POLYGON((14 194,6 190,14 174,14 194))
POLYGON((14 194,14 174,154 14,14 194))
POLYGON((154 14,14 174,14 154,154 14))
POLYGON((154 14,14 154,150 14,154 14))
POLYGON((154 14,150 14,154 6,154 14))
:
:
```

3D Beispiele

```
-- 3D-MULTIPOINT --
SELECT ST_AsText(ST_DelaunayTriangles(ST_GeomFromText (
'MULTIPOINT Z(14 14 10,
150 14 100,34 6 25, 20 10 150)')))) As wkt;

-----wkt-----
GEOMETRYCOLLECTION Z (POLYGON Z ((14 14 10,20 10 150,34 6 25,14 14 10))
,POLYGON Z ((14 14 10,34 6 25,150 14 100,14 14 10)))
```

Siehe auch

[ST_ConcaveHull](#), [ST_Dump](#)

8.11.9 ST_Difference

ST_Difference — Gibt eine Geometrie zurück, die jenen Teil der Geometrie A abbildet, der sich nicht mit der Geometrie B überschneidet.

Synopsis

```
geometry ST_Difference(geometry geomA, geometry geomB);
```

Beschreibung

Gibt eine Geometrie zurück, die jenen Teil der Geometrie A abbildet, der sich nicht mit der Geometrie B überschneidet. Man kann sich das als $\text{GeometryA} - \text{ST\_Intersection}(\text{A}, \text{B})$ vorstellen. Wenn A zur Gänze in B enthalten ist wird eine leere GeometryCollection zurückgegeben.

**Note**

Anmerkung: Die Reihenfolge spielt eine Rolle. B - A gibt immer einen Teil von B zurück

Wird vom GEOS Modul ausgeführt

**Note**

Nicht mit einer GeometryCollection als Übergabewert aufrufen



This method implements the [OpenGIS Simple Features Implementation Specification for SQL 1.1](#). s2.1.1.3

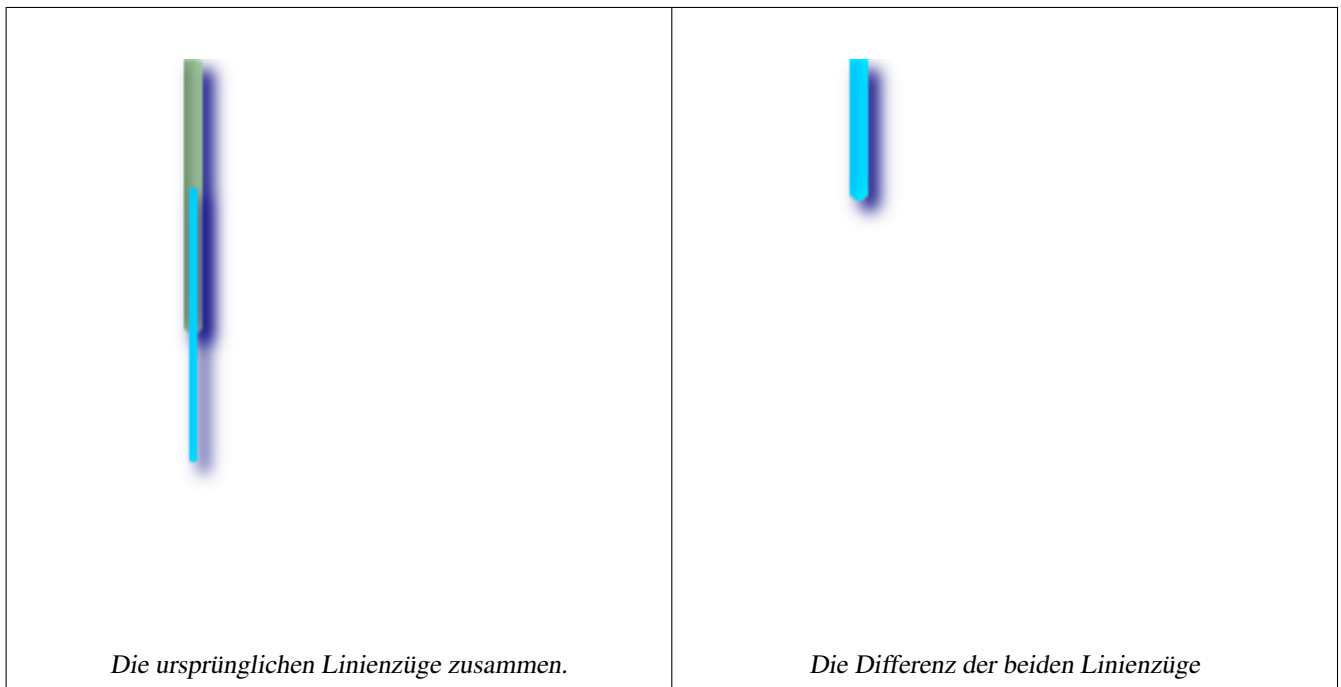


This method implements the SQL/MM specification. SQL-MM 3: 5.1.20



This function supports 3d and will not drop the z-index. Allerdings scheint nur x y bei der Berechnung der Differenz herangezogen und anschließend an den Z-Index zurückgeheftet zu werden

Beispiele



--Sicher bei 2D. Dies sind dieselben Geometrien als bei `st_symdifference` angezeigt

```
SELECT ST_AsText(
    ST_Difference(
        ST_GeomFromText('LINESTRING(50 100, 50 200)'),
        ST_GeomFromText('LINESTRING(50 50, 50 150)')
    )
);

st_astext
-----
LINESTRING(50 150,50 200)
```

--verhält sich bei 3D nicht richtig

```
SELECT ST_AsEWKT(ST_Difference(ST_GeomFromEWKT('MULTIPOINT(-118.58 38.38 5,-118.60 38.329 6,-118.614 38.281 7)'), ST_GeomFromEWKT('POINT(-118.614 38.281 5)')));

st_asewkt
-----
MULTIPOINT(-118.6 38.329 6,-118.58 38.38 5)
```

Siehe auch

[ST_SymDifference](#)

8.11.10 ST_Dump

`ST_Dump` — Gibt einen Satz an `geometry_dump` (`geom,path`) Zeilen zurück, aus denen die Geometrie `g1` zusammengesetzt ist.

Synopsis

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

Beschreibung

Dies ist eine Funktion mit Ergebnismenge (set-returning function, SRF). Sie gibt eine Menge an geometry_dump Zeilen zurück, welche aus einer Geometrie (geom) und einem Feld aus Ganzzahlen (path) gebildet werden. Wenn die Eingabegeometrie ein einfacher Typ (POINT,LINESTRING,POLYGON) ist, wird ein einzelner Datensatz, mit einem leeren path-Feld und der Eingabegeometrie als geom, zurückgegeben. Falls die Eingabegeometrie eine COLLECTION oder eine MULTI* ist, wird für jede Komponente der COLLECTION ein Datensatz ausgegeben, wobei der path die Lage der Komponente in der COLLECTION angibt.

ST_Dump ist nützlich beim Ausladen von Geometrien. Es ist das Gegenteil von GROUP BY, insofern als neue Zeilen erzeugt werden. Es kann zum Beispiel verwendet werden um MultiPolygone in Polygone überzuführen.

Erweiterung: 2.0.0 Unterstützung für polyedrische Oberflächen, Dreiecke und TIN eingeführt.

Verfügbarkeit: PostGIS 1.0.0RC1. Benötigt PostgreSQL 7.3 oder höher.



Note

Vor 1.3.4 ist diese Funktion abgestürzt, wenn die Geometrien KURVEN/CURVES enthalten. Dies wurde mit 1.3.4+ behoben



This method supports Circular Strings and Curves



This function supports Polyhedral surfaces.



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



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

Standardbeispiele

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

-- Einen zusammengesetzten Bogen in seine Linienzüge und Kreisbögen aufbrechen
SELECT ST_AsEWKT(a.geom), ST_HasArc(a.geom)
FROM ( SELECT (ST_Dump(p_geom)).geom AS geom
       FROM (SELECT ST_GeomFromEWKT('COMPOUNDCURVE(CIRCULARSTRING(0 0, 1 1, 1 0),(1 0, 0 1))') AS p_geom) AS b
     ) AS a;
      st_asewkt                | st_hasarc
-----+-----
CIRCULARSTRING(0 0,1 1,1 0) | t
LINESTRING(1 0,0 1)         | f
(2 rows)
```

Beispiele für polyedrische Oberflächen, TIN und Dreiecke

```
-- Beispiel für eine polyedrische Oberfläche
-- Eine polyedrische Oberfläche in Einzelflächen zerlegen
SELECT (a.p_geom).path[1] As path, ST_AsEWKT((a.p_geom).geom) As geom_ewkt
FROM (SELECT ST_Dump(ST_GeomFromEWKT('POLYHEDRALSURFACE(
((0 0 0, 0 0 1, 0 1 1, 0 1 0, 0 0 0)),
```

```
((0 0 0, 0 1 0, 1 1 0, 1 0 0, 0 0 0)), ((0 0 0, 1 0 0, 1 0 1, 0 0 1, 0 0 0)), ((1 1 0, 1 1 0, 1 1 0, 1 0 1, 1 0 0, 1 1 0)),
((0 1 0, 0 1 1, 1 1 1, 1 1 0, 0 1 0)), ((0 0 1, 1 0 1, 1 1 1, 0 1 1, 0 0 1))
)') ) AS p_geom ) AS a;
```

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

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

Siehe auch

[geometry_dump](#), [Section 14.6](#), [ST_Collect](#), [ST_Collect](#), [ST_GeometryN](#)

8.11.11 ST_DumpPoints

ST_DumpPoints — Gibt eine Menge von `geometry_dump` (`geom,path`) Zeilen aller Punkte zurück, aus denen die Geometrie zusammengesetzt ist.

Synopsis

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

Beschreibung

Diese Funktion mit Ergebnismenge (set-returning function, SRF) gibt eine Menge an `geometry_dump` Zeilen zurück, die aus einer Geometrie (`geom`) und einem Feld aus Ganzzahlen (`path`) aufgebaut ist.

Die `geom` Komponente des `geometry_dump` besteht aus allen POINTs aus denen die eingegebene Geometrie besteht

Die *path* Komponente des *geometry_dump* (ein *integer[]*) ist eine Indexreferenz, welche die POINTs der eingegebenen Geometrie abzählt. Wenn zum Beispiel ein LINESTRING eingegeben wird, wird ein path von {i} zurückgegeben, wobei i die nte Koordinate in dem LINESTRING ist. Falls ein POLYGON eingegeben wird, wird ein Pfad von {i, j} zurückgegeben, wobei i die Ringnummer ist (1 ist äußerer Ring; die inneren Ringe folgen) und j zählt die POINTs ab (Index beginnt mit 1).

Erweiterung: 2.1.0 Schnellere Geschwindigkeit. In nativem C neu implementiert.

Erweiterung: 2.0.0 Unterstützung für polyedrische Oberflächen, Dreiecke und TIN eingeführt.

Verfügbarkeit: 1.5.0



This method supports Circular Strings and Curves



This function supports Polyhedral surfaces.



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



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

Eine Tabelle von LineStrings auf klassische Art in Knoten zerlegen

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

Geometrische Standardbeispiele



```

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

```

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

(29 rows)

Beispiele für polyedrische Oberflächen, TIN und Dreiecke

```

-- polyedrische Würfeloberfläche --
SELECT (g.gdump).path, ST_AsEWKT((g.gdump).geom) as wkt
FROM
  (SELECT

```



```

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

```

```

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

```

```

-- TIN --

```

```

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

```

Siehe auch

[geometry_dump](#), [Section 14.6](#), [ST_Dump](#), [ST_DumpRings](#)

8.11.12 ST_DumpRings

ST_DumpRings — Gibt eine Menge an `geometry_dump` Zeilen zurück, welche die äußeren und inneren Ringe eines Polygons darstellen.

Synopsis

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

Beschreibung

Dies ist eine Funktion mit Ergebnismenge (SRF). Sie gibt eine Menge an `geometry_dump` Zeilen zurück, die durch einen `Integer[]` und eine `Geometrie`, beziehungsweise durch die Aliasse "path" und "geom", definiert sind. Das Attribut "path" enthält den Index des Polygonringes als einzelne Ganzzahl: 0 für die Hülle, >0 für Lücken/Inseln. Das Attribut "geom" enthält den entsprechenden Ring als Polygon.

Verfügbarkeit: PostGIS 1.1.3. Benötigt PostgreSQL 7.3 oder höher.



Note

Dies funktioniert nur mit POLYGON Geometrien und nicht mit MULTIPOLYGONS



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

Beispiele

```
SELECT sometable.field1, sometable.field1,
       (ST_DumpRings(sometable.the_geom)).geom As the_geom
FROM sometableOfpolys;

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

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

Siehe auch

[geometry_dump](#), [Section 14.6](#), [ST_Dump](#), [ST_ExteriorRing](#), [ST_InteriorRingN](#)

8.11.13 ST_FlipCoordinates

ST_FlipCoordinates — Gibt eine Version der gegebenen Geometrie zurück, wobei die X und Y Achse vertauscht sind. Nützlich wenn man Geoobjekte in Breite/Länge vorliegen hat und dies beheben möchte.

Synopsis

```
geometry ST_FlipCoordinates(geometry geom);
```

Beschreibung

Gibt eine Version der gegebenen Geometrie zurück, wobei die X und Y Achse vertauscht sind.

Verfügbarkeit: 2.0.0



This method supports Circular Strings and Curves



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



This function supports M coordinates.



This function supports Polyhedral surfaces.



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

Beispiel

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

Siehe auch

[ST_SwapOrdinates](#)

8.11.14 ST_GeneratePoints

ST_GeneratePoints — Wandelt ein Polygon oder ein MultiPolygon in einen MultiPoint um, welcher aus wahllos angeordneten, innerhalb der ursprünglichen Flächen liegenden Punkten besteht.

Synopsis

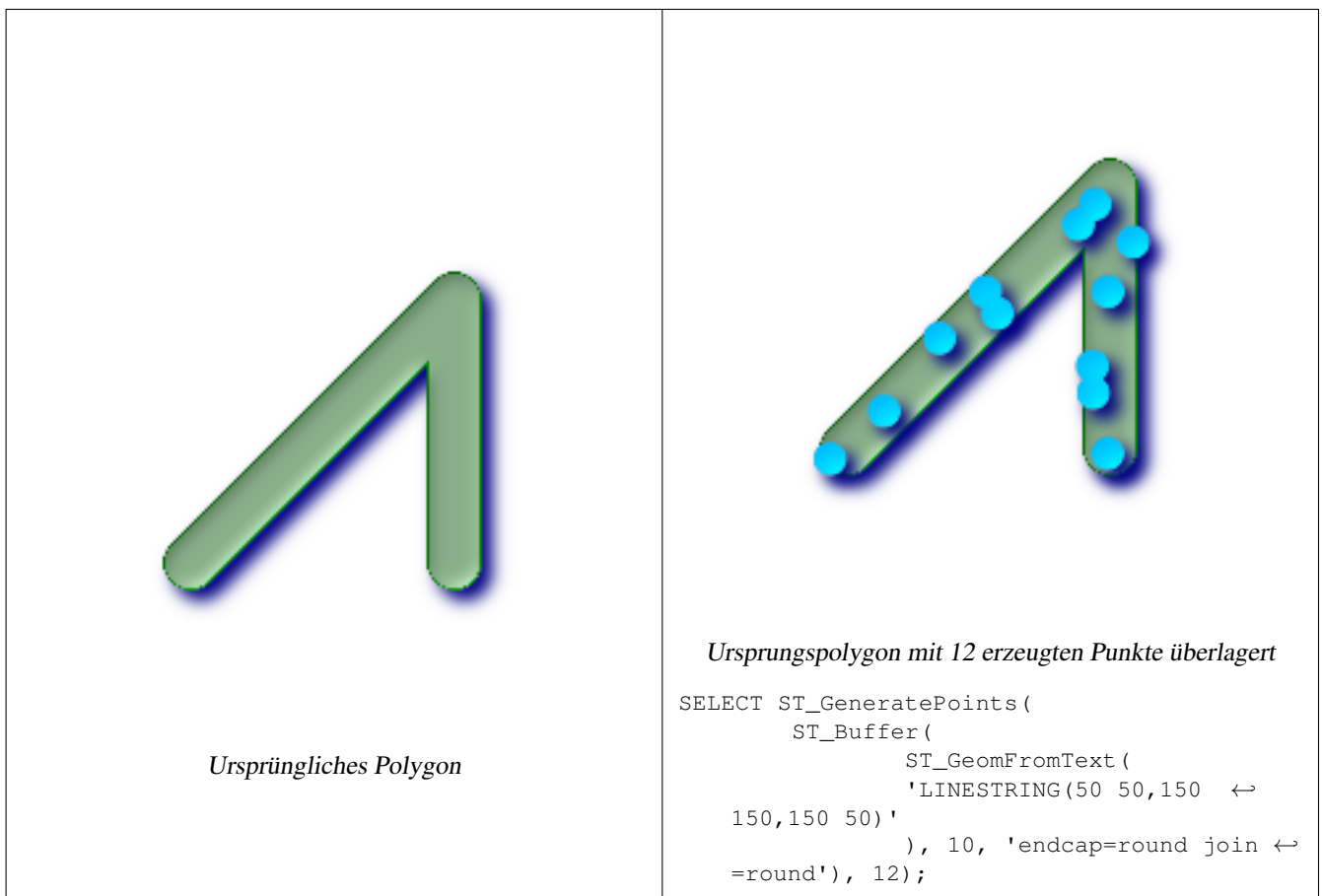
geometry **ST_GeneratePoints**(g geometry , npoints numeric);

Beschreibung

ST_GeneratePoints erzeugt solange pseudomäßige Zufallspunkte, bis die angefragte Anzahl innerhalb der Eingabefläche gefunden wurde.

Verfügbarkeit: 2.3.0

Beispiele



8.11.15 ST_Intersection

ST_Intersection — (T) Gibt eine Geometrie zurück, welche den gemeinsamen Anteil von geomA und geomB repräsentiert.

Synopsis

```
geometry ST_Intersection( geometry geomA , geometry geomB );
geography ST_Intersection( geography geogA , geography geogB );
```

Beschreibung

Gibt eine Geometrie zurück, welche der mengentheoretischen Verschneidung der Geometrien entspricht.

Mit anderen Worten - jener Teil von der Geometrie A und der Geometrie B den sich die beiden Geometrien teilen.

Wenn sich die Geometrien keinen gemeinsamen Raum teilen (sind getrennt), so wird eine leere GeometryCollection zurückgegeben.

ST_Intersection ist in Verbindung mit ST_Intersects zum Ausschneiden von Geometrien sehr nützlich. Wie bei einem Umgebungsrechteck, einem Buffer, oder bei regionalen Abfragen, bei denen nur jener Teil der Geometrie zurückgegeben werden soll, der in einem bestimmten Staat oder einer bestimmten Interessensregion liegt.

Note

Geographie: Beim geographischen Datentyp handelt es sich lediglich um einen schlanken Adapter, der um die geometrische Implementation herumgelegt wurde. Zuerst wird die passende SRID für das Umgebungsrechteck der beiden geographischen Objekte bestimmt (wenn die geographischen Objekte innerhalb eines UTM Überlappungsbereichs, aber nicht in derselben UTM Zone liegen, wird eine Zone herausgepickt) (bevorzugt UTM oder Lambert Azimuthal Equal Area (LAEA) Nord/Süd Pol, im schlimmsten Fall wird auf Mercator zurückgegriffen), dann im planaren Koordinatenreferenzsystem verschnitten und anschließend nach WGS84 Geographie zurück transformiert.

**Important**

Bitte nicht mit einer `GEOMETRYCOLLECTION` als Parameter aufrufen

**Warning**

Diese Funktion löscht die Werte der M-Koordinate, falls vorhanden.

**Warning**

Wenn Sie mit 3D-Geometrie arbeiten kann es sinnvoll sein die SFGCAL basierte Funktion `ST_3DIntersection` zu verwenden, da diese eine korrekte 3D Verschneidung mit 3D Geometrie ausführt. Obwohl die Funktion mit der Z-Koordinate arbeitet, führt sie eine Mittelung der Z-Koordinatenwerte durch, wenn `postgis.backend=geos`. Auch wenn `postgis.backend=sfcgal` wird die Z-Koordinate ignoriert und eine 2D-Geometrie ausgegeben. Siehe `postgis.backend` für Details.

Wird vom GEOS Modul ausgeführt



This method is also provided by SFCGAL backend.

Verfügbarkeit: 1.5 die Unterstützung des geographischen Datentyps wurde eingeführt



This method implements the [OpenGIS Simple Features Implementation Specification for SQL 1.1](#). s2.1.1.3



This method implements the SQL/MM specification. SQL-MM 3: 5.1.18

Beispiele

```
SELECT ST_AsText(ST_Intersection('POINT(0 0)::geometry, 'LINESTRING ( 2 0, 0 2 ) '::
    geometry));
st_astext
-----
GEOMETRYCOLLECTION EMPTY
(1 row)
SELECT ST_AsText(ST_Intersection('POINT(0 0)::geometry, 'LINESTRING ( 0 0, 0 2 ) '::
    geometry));
st_astext
-----
POINT(0 0)
(1 row)

---Schneidet alle Linien (Wege) nach dem Staat aus (wir nehmen an, dass es sich bei der
    Geometrie um Polygone oder MultiPolygone handelt)
```

```
-- Anmerkung: wir behalten nur jene Verschneidungen, die zu einem LineString oder ↵
MultiLineString führen, da wir uns nicht
-- um jene Wege kümmern, die nur einen Punkt teilen
-- ST_Dump wird benötigt um eine Sammelgeometrie/GeometryCollection in individuelle ↵
einzelne MULTI* Teile zu zerlegen
-- der untere Code ist sehr generisch und funktioniert auch mit Polygonen etc. indem man ↵
die Where-Klausel ändert
SELECT clipped.gid, clipped.f_name, clipped_geom
FROM (SELECT trails.gid, trails.f_name, (ST_Dump(ST_Intersection(country.the_geom, trails. ↵
the_geom))).geom As clipped_geom
FROM country
        INNER JOIN trails
        ON ST_Intersects(country.the_geom, trails.the_geom)) As clipped
WHERE ST_Dimension(clipped.clipped_geom) = 1 ;

--Bei Polygonen, z.B. polygonale Landgrenzen, können Sie auch den manchmal schnelleren Hack ↵
verwenden, indem Sie alles mit 0.0 puffern
-- außer ein Polygon führt zu einer leeren Sammelgeometrie/GeometryCollection
--(wie bei einer Sammelgeometrie, die Polygone, Linien und Punkte enthält)
-- mit 0.0 gepuffert hinterlässt nur die Polygone und löst die Hülle der Sammelgeometrie ↵
auf
SELECT poly.gid, ST_Multi(ST_Buffer(
                                ST_Intersection(country.the_geom, poly.the_geom),
                                0.0)
                                ) As clipped_geom
FROM country
        INNER JOIN poly
        ON ST_Intersects(country.the_geom, poly.the_geom)
WHERE Not ST_IsEmpty(ST_Buffer(ST_Intersection(country.the_geom, poly.the_geom) ↵
,0.0));
```

Beispiele: 2.5D-isch

GEOS ist standardmäßig das Back-end, falls nicht anders gesetzt. Beachten Sie bitte, dass dies keine richtige Verschneidung ist. Vergleiche das gleiche Beispiel mit [ST_3DIntersection](#).

```
set postgis.backend=geos;
select ST_AsText(ST_Intersection(linestring, polygon)) As wkt
from ST_GeomFromText('LINESTRING Z (2 2 6,1.5 1.5 7,1 1 8,0.5 0.5 8,0 0 10)') AS ↵
linestring
CROSS JOIN ST_GeomFromText('POLYGON((0 0 8, 0 1 8, 1 1 8, 1 0 8, 0 0 8))') AS polygon;

        st_astext
-----
LINESTRING Z (1 1 8,0.5 0.5 8,0 0 10)
```

Wenn PostGIS mit SFCGAL Unterstützung kompiliert wurde, haben Sie die Möglichkeit sfcgal zu verwenden. Beachten Sie aber bitte, dass hierbei beide Geometrien grundsätzlich auf 2D reduziert werden, bevor die Verschneidung ausgeführt wird und das ein Ergebnis äquivalent zu ST_Force2D zurückgegeben wird, welches eine 2D-Geometrie ist.

```
set postgis.backend=sfcgal;
select ST_AsText(ST_Intersection(linestring, polygon)) As wkt
from ST_GeomFromText('LINESTRING Z (2 2 6,1.5 1.5 7,1 1 8,0.5 0.5 8,0 0 10)') AS ↵
linestring
CROSS JOIN ST_GeomFromText('POLYGON((0 0 8, 0 1 8, 1 1 8, 1 0 8, 0 0 8))') AS polygon;

        wkt
-----
MULTILINESTRING((0.5 0.5,0 0),(1 1,0.5 0.5))
```

Siehe auch

[ST_3DIntersection](#), [ST_Difference](#), [ST_Dimension](#), [ST_Dump](#), [ST_Force2D](#), [ST_SymDifference](#), [ST_Intersects](#), [ST_Multi](#)

8.11.16 ST_LineToCurve

ST_LineToCurve — Wandelt einen LINESTRING/POLYGON in einen CIRCULARSTRING, CURVEPOLYGON um

Synopsis

geometry **ST_LineToCurve**(geometry geomANoncircular);

Beschreibung

Wandelt einen einfachen LineString/Polygon in Kreisbögen/CIRCULARSTRINGs und Kurvenpolygone um. Beachten Sie, dass wesentlich weniger Punkte zur Beschreibung des Kurvenäquivalents benötigt werden.



Note
Wenn der gegebene LINESTRING/POLYGON nicht genug gekrümmt ist um eine deutliche Kurve zu repräsentieren, wird die Geometrie von der Funktion unverändert zurückgegeben.

Verfügbarkeit: 1.2.2?



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



This method supports Circular Strings and Curves

Beispiele

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

curvedastext                                     non_curvedastext
-----|-----
CURVEPOLYGON(CIRCULARSTRING(4 3,3.12132034355964 0.878679656440359, | POLYGON((4
      3,3.94235584120969 2.41472903395162,3.77163859753386 1.85194970290473,
1 0,-1.12132034355965 5.12132034355963,4 3)) | 3.49440883690764
      1.33328930094119,3.12132034355964 0.878679656440359,
      | 2.66671069905881
      0.505591163092366,2.14805029
      0.228361402466141,
      | 1.58527096604839
      0.0576441587903094,1
      0,
      | 0.414729033951621
      0.0576441587903077,-0.1480502
      0.228361402466137,
      | -0.666710699058802
      0.505591163092361,-1.1213203
      0.878679656440353,
```



```

| -1.49440883690763 ↵
1.33328930094119,-1.77163859
1.85194970290472
| --ETC-- ↵
,3.94235584120969 ↵
3.58527096604839,4 ↵
3))

--3D-Beispiel
SELECT ST_AsText(ST_LineToCurve(geom)) As curved, ST_AsText(geom) AS not_curved
FROM (SELECT ST_Translate(ST_Force3D(ST_Boundary(ST_Buffer(ST_Point(1,3), 2,2))),0,0,3) AS
geom) AS foo;

curved | not_curved
-----+-----
CIRCULARSTRING Z (3 3 3,-1 2.999999999999999 3,3 3 3) | LINESTRING Z (3 3 3,2.4142135623731 ↵
1.58578643762691 3,1 1 3,
| -0.414213562373092 ↵
1.5857864376269 3,-1 ↵
2.999999999999999 3,
| -0.414213562373101 4.41421356237309 ↵
3,
| 0.9999999999999991 5 ↵
3,2.41421356237309 4.4142135623731 ↵
3,3 3 3)

(1 row)
```

Siehe auch

[ST_CurveToLine](#)

8.11.17 ST_MakeValid

ST_MakeValid — Versucht eine ungültige Geometrie, ohne den Verlust an Knoten zu bereinigen.

Synopsis

geometry **ST_MakeValid**(geometry input);

Beschreibung

Diese Funktion versucht aus einer invaliden Geometrie eine valide Darstellung zu erzeugen, ohne irgendeinen gegebenen Knoten zu verlieren. Geometrien, die bereits valide sind, werden ohne weiteren Eingriff zurückgegeben.

Unterstützte Eingaben sind: POINTS, MULTIPOINTS, LINESTRINGS, MULTILINESTRINGS, POLYGONS, MULTIPOLYGONS und GEOMETRYCOLLECTIONS aus einer Mischung der Vorigen.

Im Falle eines völligen oder teilweisen dimensional Kollapses kann die Ausgabegeometrie eine Sammelgeometrie von niedriger oder gleicher geometrischer Dimension oder eine Geometrie mit niedrigerer geometrischer Dimension sein.

Im Fall von Selbstüberschneidungen können Polygone zu Mehrfachgeometrien werden.

Verfügbarkeit: 2.0.0 benötigt GEOS-3.3.0

Erweiterung: 2.0.1, Geschwindigkeitsverbesserungen, benötigt GEOS-3.3.4

Erweiterung: 2.1.0 die Unterstützung von GeometryCollection und MultiPoint hinzugefügt.



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

Siehe auch

[ST_IsValid ST_CollectionExtract](#)

8.11.18 ST_MemUnion

ST_MemUnion — Das gleiche wie **ST_Union**, nur freundlicher zum Arbeitsspeicher (verwendet weniger Arbeitsspeicher und mehr Rechnerzeit).

Synopsis

```
geometry ST_MemUnion(geometry set geomfield);
```

Beschreibung

Hier fehlt eine brauchbare Beschreibung.

**Note**

Das gleiche wie **ST_Union**, nur freundlicher zum Arbeitsspeicher (verwendet weniger Arbeitsspeicher und mehr Rechnerzeit). Diese Aggregatfunktion vereingt eine Geometrie nach der anderen zu dem vorhergegangenen Ergebnis, während die Aggregatfunktion **ST_Union** zuerst ein Feld erzeugt und anschließend vereingt



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

Beispiele

Siehe [ST_Union](#)

Siehe auch

[ST_Union](#)

8.11.19 ST_MinimumBoundingCircle

ST_MinimumBoundingCircle — Gibt das kleinstmögliche Kreispolygon zurück, welches eine Geometrie zur Gänze beinhaltet. Standardmäßig werden 48 Segmente pro Viertelkreis verwendet.

Synopsis

```
geometry ST_MinimumBoundingCircle(geometry geomA, integer num_segs_per_qt_circ=48);
```

Beschreibung

Gibt das kleinstmögliche Kreispolygon zurück, welches eine Geometrie zur Gänze beinhaltet.



Note

Der Kreis wird standardmäßig durch ein Polygon mit 48 Segmenten pro Viertelkreis angenähert. Da das Polygon eine Annäherung an den minimalen Umgebungskreis ist, können einige Punkte der Eingabegeometrie nicht in dem Polygon enthalten sein. Die Annäherung kann durch Erhöhung der Anzahl der Segmente mit geringen Einbußen bei der Rechenleistung verbessert werden. Bei Anwendungen wo eine polygonale Annäherung nicht ausreicht, kann `ST_MinimumBoundingRadius` verwendet werden.

Wird üblicherweise auf Mehrfach/MULTI- und Sammelgeometrien/GeometryCollections angewandt. Obwohl es sich nicht um eine Aggregatfunktion handelt, können Sie es in Verbindung mit `ST_Collect` verwenden um den kleinstmöglichen Umgebungskreis eines Satzes an Geometrien zu erhalten. `ST_MinimumBoundingCircle(ST_Collect(somepointfield))`.

Das Verhältnis zwischen der Fläche des Polygons und der Fläche ihres kleinstmöglichen Umgebungskreises wird öfter als Roeck Test bezeichnet.

Verfügbarkeit: 1.4.0 - benötigt GEOS

Siehe auch

[ST_Collect](#), [ST_MinimumBoundingRadius](#)

Beispiele

```
SELECT d.disease_type,  
       ST_MinimumBoundingCircle(ST_Collect(d.the_geom)) As the_geom  
FROM disease_obs As d  
GROUP BY d.disease_type;
```



Minimaler Umgebungskreis eines Punktes und eines Linienzuges. Verwendet 8 Segmente um einen Viertelkreis anzunähern.

```

SELECT ST_AsText(ST_MinimumBoundingCircle(
    ST_Collect(
        ST_GeomFromEWKT('LINESTRING(55 75,125 150)'),
        ST_Point(20, 80)), 8
    )) As wktmbc;

wktmbc
-----
POLYGON((135.59714732062 115,134.384753327498 102.690357210921,130.79416296937 ↵
  90.8537670908995,124.963360620072 79.9451031602111,117.116420743937 ↵
  70.3835792560632,107.554896839789 62.5366393799277,96.6462329091006 ↵
  56.70583703063,84.8096427890789 53.115246672502,72.5000000000001 ↵
  51.9028526793802,60.1903572109213 53.1152466725019,48.3537670908996 ↵
  56.7058370306299,37.4451031602112 62.5366393799276,27.8835792560632 ↵
  70.383579256063,20.0366393799278 79.9451031602109,14.20583703063 ↵
  90.8537670908993,10.615246672502 102.690357210921,9.40285267938019 115,10.6152466725019 ↵
  127.309642789079,14.2058370306299 139.1462329091,20.0366393799275 ↵
  150.054896839789,27.883579256063 159.616420743937,
  37.4451031602108 167.463360620072,48.3537670908992 173.29416296937,60.190357210921 ↵
  176.884753327498,
  72.4999999999998 178.09714732062,84.8096427890786 176.884753327498,96.6462329091003 ↵
  173.29416296937,107.554896839789 167.463360620072,
  117.116420743937 159.616420743937,124.963360620072 150.054896839789,130.79416296937 ↵
  139.146232909101,134.384753327498 127.309642789079,135.59714732062 115))

```

Siehe auch

[ST_Collect](#), [ST_MinimumBoundingRadius](#)

8.11.20 ST_MinimumBoundingRadius

ST_MinimumBoundingRadius — Gibt den Mittelpunkt und den Radius des kleinstmöglichen Kreises zurück, der die gesamte Geometrie beinhaltet.

Synopsis

(geometry, double precision) **ST_MinimumBoundingRadius**(geometry geom);

Beschreibung

Gibt einen Datensatz zurück, der aus dem Mittelpunkt und dem Radius des kleinstmöglichen Kreises, der die gesamte Geometrie beinhaltet, besteht.

Kann in Verbindung mit [ST_Collect](#) verwendet werden, um den minimalen Umgebungskreis einer Menge von Geometrien zu erhalten.

Verfügbarkeit: 2.3.0

Siehe auch

[ST_Collect](#), [ST_MinimumBoundingCircle](#)

Beispiele

```
SELECT ST_AsText(center), radius FROM ST_MinimumBoundingRadius('POLYGON((26426 65078,26531 65242,26075 65136,26096 65427,26426 65078))');
```

st_astext	radius
POINT(26284.8418027133 65267.1145090825)	247.436045591407

8.11.21 ST_Polygonize

ST_Polygonize — Aggregatfunktion. Erzeugt eine Sammelgeometrie/GeometryCollection, welche Polygone enthält, die aus den einzelnen Linien einer Menge von Geometrien gebildet werden können.

Synopsis

```
geometry ST_Polygonize(geometry set geomfield);
geometry ST_Polygonize(geometry[] geom_array);
```

Beschreibung

Erzeugt eine Sammelgeometrie/GeometryCollection, welche Polygone enthält, die aus den einzelnen Linien einer Menge von Geometrien gebildet werden können.



Note

Tools von Drittanbietern haben öfters Probleme mit einer Sammelgeometrie. Verwenden Sie daher **ST_Polygonize** in Verbindung mit **ST_Dump**, um die Polygone in Einzelpolygone zu zerlegen.



Note

Damit diese Funktion korrekt arbeitet, müssen die Knoten des eingegebenen Liniennetzes richtig angeordnet sein

Verfügbarkeit: 1.0 - benötigt GEOS >= 2.1.0.

Beispiele: Einzelne Linienzüge in ein Polygon umwandeln.

```
SELECT ST_AsEWKT(ST_Polygonize(the_geom_4269)) As geomtextrep
FROM (SELECT the_geom_4269 FROM ma.suffolk_edges ORDER BY tlid LIMIT 45) As foo;
```

```
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)
```

```
--Verwendung von ST_Dump um die polygonisierten Geometrien in einzelne Polygone auszuladen
SELECT ST_AsEWKT((ST_Dump(foofoo.polycoll)).geom) As geomtextrep
FROM (SELECT ST_Polygonize(the_geom_4269) As polycoll
```

```
FROM (SELECT the_geom_4269 FROM ma.suffolk_edges
      ORDER BY tlid LIMIT 45) As foo) As foofoo;
```

```
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_Node](#), [ST_Dump](#)

8.11.22 ST_Node

ST_Node — Knotenberechnung für eine Menge von Linienzügen.

Synopsis

geometry **ST_Node**(geometry geom);

Beschreibung

Komplette Knotenberechnung für eine Menge von Linienzügen unter Verwendung der kleinstmöglichen Knotenanzahl und Erhaltung aller bestehenden Knoten.



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

Verfügbarkeit: 2.0.0 - benötigt GEOS >= 3.3.0.



Note

Aufgrund eines Fehlers in GEOS < 3.3.1 versagt diese Funktion bei der Knotenberechnung von selbstüberschneidenden Linien. Dies wurde mit GEOS 3.3.2 und höher behoben.



Note

Änderung: Ab 2.4.0 verwendet diese Funktion intern GEOSNode anstatt GEOSUnaryUnion. Dies kann bedingen, dass die resultierenden Linienzüge eine andere Reihenfolge und Ausrichtung haben als bei PostGIS < 2.4.

Beispiele

```
SELECT ST_AsText (
      ST_Node('LINESTRINGZ(0 0 0, 10 10 10, 0 10 5, 10 0 3)::geometry')
) As output;
output
-----
MULTILINESTRING Z ((0 0 0,5 5 4.5),(5 5 4.5,10 10 10,0 10 5,5 5 4.5),(5 5 4.5,10 0 3))
```

Siehe auch[ST_UnaryUnion](#)**8.11.23 ST_OffsetCurve**

ST_OffsetCurve — Gibt eine Linie zurück, die um eine gegebenen Entfernung und Seite von der Eingabelinie versetzt ist. Nützlich zur Berechnung von Linien, die zu einer Mittellinie parallel verlaufen

Synopsis

geometry **ST_OffsetCurve**(geometry line, float signed_distance, text style_parameters=’’);

Beschreibung

Gibt eine versetzte Linie zurück; in der gegebenen Entfernung und auf der Seite der Eingabelinie, die angegebenen wurden. Alle Punkte der Ausgabegeometrie liegen nicht weiter weg, als die zur Eingangsgeometrie angegebene Entfernung.

Bei positiven Entfernungsangaben/distance findet der Versatz auf der linken Seite der Eingabelinie statt und die Richtung wird beibehalten. Bei negativen Entfernungsangaben auf der rechten Seite und in entgegengesetzter Richtung.

Verfügbarkeit: 2.0 - benötigt GEOS >= 3.2, verbessert mit GEOS >= 3.3

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.

Die Einheiten der Entfernung werden in den Einheiten des Koordinatenreferenzsystems gemessen.

Als Eingabe gelten nur Linienzüge.

Wird vom GEOS Modul ausgeführt

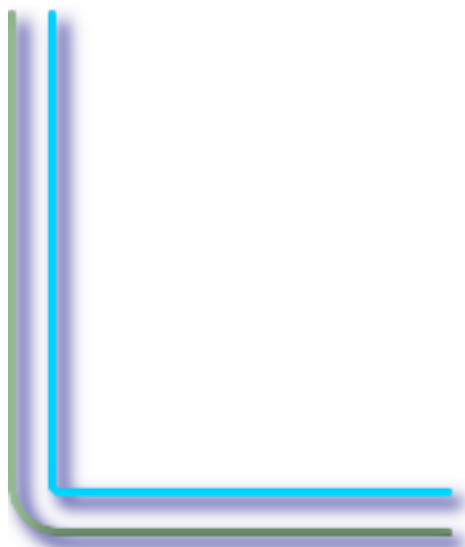
**Note**

Diese Funktion ignoriert die dritte Dimension (z) und gibt immer ein 2-D Ergebnis zurück, sogar dann wenn eine 3D-Geometrie überreicht wird.

Beispiele

Einen offenen Puffer um die Straßen rechnen

```
SELECT ST_Union(
  ST_OffsetCurve(f.the_geom, f.width/2, 'quad_segs=4 join=round'),
  ST_OffsetCurve(f.the_geom, -f.width/2, 'quad_segs=4 join=round')
) as track
FROM someroadstable;
```



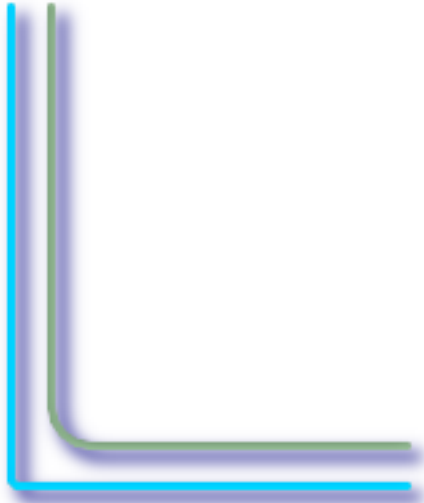
15, 'quad_segs=4 join=round' Ausgangslinie und die um 15 Einheiten versetzte Parallele.

```
SELECT ST_AsText(ST_OffsetCurve(↵
    ST_GeomFromText(↵
'LINESTRING(164 16,144 16,124 16,104 ↵
    16,84 16,64 16,↵
        44 16,24 16,20 16,18 16,17 17,↵
        16 18,16 20,16 40,16 60,16 80,16 ↵
    100,↵
        16 120,16 140,16 160,16 180,16 ↵
    195)'),↵
    15, 'quad_segs=4 join=round'));↵
--output --↵
LINESTRING(164 1,18 1,12.2597485145237 ↵
    2.1418070123307,↵
    7.39339828220179 ↵
    5.39339828220179,↵
    5.39339828220179 ↵
    7.39339828220179,↵
    2.14180701233067 ↵
    12.2597485145237,1 18,1 195)
```



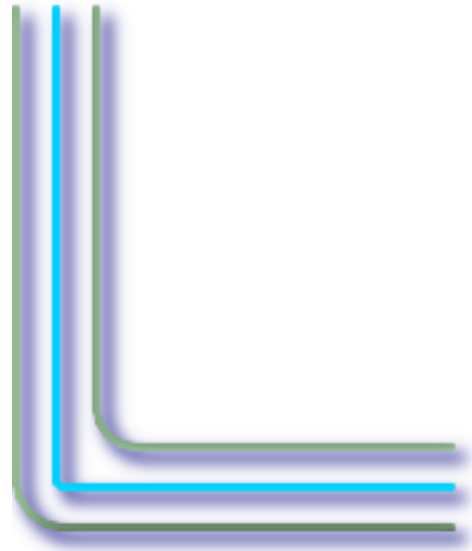
-15, 'quad_segs=4 join=round' Ausgangslinie und die um -15 Einheiten versetzte Parallele.

```
SELECT ST_AsText(ST_OffsetCurve(geom,↵
    -15, 'quad_segs=4 join=round')) ↵
As notsocurvy↵
FROM ST_GeomFromText(↵
'LINESTRING(164 16,144 16,124 16,104 ↵
    16,84 16,64 16,↵
        44 16,24 16,20 16,18 16,17 17,↵
        16 18,16 20,16 40,16 60,16 80,16 ↵
    100,↵
        16 120,16 140,16 160,16 180,16 ↵
    195)') As geom;↵
-- notsocurvy --↵
LINESTRING(31 195,31 31,164 31)
```

doppelter Versatz um es kurviger zu bekommen; beachte die Richtungsänderung, also $-30 + 15 = -15$

```
SELECT ST_AsText(ST_OffsetCurve(↵
    ST_OffsetCurve(geom,
        -30, 'quad_segs=4 join=round'), ↵
    -15, 'quad_segs=4 join=round')) As morecurvy
FROM ST_GeomFromText(
'LINESTRING(164 16,144 16,124 16,104 ↵
    16,84 16,64 16,
    44 16,24 16,20 16,18 16,17 17,
    16 18,16 20,16 40,16 60,16 80,16 ↵
    100,
    16 120,16 140,16 160,16 180,16 ↵
    195)') As geom;
-- morecurvy --
LINESTRING(164 31,46 31,40.2597485145236 ↵
    32.1418070123307,
    35.3933982822018 35.3933982822018,
    32.1418070123307 40.2597485145237,31 ↵
    46,31 195)
```



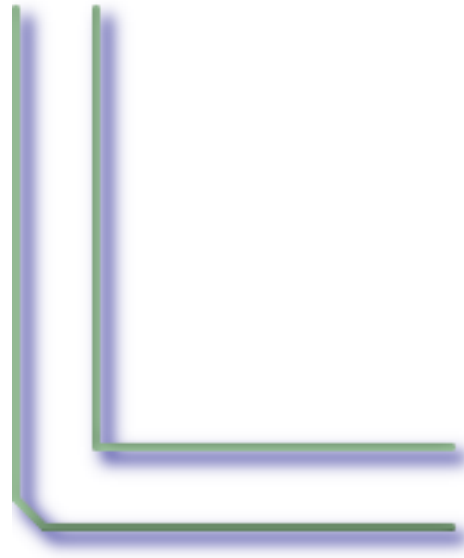
Doppelter Versatz um es kurviger zu bekommen, kombiniert mit einem normalen Versatz von 15 um parallele Linien zu erhalten. Überlagert mit dem Original.

```
SELECT ST_AsText(ST_Collect(
    ST_OffsetCurve(geom, 15, '↵
    quad_segs=4 join=round'),
    ST_OffsetCurve(ST_OffsetCurve(↵
    geom,
        -30, 'quad_segs=4 join=round'), ↵
    -15, 'quad_segs=4 join=round')
    ) As parallel_curves
FROM ST_GeomFromText(
'LINESTRING(164 16,144 16,124 16,104 ↵
    16,84 16,64 16,
    44 16,24 16,20 16,18 16,17 17,
    16 18,16 20,16 40,16 60,16 80,16 ↵
    100,
    16 120,16 140,16 160,16 180,16 ↵
    195)') As geom;
-- parallel curves --
MULTILINESTRING((164 1,18 ↵
    1,12.2597485145237 2.1418070123307,
    7.39339828220179 ↵
    5.39339828220179,5.39339828220179 7.39339828220179,
    2.14180701233067 12.2597485145237,1 18,1 ↵
    195),
(164 31,46 31,40.2597485145236 ↵
    32.1418070123307,35.3933982822018 35.3933982822018,
    32.1418070123307 40.2597485145237,31 ↵
    46,31 195))
```



15, 'quad_segs=4 join=bevel' gemeinsam mit der Ausgangslinie

```
SELECT ST_AsText(ST_OffsetCurve(↵
    ST_GeomFromText(
'LINESTRING(164 16,144 16,124 16,104 ↵
    16,84 16,64 16,
        44 16,24 16,20 16,18 16,17 17,
        16 18,16 20,16 40,16 60,16 80,16 ↵
    100,
        16 120,16 140,16 160,16 180,16 ↵
    195)'),
        15, 'quad_segs=4 join=↵
    bevel'));
-- output --
LINESTRING(164 1,18 1,7.39339828220179 ↵
    5.39339828220179,
        5.39339828220179 ↵
    7.39339828220179,1 18,1 195)
```



15,-15 collected, join=mitre mitre_limit=2.1

```
SELECT ST_AsText(ST_Collect(
    ST_OffsetCurve(geom, 15, '↵
    quad_segs=4 join=mitre mitre_limit=2.2'),
    ST_OffsetCurve(geom, -15, '↵
    quad_segs=4 join=mitre mitre_limit=2.2')
) )
FROM ST_GeomFromText(
'LINESTRING(164 16,144 16,124 16,104 ↵
    16,84 16,64 16,
        44 16,24 16,20 16,18 16,17 17,
        16 18,16 20,16 40,16 60,16 80,16 ↵
    100,
        16 120,16 140,16 160,16 180,16 ↵
    195)') As geom;
-- output --
MULTILINESTRING((164 1,11.7867965644036 ↵
    1,1 11.7867965644036,1 195),
    (31 195,31 31,164 31))
```

Siehe auch

[ST_Buffer](#)

8.11.24 ST_RemoveRepeatedPoints

ST_RemoveRepeatedPoints — Gibt eine Version der Eingabegeometrie zurück, wobei duplizierte Punkte entfernt werden.

Synopsis

geometry **ST_RemoveRepeatedPoints**(geometry geom, float8 tolerance);

Beschreibung

Gibt eine Version der gegebenen Geometrie zurück, wobei duplizierte Punkte gelöscht werden. Tut zurzeit nur mit (Multi)Lines, (Multi)Polygons und MultiPoints 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. Wenn der Parameter "tolerance" angegeben ist, werden Knoten, die einen geringeren Abstand untereinander haben zwecks Löschen als ident betrachtet.

Verfügbarkeit: 2.2.0



This function supports Polyhedral surfaces.



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

Siehe auch

[ST_Simplify](#)

8.11.25 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.

Verfügbarkeit: 2.0.0 benötigt GEOS >= 3.3.0.

Beispiele: Gemeinsame Strecken finden



Ein MultiLinestring und ein LineString



Die gemeinsame Strecke eines MultiLinestring und Linestring mit überlagerter Ursprungsgeometrie.

```
SELECT ST_AsText (
  ST_SharedPaths (
    ST_GeomFromText ('MULTILINESTRING((26 125,26 200,126 200,126 125,26 125),
      (51 150,101 150,76 175,51 150))'),
    ST_GeomFromText ('LINESTRING(151 100,126 156.25,126 125,90 161, 76 175)')
  ) As wkt
```

wkt

```
-----
GEOMETRYCOLLECTION(MULTILINESTRING((126 156.25,126 125),
(101 150,90 161),(90 161,76 175)),MULTILINESTRING EMPTY)
```

-- Das selbe Beispiel, aber die Richtung des Linienzuges ist umgedreht/flipped

```
SELECT ST_AsText (
  ST_SharedPaths (
    ST_GeomFromText ('LINESTRING(76 175,90 161,126 125,126 156.25,151 100)'),
    ST_GeomFromText ('MULTILINESTRING((26 125,26 200,126 200,126 125,26 125),
      (51 150,101 150,76 175,51 150))')
  ) As wkt
```

wkt

```
-----
GEOMETRYCOLLECTION(MULTILINESTRING EMPTY,
MULTILINESTRING((76 175,90 161),(90 161,101 150),(126 125,126 156.25)))
```

Siehe auch

[ST_Dump](#), [ST_GeometryN](#), [ST_NumGeometries](#)

8.11.26 ST_ShiftLongitude

ST_ShiftLongitude — Schaltet geometrische Koordinaten zwischen den Bereichen -180..180 und 0..360 um.

Synopsis

geometry **ST_ShiftLongitude**(geometry geomA);

Beschreibung

Liest jeden Punkt/Knoten jeder Featurekomponente einer Geometrie, und wenn die Längenkoordinate <0 ist wird 360 hinzugezählt. Das Ergebnis ist eine 0-360 Grad Version der Daten, die auf einer bei 180 Grad zentrierten Karte dargestellt werden kann.



Note

Dies ist nur sinnvoll bei Längen- und Breitenangabe, z.B.: 4326 (WGS 84 long lat)



Pre-1.3.4 Aufgrund eines Bugs funktionierte dies für MULTIPOINT nicht. Mit 1.3.4+ funktioniert es auch mit MULTIPOINT.



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

Erweiterung: Mit 2.0.0 wurde die Unterstützung für polyedrische Oberflächen und TIN eingeführt.

Anmerkung: Vor 2.2.0 hieß diese Funktion "ST_Shift_Longitude"



This function supports Polyhedral surfaces.



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

Beispiele

```
--3D Punkte
SELECT ST_AsEWKT(ST_ShiftLongitude(ST_GeomFromEWKT('SRID=4326;POINT(-118.58 38.38 10)')) ) ↔
      As geomA,
      ST_AsEWKT(ST_ShiftLongitude(ST_GeomFromEWKT('SRID=4326;POINT(241.42 38.38 10)')) ) ↔
      As geomB
geomA                                     geomB
-----
SRID=4326;POINT(241.42 38.38 10) SRID=4326;POINT(-118.58 38.38 10)

--normaler Linienzug
SELECT ST_AsText(ST_ShiftLongitude(ST_GeomFromText('LINESTRING(-118.58 38.38, -118.20 38.45)')) ) ↔
      st_astext
-----
LINESTRING(241.42 38.38,241.8 38.45)
```

Siehe auch

[ST_WrapX](#)

8.11.27 ST_WrapX

ST_WrapX — Versammelt eine Geometrie um einen X-Wert

Synopsis

geometry **ST_WrapX**(geometry geom, float8 wrap, float8 move);

Beschreibung

Diese Funktion trennt die Ausgangsgeometrie auf und verschiebt jeden resultierenden Bestandteil der auf die rechte (bei negativem 'move') oder auf die linke Seite (bei positivem 'move') der gegebenen 'wrap'-Linie fällt in die Richtung die durch den 'move' Parameter angegeben ist. Schließlich werden die Teile wieder vereinigt.



Note

Nützlich, um eine Eingabe in Länge und Breite neu zu zentrieren, damit die wesentlichen Geoobjekte nicht von einer Seite bis zur anderen abgebildet werden.

Verfügbarkeit: 2.3.0



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

Beispiele

```
-- Verschiebt alle Komponenten einer gegebenen Geometrie, deren Umgebungsrechteck
-- zur Gänze links von x=0 liegen um +360
select ST_WrapX(the_geom, 0, 360);

-- Verschiebt alle Komponenten einer gegebenen Geometrie, deren Umgebungsrechteck
-- zur Gänze links von x=30 liegen um +360
select ST_WrapX(the_geom, -30, 360);
```

Siehe auch

[ST_ShiftLongitude](#)

8.11.28 ST_Simplify

ST_Simplify — Gibt eine vereinfachte Version der Ausgangsgeometrie zurück. Verwendet den Douglas-Peucker Algorithmus.

Synopsis

geometry **ST_Simplify**(geometry geomA, float tolerance, boolean preserveCollapsed);

Beschreibung

Gibt eine "vereinfachte" Version der gegebenen Geometrie zurück. Verwendet den Visvalingam-Whyatt Algorithmus. Tut zurzeit nur mit (Multi)Lines und (Multi)Polygons etwas, kann aber gefahrlos mit jedem geometrischen Datentyp verwendet werden. Da die Vereinfachung auf einer Objekt zu Objekt Basis passiert, können Sie diese Funktion auch mit einer Sammelgeometrie speisen.



Note

Bemerke, dass die zurückgegebene Geometrie ihre Simplität verlieren kann (siehe [ST_IsSimple](#)).



Note

Beachten Sie bitte, dass die Topologie möglicherweise nicht erhalten bleibt und ungültige Geometrien entstehen können. Benutzen Sie bitte (see [ST_SimplifyPreserveTopology](#)) um die Topologie zu erhalten.



Note

Diese Funktion kann mit 3D umgehen und die dritte Dimension beeinflusst auch das Ergebnis.

Verfügbarkeit: 2.2.0

Beispiele

Ein Linienzug vereinfacht mit einem Schwellenwert von 30 für die minimale Fläche.

```
select ST_AsText(ST_SimplifyVW(geom,30)) simplified
FROM (SELECT 'LINESTRING(5 2, 3 8, 6 20, 7 25, 10 10) '::geometry geom) As foo;
-result
simplified
-----+-----+
LINESTRING(5 2,7 25,10 10)
```

Siehe auch

[ST_SetEffectiveArea](#), [ST_Simplify](#), [ST_SimplifyPreserveTopology](#), [Topology](#) [ST_Simplify](#)

8.11.31 ST_SetEffectiveArea

ST_SetEffectiveArea — Setzt die Nutzfläche für jeden Knoten und speichert den Wert in der M-Ordinate. Eine vereinfachte Geometrie kann dann über einen Filter auf die M-Ordinate erzeugt werden.

Synopsis

geometry **ST_SetEffectiveArea**(geometry geomA, float threshold = 0, integer set_area = 1);

Beschreibung

Setzt die Nutzfläche für jeden Knoten. Verwendet den Visvalingam-Whyatt Algorithmus. Die Nutzfläche wird als M-Wert des Knoten gespeichert. Wird der optionale Parameter "threshold" verwendet, so wird eine vereinfachte Geometrie zurückgegeben, die nur jene Knoten enthält, deren Nutzfläche größer oder gleich dem Schwellenwert ist.

Diese Funktion kann für die serverseitige Vereinfachung, mittels eines Schwellenwerts verwendet werden. Eine andere Möglichkeit besteht darin, einen Schwellenwert von null anzugeben. In diesem Fall wird die gesamte Geometrie inklusive der Nutzflächen als M-Werte zurückgegeben, welche dann am Client für eine rasche Vereinfachung genutzt werden können.

Tut zurzeit nur mit (Multi)Lines und (Multi)Polygons etwas, kann aber gefahrlos mit jedem geometrischen Datentyp verwendet werden. Da die Vereinfachung auf einer Objekt zu Objekt Basis passiert, können Sie diese Funktion auch mit einer Sammelgeometrie speisen.



Note

Bemerke, dass die zurückgegebene Geometrie ihre Simplizität verlieren kann (siehe [ST_IsSimple](#)).



Note

Beachten Sie bitte, dass die Topologie möglicherweise nicht erhalten bleibt und ungültige Geometrien entstehen können. Benutzen Sie bitte (see [ST_SimplifyPreserveTopology](#)) um die Topologie zu erhalten.



Note

Die Ausgabegeometrie verliert die gesamte vorhandene Information über die M-Werte



Note

Diese Funktion kann mit 3D umgehen und die dritte Dimension beeinflusst auch die tatsächliche Fläche

Verfügbarkeit: 2.2.0

Beispiele

Berechnung der Nutzfläche eines Linienzugs. Da wir einen Schwellenwert von null verwenden, werden alle Knoten der Eingabegeometrie zurückgegeben.

```
select ST_AsText(ST_SetEffectiveArea(geom)) all_pts, ST_AsText(ST_SetEffectiveArea(geom,30) ↔
) thrshld_30
FROM (SELECT 'LINESTRING(5 2, 3 8, 6 20, 7 25, 10 10)::geometry geom) As foo;
-result
all_pts | thrshld_30
-----+-----
LINESTRING M (5 2 3.40282346638529e+38,3 8 29,6 20 1.5,7 25 49.5,10 10 3.40282346638529e ↔
+38) | LINESTRING M (5 2 3.40282346638529e+38,7 25 49.5,10 10 3.40282346638529e+38)
```

Siehe auch

[ST_SimplifyVW](#)

8.11.32 ST_Split

ST_Split — Gibt eine Sammelgeometrie zurück, die beim Auftrennen einer Geometrie entsteht.

Synopsis

geometry **ST_Split**(geometry input, geometry blade);

Beschreibung

Diese Funktion unterstützt das Auftrennen einer Linie durch einen MultiPoint, eine MultiLine oder eine (Multi)Polygongrenze und das Auftrennen eines (Multi)Polygons durch eine Linie. Die zurückgegebene Geometrie ist immer eine Sammelgeometrie.

Diese Funktion ist das Gegenstück zu ST_Union. Theoretisch sollte die Anwendung von ST_Union auf die Elemente der zurückgegebenen Sammelgeometrie immer zur ursprünglichen Geometrie führen.

Verfügbarkeit: 2.0.0

Änderung: 2.2.0 Unterstützung für das Auftrennen von Linien durch eine MultiLine, einen MultiPoint oder eine (Multi)Polygongrenze eingeführt



Note

Um die Robustheit von ST_Split zu erhöhen, kann es zweckmäßig sein ein **ST_Snap** mit einem sehr niedrigen Toleranzwert auszuführen, bevor die Eingabe an die "Schneide" erfolgt. Andernfalls kann das intern verwendete Koordinatengitter Toleranzprobleme verursachen, wodurch die Koordinaten der Eingabe und der "Schneide" nicht zusammenfallen und die Eingabegeometrie nicht korrekt aufgetrennt wird (siehe [#2192](#)).

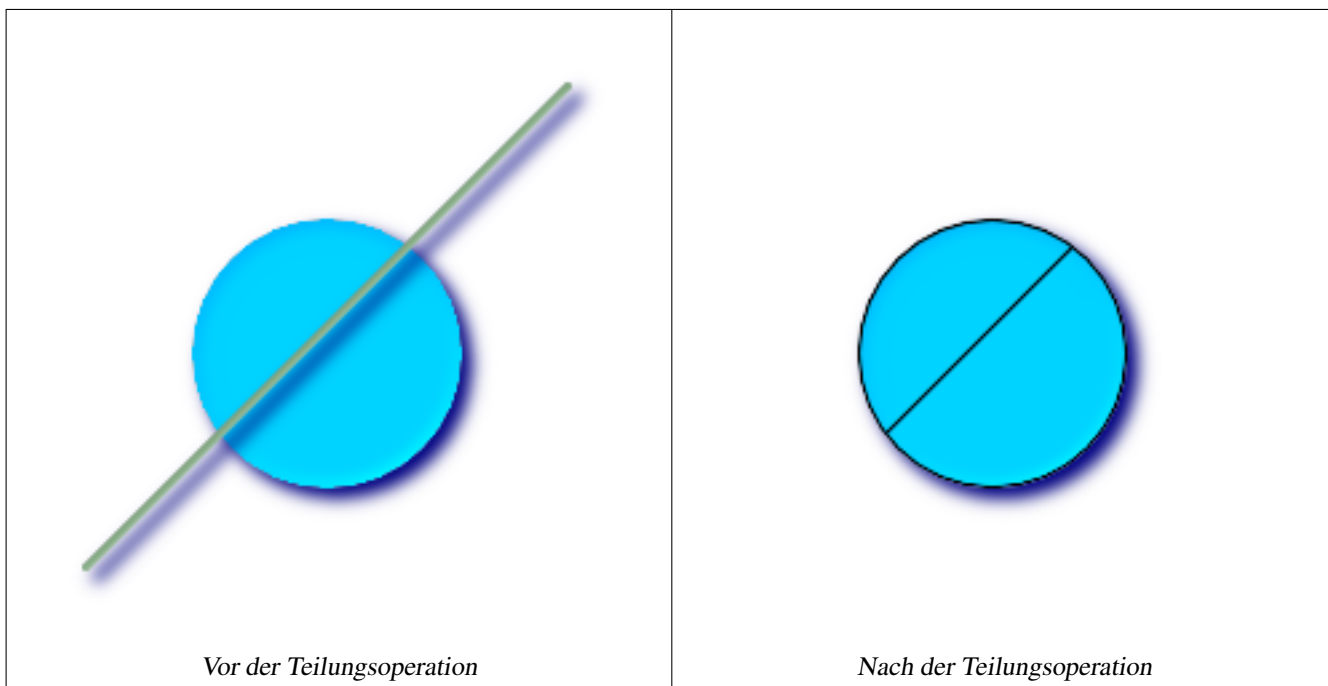


Note

Wenn ein Mehrfachpolygon als "Schneide" übergeben wird, so wird deren linearer Bestandteil (die Begrenzung) verwendet, um die Eingabegeometrie aufzutrennen.

Beispiele

Polygon aufgetrennt durch eine Linie



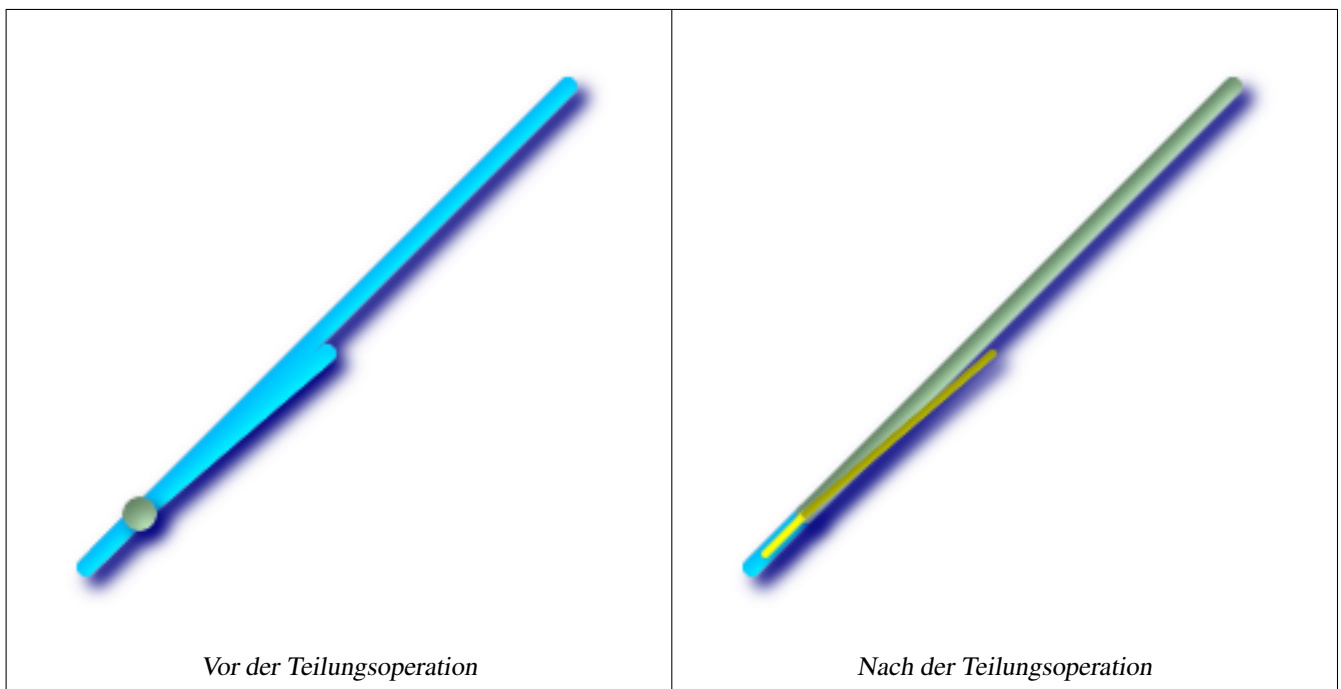
```
-- erzeugt eine Sammelgeometrie die aus 2 Hälften des Polygons besteht
-- ähnlich dem Beispiel unter ST_BuildArea
SELECT ST_Split(circle, line)
FROM (SELECT
  ST_MakeLine(ST_MakePoint(10, 10),ST_MakePoint(190, 190)) As line,
  ST_Buffer(ST_GeomFromText('POINT(100 90)'), 50) As circle) As foo;

-- result --
GEOMETRYCOLLECTION(POLYGON((150 90,149.039264020162 80.2454838991936,146.193976625564 ↵
  70.8658283817455,..), POLYGON(..)))

-- Um in einzelne Polygone umzurechnen können Sie ST_Dump oder ST_GeometryN verwenden
SELECT ST_AsText((ST_Dump(ST_Split(circle, line))).geom) As wkt
FROM (SELECT
  ST_MakeLine(ST_MakePoint(10, 10),ST_MakePoint(190, 190)) As line,
  ST_Buffer(ST_GeomFromText('POINT(100 90)'), 50) As circle) As foo;

-- result --
wkt
-----
POLYGON((150 90,149.039264020162 80.2454838991936,..))
POLYGON((60.1371179574584 60.1371179574584,58.4265193848728 ↵
  62.2214883490198,53.8060233744357 ..))
```

Ein MultiLinestring durch einen Punkt aufgetrennt



```
SELECT ST_AsText(ST_Split(mline, pt)) As wktcut
FROM (SELECT
  ST_GeomFromText('MULTILINESTRING((10 10, 190 190), (15 15, 30 30, 100 90))') As mline,
  ST_Point(30,30) As pt) As foo;
```

wktcut

```
GEOMETRYCOLLECTION(
  LINESTRING(10 10,30 30),
  LINESTRING(30 30,190 190),
  LINESTRING(15 15,30 30),
  LINESTRING(30 30,100 90)
)
```

Siehe auch

[ST_AsText](#), [ST_BuildArea](#), [ST_Dump](#), [ST_GeometryN](#), [ST_Union](#), [ST_Subdivide](#)

8.11.33 ST_SymDifference

ST_SymDifference — Gibt eine Geometrie zurück, die jene Teile von A und B repräsentiert, die sich nicht überlagern. Wird symmetrische Differenz genannt, da $\text{ST\_SymDifference}(A,B) = \text{ST\_SymDifference}(B,A)$.

Synopsis

geometry **ST_SymDifference**(geometry geomA, geometry geomB);

Beschreibung

Gibt eine Geometrie zurück, die jene Teile von A und B repräsentiert, die sich nicht überlagern. Wird symmetrische Differenz genannt, da $\text{ST\_SymDifference}(A,B) = \text{ST\_SymDifference}(B,A)$. Man kann sich das auch als $\text{ST\_Union}(\text{geomA}, \text{geomB}) - \text{ST\_Intersection}(A,B)$ vorstellen.

Wird vom GEOS Modul ausgeführt



Note

Nicht mit einer GeometryCollection als Übergabewert aufrufen



This method implements the **OpenGIS Simple Features Implementation Specification for SQL 1.1**. s2.1.1.3

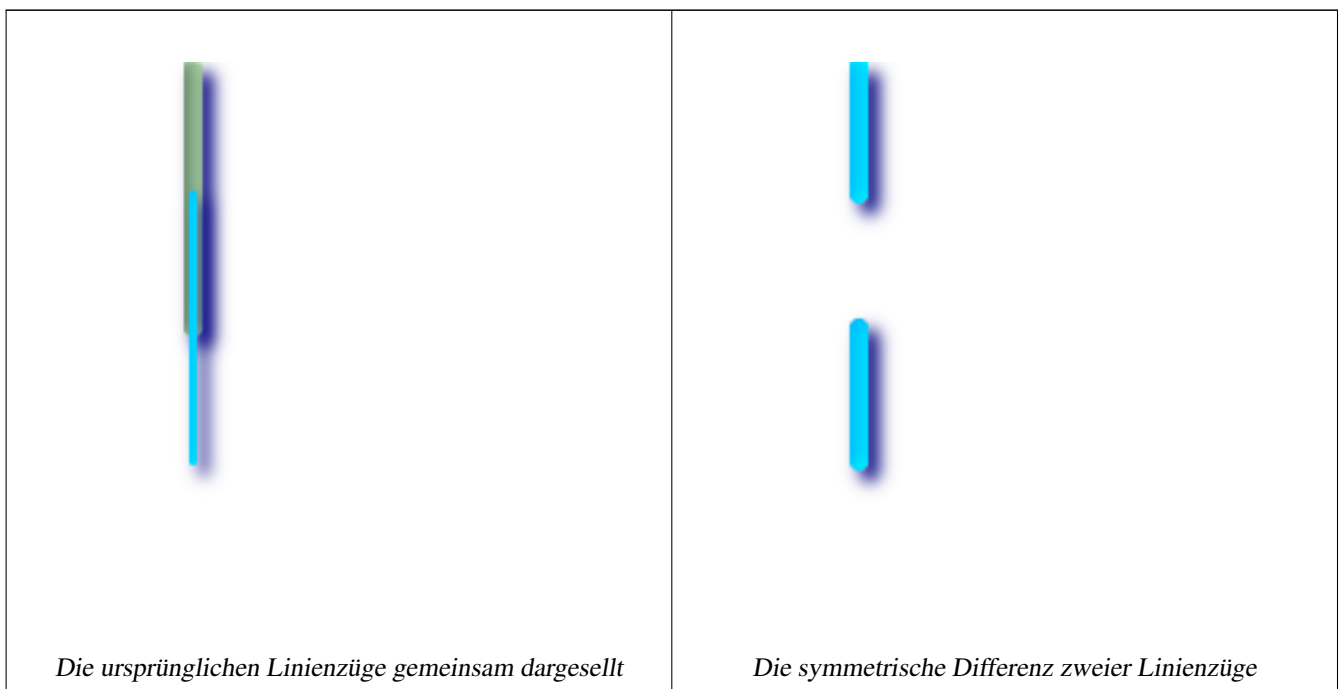


This method implements the SQL/MM specification. SQL-MM 3: 5.1.21



This function supports 3d and will not drop the z-index. Allerdings scheint nur x y bei der Berechnung der Differenz herangezogen und anschließend an den Z-Index zurückgeheftet zu werden

Beispiele



```
--Sicher für 2D - symmetrische Differenz von 2 Linienzügen
SELECT ST_AsText(
    ST_SymDifference(
        ST_GeomFromText('LINESTRING(50 100, 50 200)'),
        ST_GeomFromText('LINESTRING(50 50, 50 150)')
    )
);
```

```
st_astext
-----
MULTILINESTRING((50 150,50 200),(50 50,50 100))
```

```
--Mit 3D passiert nicht ganz das richtige
SELECT ST_AsEWKT(ST_SymDifference(ST_GeomFromEWKT('LINESTRING(1 2 1, 1 4 2)'),
    ST_GeomFromEWKT('LINESTRING(1 1 3, 1 3 4)')))
```

```
st_astext
-----
MULTILINESTRING((1 3 2.75,1 4 2),(1 1 3,1 2 2.25))
```

Siehe auch

[ST_Difference](#), [ST_Intersection](#), [ST_Union](#)

8.11.34 ST_Subdivide

ST_Subdivide — Gibt eine Geometriemenge zurück, wobei keine Geometrie der Menge mehr als die festgelegte Anzahl an Knoten aufweist.

Synopsis

```
setof geometry ST_Subdivide(geometry geom, integer max_vertices=256);
```

Beschreibung

Wandelt eine Einzelgeometrie in eine Menge um, bei der jedes Element weniger als die maximal erlaubte Anzahl an Knoten hat. Nützlich für die Konvertierung von übermäßig großen Polygonen und anderen Geoobjekten in kleine Teile, die dann in die Speicherseite der Datenbank passen. Verwendet dieselbe Methode beim Zerschneiden wie **ST_ClipByBox2D**, indem die Eingabegeometrie solange rekursiv zerteilt wird bis alle Teile weniger als die maximal erlaubte Knotenanzahl aufweisen. Die kleinste erlaubte Knotenanzahl ist 8; wenn Sie weniger als 8 angeben wird eine Fehlermeldung angezeigt.

Das Auftrennen wird durch das GEOS Modul ausgeführt.



Note

Benötigt GEOS 3.5.0+

Verfügbarkeit: 2.2.0 benötigt GEOS >= 3.5.0.

Beispiele

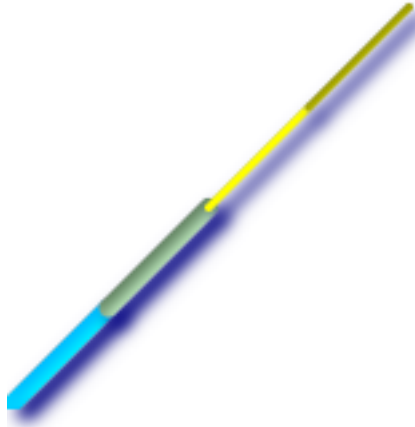
```
-- Erzeugt eine neu gegliederte Tabelle, passend um es mit dem Original zu verknüpfen
CREATE TABLE subdivided_geoms AS
SELECT pkey, ST_Subdivide(geom) AS geom
FROM original_geoms;
```



In maximal 10 Knoten unterteilen

```
SELECT row_number() OVER() As rn, ST_AsText(geom) As wkt
FROM ( SELECT ST_SubDivide('POLYGON((132 10,119 23,85 35,68 29,66 28,49 42,32 56,22 64,32 110,40 119,36 150,
57 158,75 171,92 182,114 184,132 186,146 178,176 184,179 162,184 141,190 122,
190 100,185 79,186 56,186 52,178 34,168 18,147 13,132 10))'::geometry,10)) As f(geom);
```

rn	wkt
1	POLYGON((22 64,29.3913043478263 98.0000000000001,106.000000000001 98.000000000001,106.000000000001 27.5882352941173,85 35,68 29,66 28,49 42,32 56,22 64))
2	POLYGON((29.3913043478263 98.0000000000001,32 110,40 119,36 150,57 158,75 11,92 182,106.0000000000001 183.2727272727,106.000000000001 98.000000000001,29.913043478263 98.000000000001))
3	POLYGON((106.000000000001 27.5882352941173,106.000000000001 98.000000000000,189.52380952381 98.000000000001,185 79,186 56,186 52,178 34,168 18,147 13,132 0,119 23,106.000000000001 27.5882352941173))
4	POLYGON((106.000000000001 98.000000000001,106.000000000001 183.2727272727,114 184,132 186,146 178,176 184,179 162,184 141,190 122,190 100,189.5238095238 98.000000000001,106.000000000001 98.000000000001))



Nützlich in Verbindung mit ST_Segmentize um zusätzliche Knoten zu erzeugen, die dann zum Auftrennen verwendet werden können

```
SELECT ST_AsText(ST_SubDivide(ST_Segmentize('LINESTRING(0 0, 100 100, 150 150)'):: geometry, 10), 8);
```

```
LINESTRING(0 0,7.07106781186547 7.07106781186547,14.1421356237309 ↵
14.1421356237309,21.2132034355964 21.2132034355964,28.2842712474619 ↵
28.2842712474619,35.3553390593274 35.3553390593274,37.4999999999998 37.4999999999998)
LINESTRING(37.4999999999998 37.4999999999998,42.4264068711929 ↵
42.4264068711929,49.4974746830583 49.4974746830583,56.5685424949238 ↵
56.5685424949238,63.6396103067893 63.6396103067893,70.7106781186548 ↵
70.7106781186548,74.9999999999998 74.9999999999998)
LINESTRING(74.9999999999998 74.9999999999998,77.7817459305202 ↵
77.7817459305202,84.8528137423857 84.8528137423857,91.9238815542512 ↵
91.9238815542512,98.9949493661167 98.9949493661167,100 100,107.071067811865 ↵
107.071067811865,112.4999999999998 112.4999999999998)
LINESTRING(112.4999999999998 112.4999999999998,114.142135623731 ↵
114.142135623731,121.213203435596 121.213203435596,128.284271247462 ↵
128.284271247462,135.355339059327 135.355339059327,142.426406871193 ↵
142.426406871193,149.497474683058 149.497474683058,149.9999999999998 149.9999999999998)
```

Siehe auch

[ST_AsText](#), [ST_ClipByBox2D](#), [ST_Segmentize](#), [ST_Split](#)

8.11.35 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



This method supports Circular Strings and Curves



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



This function supports M coordinates.



This function supports Polyhedral surfaces.



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

Beispiel

```
-- Scale M value by 2
SELECT ST_AsText(
  ST_SwapOrdinates(
    ST_Scale(
      ST_SwapOrdinates(g, 'xm'),
      2, 1
    ),
    'xm'
  )
) FROM ( SELECT 'POINT ZM (0 0 0 2)::geometry g ) foo;
      st_astext
-----
POINT ZM (0 0 0 4)
```

Siehe auch

[ST_FlipCoordinates](#)

8.11.36 ST_Union

ST_Union — Gibt eine Geometrie zurück, welche der mengentheoretischen Vereinigung der Geometrien entspricht.

Synopsis

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

Beschreibung

Der Ausgabetypp kann eine MULTI*/Mehrfach-, eine Einzel- oder eine Sammelgeometrie sein. Hat 2 Varianten. Variante 1 vereinigt 2 Geometrien zu einer neuen Geometrie, welche keine überschneidenden Bereiche mehr aufweist. Variante 2 ist eine Aggregatfunktion, die eine Geometriemenge entgegennimmt und diese zu einer einzelnen ST_Geometry vereinigt, welche ebenfalls keine überschneidenden Bereiche mehr aufweist.

Aggregat Version:: Diese Funktion gibt eine Mehrfachgeometrie oder eine Einzelgeometrie von einem Satz an Geometrien zurück. Die Funktion ST_Union() ist in der Terminologie von PostgreSQL eine Aggregatfunktion. Dies bedeutet, dass sie so wie die SUM() und AVG() Funktionen und so wie die meisten Aggregatfunktionen mit Datenzeilen arbeitet. Eine NULL-Geometrie wird ignoriert.

Nicht-Aggregat Version: Diese Funktion gibt eine Geometrie zurück, die eine Vereinigung von zwei Eingabegeometrien darstellt. Der Ausgabetypp kann eine MULTI*, NON-MULTI oder eine GEOMETRYCOLLECTION sein. Wenn eine der beiden Geometrien NULL ist, so wird NULL zurückgegeben.



Note

ST_Collect und ST_Union sind oftmals untereinander austauschbar. ST_Union ist im Allgemeinen um Größenordnungen langsamer als ST_Collect, da es versucht die Grenzen aufzulösen und die Geometrie neu zu sortieren um sicherzustellen, dass eine erzeugte MULTI* keine überschneidenden Bereiche aufweist.

Wird vom GEOS Modul ausgeführt

Anmerkung: diese Funktion wurde früher GeomUnion() genannt und wurde von "Union" umbenannt, da UNION ein reserviertes SQL-Wort ist.

Verfügbarkeit: 1.4.0 - ST_Union wurde erweitert. ST_Union(geomarray) wurde eingeführt und ebenso wurde die Aggregatfunktion in PostgreSQL schneller gemacht. Wenn Sie GEOS 3.1.0+ verwenden, so verwendet ST_Union den schnelleren, kaskadierenden Algorithmus für die Vereinigung. Letzterer ist unter <http://blog.cleverelephant.ca/2009/01/must-faster-unions-in-postgis-14.html> beschrieben



This method implements the **OpenGIS Simple Features Implementation Specification for SQL 1.1**. s2.1.1.3



Note

Die Aggregatversion ist in der OGC SPEC nicht ausdrücklich definiert.



This method implements the SQL/MM specification. SQL-MM 3: 5.1.19 der Z-Index (Höhe) wenn Polygone beteiligt sind.

Beispiele

Aggregat Beispiel

```
SELECT stusps,
       ST_Multi(ST_Union(f.the_geom)) as singlegeom
FROM sometable As f
GROUP BY stusps
```

Nicht-Aggregat Beispiel

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

st_astext
```

```

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

SELECT ST_AsText(ST_Union(ST_GeomFromText('POINT(1 2)'),
                          ST_GeomFromText('POINT(1 2') ) ));
st_astext
-----
POINT(1 2)

--3D-Beispiel - Eine Art 3D-Unterstützung (mit gemischten Dimensionen!)
SELECT ST_AsEWKT(st_union(the_geom))
FROM
(SELECT ST_GeomFromEWKT('POLYGON((-7 4.2,-7.1 4.2,-7.1 4.3,
-7 4.2))') as the_geom
UNION ALL
SELECT ST_GeomFromEWKT('POINT(5 5 5)') as the_geom
UNION ALL
      SELECT ST_GeomFromEWKT('POINT(-2 3 1)') as the_geom
UNION ALL
SELECT ST_GeomFromEWKT('LINESTRING(5 5 5, 10 10 10)') as the_geom ) as foo;

st_asewkt
-----
GEOMETRYCOLLECTION(POINT(-2 3 1),LINESTRING(5 5 5,10 10 10),POLYGON((-7 4.2 5,-7.1 4.2 5,-7.1 4.3 5,-7 4.2 5)));

--3D-Beispiel ohne gemischte Dimensionen
SELECT ST_AsEWKT(st_union(the_geom))
FROM
(SELECT ST_GeomFromEWKT('POLYGON((-7 4.2 2,-7.1 4.2 3,-7.1 4.3 2,
-7 4.2 2))') as the_geom
UNION ALL
SELECT ST_GeomFromEWKT('POINT(5 5 5)') as the_geom
UNION ALL
      SELECT ST_GeomFromEWKT('POINT(-2 3 1)') as the_geom
UNION ALL
SELECT ST_GeomFromEWKT('LINESTRING(5 5 5, 10 10 10)') as the_geom ) as foo;

st_asewkt
-----
GEOMETRYCOLLECTION(POINT(-2 3 1),LINESTRING(5 5 5,10 10 10),POLYGON((-7 4.2 2,-7.1 4.2 3,-7.1 4.3 2,-7 4.2 2)));

--Beispiele mit dem neuen Feld/Array Konstrukt
SELECT ST_Union(ARRAY(SELECT the_geom FROM sometable));

SELECT ST_AsText(ST_Union(ARRAY[ST_GeomFromText('LINESTRING(1 2, 3 4)'),
                          ST_GeomFromText('LINESTRING(3 4, 4 5)')])) As wktunion;

--wktunion---
MULTILINESTRING((3 4,4 5),(1 2,3 4))

```

Siehe auch

[ST_Collect](#) [ST_UnaryUnion](#)

8.11.37 ST_UnaryUnion

ST_UnaryUnion — Wie ST_Union, arbeitet aber auf der Ebene der Geometriebestandteile.

Synopsis

```
geometry ST_UnaryUnion(geometry geom);
```

Beschreibung

Anders als ST_union löst ST_UnaryUnion die Grenzen zwischen den Bestandteilen des Mehrfachpolygons (invalid) auf und führt eine Vereinigung der Komponenten einer Sammelgeometrie durch. Alle Bestandteile der Eingabegeometrie werden als valide angenommen, sodass Sie kein valides Mehrfachpolygon von einem invaliden, sich selbst überschneidenden Polygon erhalten können.

Diese Funktion kann für Knotenberechnungen an einer Menge von Linienzügen verwendet werden. Sie können ST_UnaryUnion und ST_Collect mischen, um die Anzahl der Geometrien, bei denen die Grenzen auf einmal aufgelöst werden abzustimmen. Auf diese Weise kann eine Balance zwischen ST_Union und ST_MemUnion gefunden werden, die sowohl den Arbeitsspeicher als auch die CPU-Zeit schont.



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

Verfügbarkeit: 2.0.0 - benötigt GEOS >= 3.3.0.

Siehe auch

[ST_Union](#), [ST_MemUnion](#), [ST_Collect](#), [ST_Node](#)

8.11.38 ST_VoronoiLines

ST_VoronoiLines — Gibt die Grenzen zwischen den Zellen des Voronoi Diagramms aus, das aus den Knoten der Geometrie konstruiert wurde.

Synopsis

```
geometry ST_VoronoiLines( g1 geometry , tolerance float8 , extend_to geometry );
```

Beschreibung

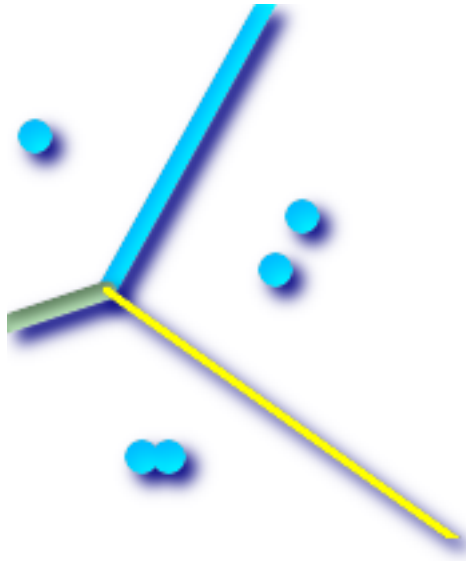
ST_VoronoiLines berechnet ein zweidimensionales **Voronoi Diagramm** aus den Knoten der gegebenen Geometrie und stellt die Grenzen zwischen den Zellen des Diagramms als MultiLineString dar.

Optionale Parameter:

- 'tolerance' : Die Entfernung innerhalb derer Knoten als ident betrachtet werden. Die Robustheit des Algorithmus kann verbessert werden, wenn die Entfernungstoleranz nicht mit Null angegeben wird. (Standardwert = 0.0)
- 'extend_to' : Wird eine Geometrie als "extend_to" Parameter zur Verfügung gestellt, so wird das Diagramm erweitert, um die Einhüllende der "extend_to"-Geometrie zu erfassen. Dies geschieht solange die Einhüllende nicht kleiner als die Standardeinhüllende ist. (Standardwert = NULL)

Verfügbarkeit: 2.3.0 - benötigt GEOS >= 3.5.0.

Beispiele



Voronoi Linien mit einer Toleranz von 30 Einheiten

```
SELECT ST_VoronoiLines(geom, 30) As geom
FROM (SELECT 'MULTIPOINT (50 30, 60 30, 100 100,10 150, 110 120) '::geometry As geom ) ↔
      As g

-- ST_AsText output
MULTILINESTRING((135.555555555556 270,36.8181818181818 92.2727272727273) ↔
, (36.8181818181818 92.2727272727273,-110 43.3333333333333), (230 ↔
-45.7142857142858,36.8181818181818 92.2727272727273))
```

Siehe auch

[ST_DelaunayTriangles](#), [ST_VoronoiPolygons](#), [ST_Collect](#)

8.11.39 ST_VoronoiPolygons

ST_VoronoiPolygons — Gibt die Zellen des Voronoi Diagramms zurück, die aus den Knoten der Geometrie erzeugt wurden.

Synopsis

geometry **ST_VoronoiPolygons**(g1 geometry , tolerance float8 , extend_to geometry);

Beschreibung

ST_VoronoiPolygons berechnet ein zweidimensionales **Voronoi diagram** aus den Knoten der gegebenen Geometrie. Das Ergebnis ist eine Sammelgeometrie/GeometryCollection von Polygonen, deren Einhüllende größer ist als die Ausdehnung der Ausgangsknoten.

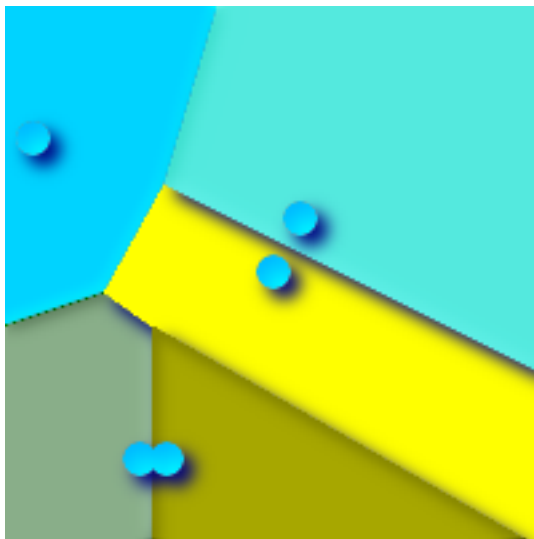
Optionale Parameter:

- 'tolerance' : Die Entfernung innerhalb derer Knoten als ident betrachtet werden. Die Robustheit des Algorithmus kann verbessert werden, wenn die Entfernungstoleranz nicht mit Null angegeben wird. (Standardwert = 0.0)

- 'extend_to' : Wird eine Geometrie als "extend_to" Parameter zur Verfügung gestellt, so wird das Diagramm erweitert, um die Einhüllende der "extend_to"-Geometrie zu erfassen. Dies geschieht solange die Einhüllende nicht kleiner als die Standardeinhüllende ist. (Standardwert = NULL)

Verfügbarkeit: 2.3.0 - benötigt GEOS >= 3.5.0.

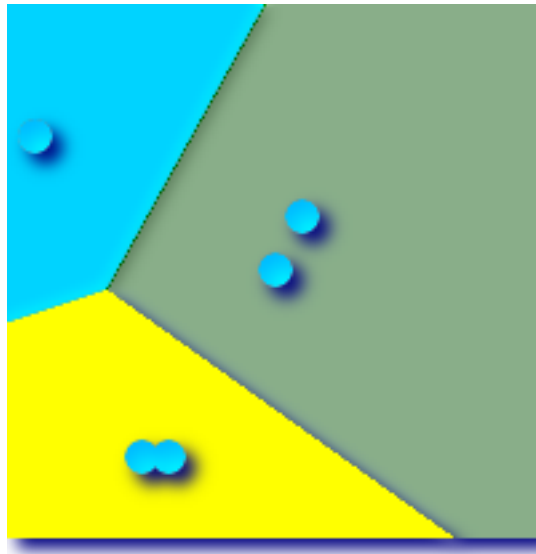
Beispiele



Punkte über dem Voronoi Diagramm

```
SELECT
    ST_VoronoiPolygons(geom) As geom
FROM (SELECT 'MULTIPOINT (50 30, 60 30, 100 100,10 150, 110 120) '::geometry As geom ) ↵
    As g;

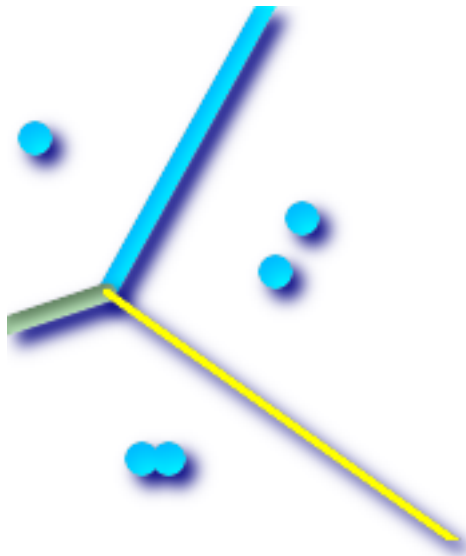
-- Ausgabe mit ST_AsText
GEOMETRYCOLLECTION(POLYGON((-110 43.3333333333333,-110 270,100.5 270,59.3478260869565 ↵
    132.826086956522,36.8181818181818 92.2727272727273,-110 43.3333333333333)), ↵
POLYGON((55 -90,-110 -90,-110 43.3333333333333,36.8181818181818 92.2727272727273,55 ↵
    79.2857142857143,55 -90)), ↵
POLYGON((230 47.5,230 -20.7142857142857,55 79.2857142857143,36.8181818181818 ↵
    92.2727272727273,59.3478260869565 132.826086956522,230 47.5)),POLYGON((230 ↵
    -20.7142857142857,230 -90,55 -90,55 79.2857142857143,230 -20.7142857142857)), ↵
POLYGON((100.5 270,230 270,230 47.5,59.3478260869565 132.826086956522,100.5 270)))
```



Voronoi mit einer Toleranz von 30 Einheiten

```
SELECT ST_VoronoiPolygons(geom, 30) As geom
FROM (SELECT 'MULTIPOINT (50 30, 60 30, 100 100,10 150, 110 120) '::geometry As geom ) ↔
      As g;

-- Ausgabe mit ST_AsText
GEOMETRYCOLLECTION(POLYGON((-110 43.3333333333333,-110 270,100.5 270,59.3478260869565 ↔
      132.826086956522,36.8181818181818 92.2727272727273,-110 43.3333333333333)),
POLYGON((230 47.5,230 -45.7142857142858,36.8181818181818 ↔
      92.2727272727273,59.3478260869565 132.826086956522,230 47.5)),POLYGON((230 ↔
      -45.7142857142858,230 -90,-110 -90,-110 43.3333333333333,36.8181818181818 ↔
      92.2727272727273,230 -45.7142857142858)),
POLYGON((100.5 270,230 270,230 47.5,59.3478260869565 132.826086956522,100.5 270)))
```

Voronoi als MultiLinestring mit einer Toleranz von 30 Einheiten

```
SELECT ST_VoronoiLines(geom, 30) As geom
FROM (SELECT 'MULTIPOINT (50 30, 60 30, 100 100,10 150, 110 120) '::geometry As geom ) ↔
      As g

-- ST_AsText output
MULTILINESTRING((135.555555555556 270,36.8181818181818 92.2727272727273) ↔
, (36.8181818181818 92.2727272727273,-110 43.3333333333333), (230 ↔
-45.7142857142858,36.8181818181818 92.2727272727273))
```

Siehe auch

[ST_DelaunayTriangles](#), [ST_VoronoiLines](#), [ST_Collect](#)

8.12 Lineare Referenzierung

8.12.1 ST_LineInterpolatePoint

ST_LineInterpolatePoint — Fügt einen Punkt entlang einer Linie ein. 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.

Synopsis

```
geometry ST_LineInterpolatePoint(geometry a_linestring, float8 a_fraction);
```

Beschreibung

Fügt einen Punkt entlang einer Linie ein. Der erste Parameter muss einen Linienzug beschreiben. Der zweite Parameter, in Float8-Darstellung mit den Werten von 0 bis 1, gibt jenen Bruchteil der Gesamtlänge des Linienzuges an, wo der Punkt liegen soll.

Siehe [ST_LineLocatePoint](#) um die nächstliegende Linie zu einem Punkt zu berechnen.

**Note**

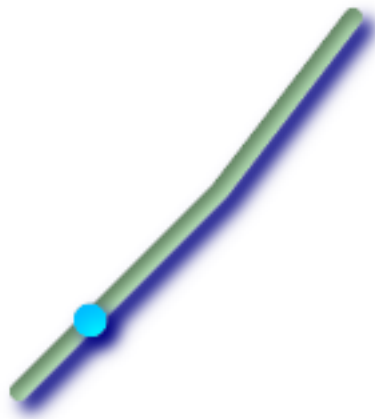
Ab Version 1.1.1 interpoliert diese Funktion auch M- und Z-Werte (falls vorhanden), während frühere Versionen diese Werte auf 0.0 setzten.

Verfügbarkeit: 0.8.2, Z und M Unterstützung wurde mit 1.1.1 hinzugefügt

Änderung: 2.1.0. Bis zu 2.0.x wurde diese Funktion mit `ST_Line_Interpolate_Point` bezeichnet.



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

Beispiele

Ein Linienzug mit dem interpolierten Punkt bei Position 0.20 (20%)

```
--Gibt einen Punkt zurück, der entlang einer Linie bei 20% liegt
SELECT ST_AsEWKT(ST_LineInterpolatePoint(the_line, 0.20))
      FROM (SELECT ST_GeomFromEWKT('LINESTRING(25 50, 100 125, 150 190)') as the_line) As foo;
      st_asewkt
-----
POINT(51.5974135047432 76.5974135047432)
```

```
--Gibt einen Punkt auf halber Strecke einer 3D-Linie zurück
SELECT ST_AsEWKT(ST_LineInterpolatePoint(the_line, 0.5))
      FROM (SELECT ST_GeomFromEWKT('LINESTRING(1 2 3, 4 5 6, 6 7 8)') as the_line) As foo;
      st_asewkt
-----
POINT(3.5 4.5 5.5)
```

```
--findet den nächstgelegenen Punkt auf einer Linie zu einem Punkt oder zu einer andere Geometrie
SELECT ST_AsText(ST_LineInterpolatePoint(foo.the_line, ST_LineLocatePoint(foo.the_line, ST_GeomFromText('POINT(4 3)'))))
```

```
FROM (SELECT ST_GeomFromText('LINESTRING(1 2, 4 5, 6 7)') As the_line) As foo;
      st_astext
-----
POINT(3 4)
```

Siehe auch

[ST_AsText](#), [ST_AsEWKT](#), [ST_Length](#), [ST_LineLocatePoint](#)

8.12.2 ST_LineLocatePoint

ST_LineLocatePoint — Returns a float between 0 and 1 representing the location of the closest point on LineString to the given Point, as a fraction of total 2d line length.

Synopsis

```
float8 ST_LineLocatePoint(geometry a_linestring, geometry a_point);
```

Beschreibung

Returns a float between 0 and 1 representing the location of the closest point on LineString to the given Point, as a fraction of total **2d line** length.

You can use the returned location to extract a Point ([ST_LineInterpolatePoint](#)) or a substring ([ST_LineSubstring](#)).

This is useful for approximating numbers of addresses

Verfügbarkeit: 1.1.0

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

Beispiele

```
--Rough approximation of finding the street number of a point along the street
--Note the whole foo thing is just to generate dummy data that looks
--like house centroids and street
--We use ST_DWithin to exclude
--houses too far away from the street to be considered on the street
SELECT ST_AsText(house_loc) As as_text_house_loc,
       startstreet_num +
       CAST( (endstreet_num - startstreet_num)
            * ST_LineLocatePoint(street_line, house_loc) As integer) As
       street_num
FROM
  (SELECT ST_GeomFromText('LINESTRING(1 2, 3 4)') As street_line,
        ST_MakePoint(x*1.01,y*1.03) As house_loc, 10 As startstreet_num,
        20 As endstreet_num
  FROM generate_series(1,3) x CROSS JOIN generate_series(2,4) As y)
As foo
WHERE ST_DWithin(street_line, house_loc, 0.2);
```

as_text_house_loc	street_num
POINT(1.01 2.06)	10
POINT(2.02 3.09)	15
POINT(3.03 4.12)	20

```
--find closest point on a line to a point or other geometry
SELECT ST_AsText(ST_LineInterpolatePoint(foo.the_line, ST_LineLocatePoint(foo.the_line, ↵
    ST_GeomFromText('POINT(4 3)'))))
FROM (SELECT ST_GeomFromText('LINESTRING(1 2, 4 5, 6 7)') As the_line) As foo;
    st_astext
-----
POINT(3 4)
```

Siehe auch

[ST_DWithin](#), [ST_Length2D](#), [ST_LineInterpolatePoint](#), [ST_LineSubstring](#)

8.12.3 ST_LineSubstring

ST_LineSubstring — Return a linestring being a substring of the input one starting and ending at the given fractions of total 2d length. Second and third arguments are float8 values between 0 and 1.

Synopsis

geometry **ST_LineSubstring**(geometry a_linestring, float8 startfraction, float8 endfraction);

Beschreibung

Return a linestring being a substring of the input one starting and ending at the given fractions of total 2d length. Second and third arguments are float8 values between 0 and 1. This only works with LINESTRINGS. To use with contiguous MULTI-LINESTRINGS use in conjunction with [ST_LineMerge](#).

Gleichbedeutend mit [ST_LineInterpolatePoint](#), wenn Anfangswert und Endwert ident sind.

Siehe [ST_LineLocatePoint](#) um die nächstliegende Linie zu einem Punkt zu berechnen.



Note

Ab Version 1.1.1 interpoliert diese Funktion auch M- und Z-Werte (falls vorhanden), während frühere Versionen unbestimmte Werte setzten.

Verfügbarkeit: 1.1.0, mit 1.1.1 wurde die Unterstützung für Z und M hinzugefügt

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



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

Beispiele



A linestring seen with 1/3 midrange overlaid (0.333, 0.666)

```
--Return the approximate 1/3 mid-range part of a linestring
SELECT ST_AsText(ST_Line_SubString(ST_GeomFromText('LINESTRING(25 50, 100 125, 150 190)'), ←
    0.333, 0.666));
```

st_astext ←

```
LINESTRING(69.2846934853974 94.2846934853974,100 125,111.700356260683 140.210463138888)

--The below example simulates a while loop in
--SQL using PostgreSQL generate_series() to cut all
--linestrings in a table to 100 unit segments
-- of which no segment is longer than 100 units
-- units are measured in the SRID units of measurement
-- It also assumes all geometries are LINESTRING or contiguous MULTILINESTRING
--and no geometry is longer than 100 units*10000
--for better performance you can reduce the 10000
--to match max number of segments you expect

SELECT field1, field2, ST_LineSubstring(the_geom, 100.00*n/length,
    CASE
        WHEN 100.00*(n+1) < length THEN 100.00*(n+1)/length
        ELSE 1
    END) As the_geom
FROM
    (SELECT sometable.field1, sometable.field2,
        ST_LineMerge(sometable.the_geom) AS the_geom,
        ST_Length(sometable.the_geom) As length
    FROM sometable
    ) AS t
CROSS JOIN generate_series(0,10000) AS n
WHERE n*100.00/length < 1;
```

Siehe auch

[ST_Length](#), [ST_LineInterpolatePoint](#), [ST_LineMerge](#)

8.12.4 ST_LocateAlong

ST_LocateAlong — Return a derived geometry collection value with elements that match the specified measure. Polygonal elements are not supported.

Synopsis

geometry **ST_LocateAlong**(geometry geom_with_measure, float8 a_measure, float8 offset);

Beschreibung

Return a derived geometry collection value with elements that match the specified measure. Polygonal elements are not supported.

If an offset is provided, the resultant will be offset to the left or right of the input line by the specified number of units. A positive offset will be to the left, and a negative one to the right.

Semantic is specified by: ISO/IEC CD 13249-3:200x(E) - Text for Continuation CD Editing Meeting

Verfügbarkeit: 1.1.0 über die alte Bezeichnung **ST_LocateAlong_Measure**.

Changed: 2.0.0 in prior versions this used to be called **ST_LocateAlong_Measure**. The old name has been deprecated and will be removed in the future but is still available.



Note

Use this function only for geometries with an M component



This function supports M coordinates.

Beispiele

```
SELECT ST_AsText(the_geom)
      FROM
      (SELECT ST_LocateAlong(
              ST_GeomFromText('MULTILINESTRINGM((1 2 3, 3 4 2, 9 4 3),
              (1 2 3, 5 4 5))'),3) As the_geom) As foo;

                                     st_asewkt
-----
MULTIPOINT M (1 2 3)

--Sammelgeometrien sind schwierige Viecher, weshalb man sie
--entladen/dump sollte um sie verdaulicher zu machen
SELECT ST_AsText((ST_Dump(the_geom)).geom)
      FROM
      (SELECT ST_LocateAlong(
              ST_GeomFromText('MULTILINESTRINGM((1 2 3, 3 4 2, 9 4 3),
              (1 2 3, 5 4 5))'),3) As the_geom) As foo;

      st_asewkt
-----
POINTM(1 2 3)
POINTM(9 4 3)
POINTM(1 2 3)
```

Siehe auch[ST_Dump](#), [ST_LocateBetween](#)**8.12.5 ST_LocateBetween**

ST_LocateBetween — Return a derived geometry collection value with elements that match the specified range of measures inclusively. Polygonal elements are not supported.

Synopsis

```
geometry ST_LocateBetween(geometry geomA, float8 measure_start, float8 measure_end, float8 offset);
```

Beschreibung

Return a derived geometry collection value with elements that match the specified range of measures inclusively. Polygonal elements are not supported.

Semantic is specified by: ISO/IEC CD 13249-3:200x(E) - Text for Continuation CD Editing Meeting

Verfügbarkeit: 1.1.0 über die alte Bezeichnung `ST_Locate_Between_Measures`.

Changed: 2.0.0 - in prior versions this used to be called `ST_Locate_Between_Measures`. The old name has been deprecated and will be removed in the future but is still available for backward compatibility.



This function supports M coordinates.

Beispiele

```
SELECT ST_AsText(the_geom)
      FROM
      (SELECT ST_LocateBetween(
              ST_GeomFromText('MULTILINESTRING M ((1 2 3, 3 4 2, 9 4 3),
              (1 2 3, 5 4 5))'),1.5, 3) As the_geom) As foo;

                                st_asewkt
-----
GEOMETRYCOLLECTION M (LINESTRING M (1 2 3,3 4 2,9 4 3),POINT M (1 2 3))

--Sammelgeometrien sind schwierige Viecher, weshalb man sie --entladen/dump sollte um sie  ←
verdaulicher zu machen
SELECT ST_AsText((ST_Dump(the_geom)).geom)
      FROM
      (SELECT ST_LocateBetween(
              ST_GeomFromText('MULTILINESTRING M ((1 2 3, 3 4 2, 9 4 3),
              (1 2 3, 5 4 5))'),1.5, 3) As the_geom) As foo;

                                st_asewkt
-----
LINESTRING M (1 2 3,3 4 2,9 4 3)
POINT M (1 2 3)
```

Siehe auch[ST_Dump](#), [ST_LocateAlong](#)

8.12.6 ST_LocateBetweenElevations

ST_LocateBetweenElevations — Return a derived geometry (collection) value with elements that intersect the specified range of elevations inclusively. Only 3D, 4D LINESTRINGS and MULTILINESTRINGS are supported.

Synopsis

geometry **ST_LocateBetweenElevations**(geometry geom_mline, float8 elevation_start, float8 elevation_end);

Beschreibung

Return a derived geometry (collection) value with elements that intersect the specified range of elevations inclusively. Only 3D, 3DM LINESTRINGS and MULTILINESTRINGS are supported.

Verfügbarkeit: 1.4.0



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

Beispiele

```
SELECT ST_AsEWKT(ST_LocateBetweenElevations(
    ST_GeomFromEWKT('LINESTRING(1 2 3, 4 5 6)'),2,4)) As ewelev;
-----
MULTILINESTRING((1 2 3,2 3 4))

SELECT ST_AsEWKT(ST_LocateBetweenElevations(
    ST_GeomFromEWKT('LINESTRING(1 2 6, 4 5 -1, 7 8 9)'),6,9)) As ewelev ←
;
-----
ewelev
-----
GEOMETRYCOLLECTION(POINT(1 2 6),LINESTRING(6.1 7.1 6,7 8 9))

--Sammelgeometrien sind schwierige Viecher, weshalb man sie --entladen/dump sollte um sie ←
verdaulicher zu machen
SELECT ST_AsEWKT((ST_Dump(the_geom)).geom)
FROM
    (SELECT ST_LocateBetweenElevations(
        ST_GeomFromEWKT('LINESTRING(1 2 6, 4 5 -1, 7 8 9)'),6,9) As ←
        the_geom) As foo;
-----
st_asewkt
-----
POINT(1 2 6)
LINESTRING(6.1 7.1 6,7 8 9)
```

Siehe auch

[ST_Dump](#)

8.12.7 ST_InterpolatePoint

ST_InterpolatePoint — Return the value of the measure dimension of a geometry at the point closed to the provided point.

Synopsis

```
float8 ST_InterpolatePoint(geometry line, geometry point);
```

Beschreibung

Return the value of the measure dimension of a geometry at the point closed to the provided point.

Verfügbarkeit: 2.0.0



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

Beispiele

```
SELECT ST_InterpolatePoint('LINESTRING M (0 0 0, 10 0 20)', 'POINT(5 5)');
 st_interpolatepoint
-----
                        10
```

Siehe auch

[ST_AddMeasure](#), [ST_LocateAlong](#), [ST_LocateBetween](#)

8.12.8 ST_AddMeasure

ST_AddMeasure — Return a derived geometry with measure elements linearly interpolated between the start and end points.

Synopsis

```
geometry ST_AddMeasure(geometry geom_mline, float8 measure_start, float8 measure_end);
```

Beschreibung

Return a derived geometry with measure elements linearly interpolated between the start and end points. If the geometry has no measure dimension, one is added. If the geometry has a measure dimension, it is over-written with new values. Only LINESTRINGS and MULTILINESTRINGS are supported.

Verfügbarkeit: 1.5.0



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

Beispiele

```
SELECT ST_AsText(ST_AddMeasure(
ST_GeomFromEWKT('LINESTRING(1 0, 2 0, 4 0)'),1,4)) As ewelev;
      ewelev
-----
LINESTRINGM(1 0 1,2 0 2,4 0 4)

SELECT ST_AsText(ST_AddMeasure(
ST_GeomFromEWKT('LINESTRING(1 0 4, 2 0 4, 4 0 4)'),10,40)) As ewelev;
      ewelev
```

```

-----
LINESTRING(1 0 4 10,2 0 4 20,4 0 4 40)

SELECT ST_AsText(ST_AddMeasure(
ST_GeomFromEWKT('LINESTRINGM(1 0 4, 2 0 4, 4 0 4)'),10,40)) As ewelev;
           ewelev
-----
LINESTRINGM(1 0 10,2 0 20,4 0 40)

SELECT ST_AsText(ST_AddMeasure(
ST_GeomFromEWKT('MULTILINESTRINGM((1 0 4, 2 0 4, 4 0 4),(1 0 4, 2 0 4, 4 0 4)'),10,70)) As ←
           ewelev;
                                   ewelev
-----
MULTILINESTRINGM((1 0 10,2 0 20,4 0 40),(1 0 40,2 0 50,4 0 70))

```

8.13 Unterstützung zeitbezogener Daten

8.13.1 ST_IsValidTrajectory

`ST_IsValidTrajectory` — Returns `true` if the geometry is a valid trajectory.

Synopsis

boolean **ST_IsValidTrajectory**(geometry line);

Beschreibung

Tell if a geometry encodes a valid trajectory. Valid trajectories are encoded as `LINESTRING` with `M` value growing from each vertex to the next.

Valid trajectories are expected as input to some spatio-temporal queries like [ST_ClosestPointOfApproach](#)

Verfügbarkeit: 2.2.0



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

Beispiele

```

-- A valid trajectory
SELECT ST_IsValidTrajectory(ST_MakeLine(
  ST_MakePointM(0,0,1),
  ST_MakePointM(0,1,2))
);
t

-- An invalid trajectory
SELECT ST_IsValidTrajectory(ST_MakeLine(ST_MakePointM(0,0,1), ST_MakePointM(0,1,0)));
NOTICE:  Measure of vertex 1 (0) not bigger than measure of vertex 0 (1)
st_isvalidtrajectory
-----
f

```

Siehe auch[ST_ClosestPointOfApproach](#)**8.13.2 ST_ClosestPointOfApproach**

`ST_ClosestPointOfApproach` — Returns the measure at which points interpolated along two lines are closest.

Synopsis

```
float8 ST_ClosestPointOfApproach(geometry track1, geometry track2);
```

Beschreibung

Returns the smallest measure at which point interpolated along the given lines are at the smallest distance. Inputs must be valid trajectories as checked by [ST_IsValidTrajectory](#). Null is returned if the trajectories do not overlap on the M range.

See [ST_LocateAlong](#) for getting the actual points at the given measure.

Verfügbarkeit: 2.2.0



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

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
WITH inp AS ( SELECT
  ST_AddMeasure('LINESTRING Z (0 0 0, 10 0 5) '::geometry,
    extract(epoch from '2015-05-26 10:00'::timestampz),
    extract(epoch from '2015-05-26 11:00'::timestampz)
  ) a,
  ST_AddMeasure('LINESTRING Z (0 2 10, 12 1 2) '::geometry,
    extract(epoch from '2015-05-26 10:00'::timestampz),
    extract(epoch from '2015-05-26 11:00'::timestampz)
  ) b
), cpa AS (
  SELECT ST_ClosestPointOfApproach(a,b) m FROM inp
), points AS (
  SELECT ST_Force3DZ(ST_GeometryN(ST_LocateAlong(a,m),1)) pa,
    ST_Force3DZ(ST_GeometryN(ST_LocateAlong(b,m),1)) pb
  FROM inp, cpa
)
SELECT to_timestamp(m) t,
  ST_Distance(pa,pb) distance
FROM points, cpa;
```

t	distance
2015-05-26 10:45:31.034483+02	1.96036833151395

Siehe auch

[ST_IsValidTrajectory](#), [ST_DistanceCPA](#), [ST_LocateAlong](#), [ST_AddMeasure](#)

8.13.3 ST_DistanceCPA

ST_DistanceCPA — Returns the distance between closest points of approach in two trajectories.

Synopsis

```
float8 ST_DistanceCPA(geometry track1, geometry track2);
```

Beschreibung

Returns the minimum distance two moving objects have ever been each-other. Inputs must be valid trajectories as checked by [ST_IsValidTrajectory](#). Null is returned if the trajectories do not overlap on the M range.

Verfügbarkeit: 2.2.0



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

Beispiele

```
-- Return the minimum distance of two objects moving between 10:00 and 11:00
WITH inp AS ( SELECT
  ST_AddMeasure('LINESTRING Z (0 0 0, 10 0 5)::geometry',
    extract(epoch from '2015-05-26 10:00'::timestampz),
    extract(epoch from '2015-05-26 11:00'::timestampz)
  ) a,
  ST_AddMeasure('LINESTRING Z (0 2 10, 12 1 2)::geometry',
    extract(epoch from '2015-05-26 10:00'::timestampz),
    extract(epoch from '2015-05-26 11:00'::timestampz)
  ) b
)
SELECT ST_DistanceCPA(a,b) distance FROM inp;

      distance
-----
1.96036833151395
```

Siehe auch

[ST_IsValidTrajectory](#), [ST_ClosestPointOfApproach](#), [ST_AddMeasure](#), [ST_Distance](#)

8.13.4 ST_CPAWithin

ST_CPAWithin — Returns true if the trajectories' closest points of approach are within the specified distance.

Synopsis

```
float8 ST_CPAWithin(geometry track1, geometry track2, float8 maxdist);
```

Beschreibung

Überprüft, ob sich 2 bewegte Objekte jemals innerhalb der angegebenen maximalen Entfernung befunden haben.

Inputs must be valid trajectories as checked by [ST_IsValidTrajectory](#). False is returned if the trajectories do not overlap on the M range.

Verfügbarkeit: 2.2.0



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

Beispiele

```
WITH inp AS ( SELECT
  ST_AddMeasure('LINESTRING Z (0 0 0, 10 0 5)::geometry',
    extract(epoch from '2015-05-26 10:00'::timestampz),
    extract(epoch from '2015-05-26 11:00'::timestampz)
  ) a,
  ST_AddMeasure('LINESTRING Z (0 2 10, 12 1 2)::geometry',
    extract(epoch from '2015-05-26 10:00'::timestampz),
    extract(epoch from '2015-05-26 11:00'::timestampz)
  ) b
)
SELECT ST_CPWithin(a,b,2), ST_DistanceCPA(a,b) distance FROM inp;
```

st_cpwithin	distance
t	1.96521473776207

Siehe auch

[ST_IsValidTrajectory](#), [ST_ClosestPointOfApproach](#), [ST_DistanceCPA](#), [ST_Distance](#)

8.14 Unterstützung für lange andauernde Transaktionen/Long Transactions

Dieses Modul und die zugehörigen pl/pgsql Funktionen wurden implementiert, um "long locking" Unterstützung anzubieten. Dies wird von der [Web Feature Service](#) Spezifikation verlangt.



Note

Es muss [serializable transaction level](#) angewandt werden, da sonst der "Locking" Mechanismus versagt.

8.14.1 AddAuth

AddAuth — Fügt einen Berechtigungsschlüssel für die aktuelle Transaktion hinzu.

Synopsis

boolean **AddAuth**(text auth_token);

Beschreibung

Fügt einen Berechtigungsschlüssel für die aktuelle Transaktion hinzu.

Fügt zu einer temporären Tabelle mit der Bezeichnung "temp_lock_have_table" den aktuellen Identifikator der Transaktion und einen Berechtigungsschlüssel hinzu.

Verfügbarkeit: 1.1.3

Beispiele

```
SELECT LockRow('towns', '353', 'priscilla');
      BEGIN TRANSACTION;
          SELECT AddAuth('joey');
          UPDATE towns SET the_geom = ST_Translate(the_geom,2,2) WHERE gid = 353;
      COMMIT;

---Error--
ERROR:  UPDATE where "gid" = '353' requires authorization 'priscilla'
```

Siehe auch

[LockRow](#)

8.14.2 CheckAuth

CheckAuth — Legt einen Trigger auf eine Tabelle an um, basierend auf einen Token, Updates und Deletes von Zeilen zu verhindern oder zu erlauben.

Synopsis

```
integer CheckAuth(text a_schema_name, text a_table_name, text a_key_column_name);
integer CheckAuth(text a_table_name, text a_key_column_name);
```

Beschreibung

Legt einen Trigger auf eine Tabelle an, um über ein Autorisierungs-Token, Updates und Deletes von Zeilen zu verhindern oder zu erlauben. Identifiziert Zeilen über die Spalte <rowid_col>.

Wenn "a_schema_name" nicht angegeben ist, wird im aktuellen Schema nach der Tabelle gesucht.



Note

Wenn ein Autorisierungstrigger auf dieser Tabelle bereits existiert, gibt die Funktion eine Fehlermeldung aus. Wenn die Transaktionsunterstützung nicht aktiviert wurde, wird ein Fehler gemeldet.

Verfügbarkeit: 1.1.3

Beispiele

```
SELECT CheckAuth('public', 'towns', 'gid');
      result
      -----
         0
```

Siehe auch

[EnableLongTransactions](#)

8.14.3 DisableLongTransactions

DisableLongTransactions — Deaktiviert die Unterstützung für lang andauernde Transaktionen. Diese Funktion entfernt die Metadatentabellen der Unterstützung für lang andauernde Transaktionen und löscht alle Trigger der auf Lock geprüften Tabellen.

Synopsis

text **DisableLongTransactions**();

Beschreibung

Deaktiviert die Unterstützung von lang andauernden Transaktionen. Diese Funktion entfernt die Metadatentabellen der Unterstützung für lang andauernde Transaktionen und löscht alle Trigger der auf Locks überprüften Tabellen.

Löscht die Metadatentabelle mit der Bezeichnung `authorization_table` und den View mit der Bezeichnung `authorized_tables`, sowie alle Trigger mit der Bezeichnung `checkauthtrigger`

Verfügbarkeit: 1.1.3

Beispiele

```
SELECT DisableLongTransactions();
--result--
Long transactions support disabled
```

Siehe auch

[EnableLongTransactions](#)

8.14.4 EnableLongTransactions

EnableLongTransactions — Aktiviert die Unterstützung für lang andauernde Transaktionen. Diese Funktion erzeugt die benötigten Metadatentabellen und muss einmal aufgerufen werden bevor die anderen Funktionen dieses Abschnitts angewandt werden. Ein zweimaliger Aufruf ist harmlos.

Synopsis

text **EnableLongTransactions**();

Beschreibung

Aktiviert die Unterstützung für lang andauernde Transaktionen. Diese Funktion erzeugt die benötigten Metadatentabellen und muss einmal aufgerufen werden bevor die anderen Funktionen dieses Abschnitts angewandt werden. Ein zweimaliger Aufruf ist harmlos.

Erzeugt eine Metatable mit der Bezeichnung `authorization_table` und den View `authorized_tables`

Verfügbarkeit: 1.1.3

Beispiele

```
SELECT EnableLongTransactions();
--result--
Long transactions support enabled
```

Siehe auch

[DisableLongTransactions](#)

8.14.5 LockRow

LockRow — Setzt einen Lock/Autorisierung auf eine bestimmte Zeile in der Tabelle

Synopsis

```
integer LockRow(text a_schema_name, text a_table_name, text a_row_key, text an_auth_token, timestamp expire_dt);
integer LockRow(text a_table_name, text a_row_key, text an_auth_token, timestamp expire_dt);
integer LockRow(text a_table_name, text a_row_key, text an_auth_token);
```

Beschreibung

Setzt einen Lock/eine Autorisierung für eine bestimmte Zeile in der Tabelle `<authid>`. `<authid>` ist ein Text, `<expires>` ist ein Zeitstempel mit dem Standardwert `now()+1hour`. Gibt 1 aus, wenn ein Lock zugewiesen wurde, ansonsten 0 (bereits über eine andere Authentifizierung gesperrt)

Verfügbarkeit: 1.1.3

Beispiele

```
SELECT LockRow('public', 'towns', '2', 'joey');
LockRow
-----
1

--Joey hat den Datensatz bereits gesperrt und die arme Priscilla hat das Nachsehen
SELECT LockRow('public', 'towns', '2', 'priscilla');
LockRow
-----
0
```


Siehe auch[UnlockRows](#)

8.14.6 UnlockRows

UnlockRows — Entfernt alle Locks einer bestimmten Authorisierungs-ID. Gibt die Anzahl der freigegebenen Locks aus.

Synopsis

integer **UnlockRows**(text auth_token);

Beschreibung

Entfernt alle Locks einer bestimmten Authorisierungs-ID. Gibt die Anzahl der freigegebenen Locks aus.

Verfügbarkeit: 1.1.3

Beispiele

```
SELECT LockRow('towns', '353', 'priscilla');
        SELECT LockRow('towns', '2', 'priscilla');
        SELECT UnLockRows('priscilla');
        UnLockRows
        -----
                2
```

Siehe auch[LockRow](#)

8.15 Sonstige Funktionen

8.15.1 ST_Accum

ST_Accum — Aggregatfunktion. Erzeugt ein Feld mit Geometrien.

Synopsis

geometry[] **ST_Accum**(geometry set geomfield);

Beschreibung

Aggregatfunktion. Erzeugt ein Feld mit Geometrien.

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



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



This method supports Circular Strings and Curves



This function supports Polyhedral surfaces.



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

Beispiele

```
SELECT (ST_Accum(the_geom)) As all_em, ST_AsText((ST_Accum(the_geom))[1]) As grabone,
(ST_Accum(the_geom))[2:4] as grab_rest
      FROM (SELECT ST_MakePoint(a*CAST(random()*10 As integer), a*CAST( ←
random()*10 As integer), a*CAST(random()*10 As integer)) As ←
the_geom
      FROM generate_series(1,4) a) As foo;

all_em|grabone      | grab_rest
-----+-----
{0101000080000000000000000000001440000000000000024400000000000001040:
010100008000000000000000000000
0001840000000000000000002C400000000000000003040:
01010000800000000000000000003540000000000000038400000000000001840:
0101000080000000000000000000404000000000000003C4000000000000003040} |
POINT(5 10) | {01010000800000000000000000001840000000000002C4000000000000003040:
01010000800000000000000000003540000000000000000038400000000000001840:
0101000080000000000000000000404000000000000003C4000000000000003040}
(1 row)
```

Siehe auch

[ST_Collect](#)

8.15.2 Box2D

Box2D — Returns a BOX2D representing the maximum extents of the geometry.

Synopsis

box2d **Box2D**(geometry geomA);

Beschreibung

Returns a BOX2D representing the maximum extents of the geometry.

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



This method supports Circular Strings and Curves



This function supports Polyhedral surfaces.



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

Beispiele

```
SELECT Box2D(ST_GeomFromText('LINESTRING(1 2, 3 4, 5 6)'));
      box2d
      -----
      BOX(1 2,5 6)

SELECT Box2D(ST_GeomFromText('CIRCULARSTRING(220268 150415,220227 150505,220227 ↵
      150406)'));
      box2d
      -----
      BOX(220186.984375 150406,220288.25 150506.140625)
```

Siehe auch

[Box3D](#), [ST_GeomFromText](#)

8.15.3 Box3D

Box3D — Returns a BOX3D representing the maximum extents of the geometry.

Synopsis

box3d **Box3D**(geometry geomA);

Beschreibung

Returns a BOX3D representing the maximum extents of the geometry.

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



This method supports Circular Strings and Curves



This function supports Polyhedral surfaces.



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



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

Beispiele

```
SELECT Box3D(ST_GeomFromEWKT('LINESTRING(1 2 3, 3 4 5, 5 6 5)'));
      Box3d
      -----
      BOX3D(1 2 3,5 6 5)

SELECT Box3D(ST_GeomFromEWKT('CIRCULARSTRING(220268 150415 1,220227 150505 1,220227 ↵
      150406 1)'));
      Box3d
      -----
      BOX3D(220227 150406 1,220268 150415 1)
```

Siehe auch[Box2D](#), [ST_GeomFromEWKT](#)**8.15.4 ST_EstimatedExtent**

ST_EstimatedExtent — Return the 'estimated' extent of the given spatial table. The estimated is taken from the geometry column's statistics. The current schema will be used if not specified.

Synopsis

```
box2d ST_EstimatedExtent(text schema_name, text table_name, text geocolumn_name, boolean parent_ony);
box2d ST_EstimatedExtent(text schema_name, text table_name, text geocolumn_name);
box2d ST_EstimatedExtent(text table_name, text geocolumn_name);
```

Beschreibung

Return the 'estimated' extent of the given spatial table. The estimated is taken from the geometry column's statistics. The current schema will be used if not specified. The default behavior is to also use statistics collected from children tables (tables with INHERITS) if available. If 'parent_ony' is set to TRUE, only statistics for the given table are used and children tables are ignored.

For PostgreSQL>=8.0.0 statistics are gathered by VACUUM ANALYZE and resulting extent will be about 95% of the real one.

**Note**

In absence of statistics (empty table or no ANALYZE called) this function returns NULL. Prior to version 1.5.4 an exception was thrown instead.

For PostgreSQL<8.0.0 statistics are gathered by update_geometry_stats() and resulting extent will be exact.

Verfügbarkeit: 1.0.0

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



This method supports Circular Strings and Curves

Beispiele

```
SELECT ST_EstimatedExtent('ny', 'edges', 'the_geom');
--result--
BOX(-8877653 4912316,-8010225.5 5589284)

SELECT ST_EstimatedExtent('feature_poly', 'the_geom');
--result--
BOX(-124.659652709961 24.6830825805664,-67.7798080444336 49.0012092590332)
```

Siehe auch[ST_Extent](#)

8.15.5 ST_Expand

ST_Expand — Returns bounding box expanded in all directions from the bounding box of the input geometry. Uses double-precision

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

This function returns a bounding box expanded from the bounding box of the input, either by specifying a single distance with which the box should be expanded in all directions, or by specifying an expansion distance for each direction. Uses double-precision. Can be very useful for distance queries, or to add a bounding box filter to a query to take advantage of a spatial index.

In addition to the geometry version of ST_Expand, which is the most commonly used, variants are provided that accept and produce internal BOX2D and BOX3D data types.

ST_Expand is similar in concept to ST_Buffer, except while buffer expands the geometry in all directions, ST_Expand expands the bounding box an x,y,z unit amount.

Die Einheiten entsprechen den Einheiten des im Einsatz befindlichen Koordinatenreferenzsystems, welches durch die SRID angegeben wird.



Note

Pre 1.3, ST_Expand was used in conjunction with distance to do indexable queries. Something of the form `the_geom && ST_Expand('POINT(10 20)', 10) AND ST_Distance(the_geom, 'POINT(10 20)') < 10` Post 1.2, this was replaced with the easier ST_DWithin construct.



Note

Availability: 1.5.0 behavior changed to output double precision instead of float4 coordinates.
Enhanced: 2.0.0 support for Polyhedral surfaces, Triangles and TIN was introduced.
Enhanced: 2.3.0 support was added to expand a box by different amounts in different dimensions.



This function supports Polyhedral surfaces.



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

Beispiele



Note

Examples below use US National Atlas Equal Area (SRID=2163) which is a meter projection

```
--10 meter expanded box around bbox of a linestring
SELECT CAST(ST_Expand(ST_GeomFromText('LINESTRING(2312980 110676,2312923 110701,2312892 110714)', 2163),10) As box2d);
--10 meter expanded 3d box of a 3d box
SELECT ST_Expand(CAST('BOX3D(778783 2951741 1,794875 2970042.61545891 10)' As box3d),10)
--10 meter geometry astext rep of a expand box around a point geometry
SELECT ST_AsEWKT(ST_Expand(ST_GeomFromEWKT('SRID=2163;POINT(2312980 110676)'),10));
SRID=2163;POLYGON((2312970 110666,2312970 110686,2312990 110686,2312990 110666,2312970 110666))
```

Siehe auch

[ST_AsEWKT](#), [ST_Buffer](#), [ST_DWithin](#), [ST_GeomFromEWKT](#), [ST_GeomFromText](#), [ST_SRID](#)

8.15.6 ST_Extent

ST_Extent — an aggregate function that returns the bounding box that bounds rows of geometries.

Synopsis

box2d **ST_Extent**(geometry set geomfield);

Beschreibung

ST_Extent returns a bounding box that encloses a set of geometries. The **ST_Extent** function is an "aggregate" function in the terminology of SQL. That means that it operates on lists of data, in the same way the **SUM()** and **AVG()** functions do.

Since it returns a bounding box, the spatial Units are in the units of the spatial reference system in use denoted by the **SRID**

ST_Extent is similar in concept to Oracle Spatial/Locator's **SDO_AGGR_MBR**



Note

Since **ST_Extent** returns a bounding box, the **SRID** meta-data is lost. Use **ST_SetSRID** to force it back into a geometry with **SRID** meta data. The coordinates are in the units of the spatial ref of the original geometries.



Note

ST_Extent will return boxes with only an x and y component even with (x,y,z) coordinate geometries. To maintain x,y,z use **ST_3DExtent** instead.



Note
Verfügbarkeit: 1.4.0

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



This function supports Polyhedral surfaces.



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

Beispiele



Note
Untere Beispiele nutzen Massachusetts State Plane ft (SRID=2249)

```
SELECT ST_Extent(the_geom) as bextent FROM sometable;
                                st_bextent
-----
BOX(739651.875 2908247.25,794875.8125 2970042.75)

--Return extent of each category of geometries
SELECT ST_Extent(the_geom) as bextent
FROM sometable
GROUP BY category ORDER BY category;

                                bextent                                |          name
-----+-----
BOX(778783.5625 2951741.25,794875.8125 2970042.75) | A
BOX(751315.8125 2919164.75,765202.6875 2935417.25) | B
BOX(739651.875 2917394.75,756688.375 2935866)      | C

--Force back into a geometry
-- and render the extended text representation of that geometry
SELECT ST_SetSRID(ST_Extent(the_geom),2249) as bextent FROM sometable;

                                bextent
-----
SRID=2249;POLYGON((739651.875 2908247.25,739651.875 2970042.75,794875.8125 2970042.75,
794875.8125 2908247.25,739651.875 2908247.25))
```

Siehe auch

[ST_AsEWKT](#), [ST_3DExtent](#), [ST_SetSRID](#), [ST_SRID](#)

8.15.7 ST_3DExtent

ST_3DExtent — an aggregate function that returns the box3D bounding box that bounds rows of geometries.

Synopsis

box3d **ST_3DExtent**(geometry set geomfield);

Beschreibung

ST_3DExtent returns a box3d (includes Z coordinate) bounding box that encloses a set of geometries. The ST_3DExtent function is an "aggregate" function in the terminology of SQL. That means that it operates on lists of data, in the same way the SUM() and AVG() functions do.

Since it returns a bounding box, the spatial Units are in the units of the spatial reference system in use denoted by the SRID



Note

Since ST_3DExtent returns a bounding box, the SRID meta-data is lost. Use ST_SetSRID to force it back into a geometry with SRID meta data. The coordinates are in the units of the spatial ref of the original geometries.

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

Changed: 2.0.0 In prior versions this used to be called ST_Extent3D



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



This method supports Circular Strings and Curves



This function supports Polyhedral surfaces.



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

Beispiele

```
SELECT ST_3DExtent(foo.the_geom) As b3extent
FROM (SELECT ST_MakePoint(x,y,z) As the_geom
      FROM generate_series(1,3) As x
      CROSS JOIN generate_series(1,2) As y
      CROSS JOIN generate_series(0,2) As Z) As foo;

b3extent
-----
BOX3D(1 1 0,3 2 2)

--Gibt die Ausdehnung mehrerer erhöhter Kreisbögen aus
SELECT ST_3DExtent(foo.the_geom) As b3extent
FROM (SELECT ST_Translate(ST_Force_3DZ(ST_LineToCurve(ST_Buffer(ST_MakePoint(x,y),1))),0,0, ←
      z) As the_geom
      FROM generate_series(1,3) As x
      CROSS JOIN generate_series(1,2) As y
      CROSS JOIN generate_series(0,2) As Z) As foo;

b3extent
-----
BOX3D(1 0 0,4 2 2)
```

Siehe auch

[ST_Extent](#), [ST_Force3DZ](#)

8.15.8 Find_SRID

Find_SRID — The syntax is `find_srid(a_db_schema, a_table, a_column)` and the function returns the integer SRID of the specified column by searching through the `GEOMETRY_COLUMNS` table.

Synopsis

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

Beschreibung

The syntax is `find_srid(<db/schema>, <table>, <column>)` and the function returns the integer SRID of the specified column by searching through the `GEOMETRY_COLUMNS` table. If the geometry column has not been properly added with the `AddGeometryColumns()` function, this function will not work either.

Beispiele

```
SELECT Find_SRID('public', 'tiger_us_state_2007', 'the_geom_4269');
find_srid
-----
4269
```

Siehe auch

[ST_SRID](#)

8.15.9 ST_MemSize

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

Synopsis

integer **ST_MemSize**(geometry geomA);

Beschreibung

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

This is a nice compliment to PostgreSQL built in functions `pg_column_size`, `pg_size_pretty`, `pg_relation_size`, `pg_total_relation_size`.

Note



`pg_relation_size` which gives the byte size of a table may return byte size lower than `ST_MemSize`. This is because `pg_relation_size` does not add toasted table contribution and large geometries are stored in TOAST tables.
`pg_total_relation_size` - umfasst die Tabelle, die TOAST-Tabellen und die Indizes.
`pg_column_size` returns how much space a geometry would take in a column considering compression, so may be lower than `ST_MemSize`



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



This method supports Circular Strings and Curves



This function supports Polyhedral surfaces.



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

Changed: 2.2.0 name changed to `ST_MemSize` to follow naming convention. In prior versions this function was called `ST_Mem_Size`, old name deprecated though still available.

Beispiele

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

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

```
SELECT ST_MemSize(ST_GeomFromText('CIRCULARSTRING(220268 150415,220227 150505,220227
150406)'));
---
73

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

Siehe auch

8.15.10 ST_PointInsideCircle

`ST_PointInsideCircle` — Is the point geometry inside the circle defined by `center_x`, `center_y`, `radius`

Synopsis

boolean **ST_PointInsideCircle**(geometry a_point, float center_x, float center_y, float radius);

Beschreibung

The syntax for this functions is `ST_PointInsideCircle(<geometry>,<circle_center_x>,<circle_center_y>,<radius>)`. Returns the true if the geometry is a point and is inside the circle. Returns false otherwise.



Note

Dies funktioniert nur für Punkte, wie die Bezeichnung verrät

Verfügbarkeit: 1.2

Changed: 2.2.0 In prior versions this used to be called `ST_Point_Inside_Circle`

Beispiele

```
SELECT ST_PointInsideCircle(ST_Point(1,2), 0.5, 2, 3);
 st_pointinsidecircle
-----
t
```

Siehe auch

[ST_DWithin](#)

8.16 Außergewöhnliche Funktionen

Diese selten genutzten Funktionen sollten nur im Falle beschädigten Daten verwendet werden. Sie werden zur Fehlerbeseitigung und zur Reparatur von Dingen verwendet, die unter normalen Umständen nicht vorkommen sollten.

8.16.1 PostGIS_AddBBox

`PostGIS_AddBBox` — Fügt der Geometrie ein Umgebungsrechteck hinzu.

Synopsis

`geometry` **`PostGIS_AddBBox`**(`geometry geomA`);

Beschreibung

Fügt ein Umgebungsrechteck zur Geometrie hinzu. Dies macht die, auf das Umgebungsrechteck bezogenen, Abfragen schneller, erhöht aber die Größe der Geometrie.



Note

Umgebungsrechtecke werden üblicherweise automatisch zu den Geometrien hinzugefügt, so dass es nicht nötig ist dies händisch zu tun, außer das Umgebungsrechteck/Bounding Box wurde irgendwie beschädigt, oder Sie verfügen über eine alte Installation die noch keine Umgebungsrechtecke kannte. Falls dies der Fall ist, müssen Sie die Alten löschen und neu hinzufügen.



This method supports Circular Strings and Curves

Beispiele

```
UPDATE sometable
SET the_geom = PostGIS_AddBBox(the_geom)
WHERE PostGIS_HasBBox(the_geom) = false;
```

Siehe auch

[PostGIS_DropBBox](#), [PostGIS_HasBBox](#)

8.16.2 PostGIS_DropBBox

PostGIS_DropBBox — Löscht das zwischengespeicherte Umgebungsrechteck der Geometrie.

Synopsis

geometry **PostGIS_DropBBox**(geometry geomA);

Beschreibung

Löscht das zwischengespeicherte Umgebungsrechteck der Geometrie. Dies verringert die Größe der Geometrie, macht aber die auf Umgebungsrechtecke aufbauenden Abfragen langsamer. Wird auch zum Löschen eines beschädigten Umgebungsrechtecks benutzt. Ein guter Hinweis auf ein beschädigtes Umgebungsrechteck ist, wenn ST_Intersects und andere Abfragen über räumliche Beziehungen, Geometrien auslassen die eigentlich TRUE zurückgeben sollten.

Note



Umgebungsrechtecke werden üblicherweise automatisch zu den Geometrien hinzugefügt, um die Geschwindigkeit von Abfragen zu steigern, somit ist es nicht nötig dies händisch zu tun, außer das Umgebungsrechteck wurde irgendwie beschädigt, oder Sie verfügen über eine alte Installation die noch keine Umgebungsrechtecke kannte. 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, das zwischengespeicherte Umgebungsrechtecke 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 BBox wie unten angeführt neu hinzufügt, oder einen PostgreSQL-Dump einspielt.



This method supports Circular Strings and Curves

Beispiele

```
--Dieses Beispiel löscht unrichtige Umgebungsrechtecke
--Durch die Anwendung von ST_AsBinary - vor Box2D - wird die Neuberechnung des ←
  Umgebungsrechteckes erzwungen,
--and Box2D auf die Tabellengeometrie angewendet
--gibt immer das zwischengespeicherte Umgebungsrechteck zurück.
      UPDATE sometable
SET the_geom = PostGIS_DropBBox(the_geom)
WHERE Not (Box2D(ST_AsBinary(the_geom)) = Box2D(the_geom));

      UPDATE sometable
SET the_geom = PostGIS_AddBBox(the_geom)
WHERE Not PostGIS_HasBBOX(the_geom);
```

Siehe auch

[PostGIS_AddBBox](#), [PostGIS_HasBBox](#), [Box2D](#)

8.16.3 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.



This method supports Circular Strings and Curves

Beispiele

```
SELECT the_geom
FROM sometable WHERE PostGIS_HasBBox(the_geom) = false;
```

Siehe auch

[PostGIS_AddBBox](#), [PostGIS_DropBBox](#)

Chapter 9

Raster Referenz

Im Folgenden sind die gebräuchlichsten Funktionen aufgeführt, die zur Zeit durch PostGIS-Raster zur Verfügung gestellt werden. Es gibt noch zusätzliche Funktionen, die zur Unterstützung von Rasterobjekten benötigt werden und für den Anwender nur geringe Bedeutung haben.

`raster` ist ein neuer PostGIS Datentyp, der zum Speichern und zur Analyse von Rasterdaten verwendet wird.

Um Raster aus Rasterdateien zu laden, siehe Section 5.1

Die Beispiele dieser Referenz benutzen eine Rastertabelle, die mit folgendem Code aus Dummy-Rastern erstellt wurde

```
CREATE TABLE dummy_rast(rid integer, rast raster);
INSERT INTO dummy_rast(rid, rast)
VALUES (1,
('01' -- little endian (uint8 ndr)
||
'0000' -- version (uint16 0)
||
'0000' -- nBands (uint16 0)
||
'000000000000000040' -- scaleX (float64 2)
||
'00000000000000840' -- scaleY (float64 3)
||
'000000000000E03F' -- ipX (float64 0.5)
||
'000000000000E03F' -- ipY (float64 0.5)
||
'0000000000000000' -- skewX (float64 0)
||
'0000000000000000' -- skewY (float64 0)
||
'00000000' -- SRID (int32 0)
||
'0A00' -- width (uint16 10)
||
'1400' -- height (uint16 20)
)::rast
),
-- Raster: 5 x 5 Zellen, 3 Bänder, PT_8BUI Zelltyp, NODATA = 0
(2, ('01000003009A9999999999A93F9A9999999999A9BF000000E02B274A' ||
'410000000077195641000000000000000000000000000000000000000000000000 ←
FFFFFFFFFF050005000400FDFFDFEFDFEFDFEFDF9FAFEF' ||
' ←
EFCF9FBFDFEFDFCFEFEFEFE04004E627AADD16076B4F9FE6370A9F5FE59637AB0E54F58617087040046566487A1506C
')::rast);
```

9.1 Datentypen zur Unterstützung von Rastern.

9.1.1 geomval

geomval — Ein räumlicher Datentyp mit zwei Feldern - **geom** (enthält das geometrische Element) und **val** (enthält den Zellwert eines Rasterbandes in Doppelter Genauigkeit).

Beschreibung

geomval ist ein 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 Ausgabebetyp in der Familie der Rasterüberlagerungsfunktionen um ein Rasterband in Polygone zu zerlegen.

Siehe auch

Section [14.6](#)

9.1.2 addbandarg

addbandarg — Ein zusammengesetzter Datentyp, der als Eingabewert für die Funktion "**ST_AddBand**" verwendet wird und sowohl Attribute als auch Initialwert des neuen Bandes festlegt.

Beschreibung

Ein zusammengesetzter Datentyp, der als Eingabewert für die Funktion "**ST_AddBand**" verwendet wird und sowohl Attribute als auch Initialwert des neuen Bandes festlegt.

index integer Ein von 1 aufwärts zählender Wert, der die Position unter den Rasterbänder angibt, an der das neue Band eingefügt werden soll. Wenn er NULL ist wird das neue Band am Ende der Rasterbänder hinzugefügt.

pixeltype text Der Datentyp der Rasterzellen/"Pixel Type" des neuen Bandes. Einer jener Datentypen, die unter **ST_BandPixelType** definiert sind.

initialvalue double precision Der Ausgangswert, auf den die Zellen eines neuen Bandes gesetzt werden.

nodataval double precision Der NODATA-Wert des neuen Bandes. Wenn NULL, dann wird für das neue Band kein NODATA-Wert vergeben.

Siehe auch

[ST_AddBand](#)

9.1.3 rastbandarg

rastbandarg — Ein zusammengesetzter Datentyp, der verwendet wird um den Raster und, über einen Index, das Band des Rasters anzugeben.

Beschreibung

Ein zusammengesetzter Datentyp, der verwendet wird um den Raster und, über einen Index, das Band des Rasters anzugeben.

rast raster Besagter Raster

nband integer Der 1-basierte Wert, welcher das betreffende Rasterband kennzeichnet

Siehe auch[ST_MapAlgebra](#)**9.1.4 raster**

raster — Der räumliche Datentyp Raster

Beschreibung

Raster ist ein Geodatentyp, der verwendet wird um digitale Bilder darzustellen. Diese Daten können zum Beispiel von JPEGs, TIFFs, PNGs oder digitalen Höhenmodellen stammen. Jeder Raster kann aus 1 oder mehreren Bänder bestehen, diese wiederum bestehen aus einzelnen Zellen denen Werte zugewiesen werden können. Raster können georeferenziert werden.

**Note**

Verlangt, dass PostGIS mit GDAL-Unterstützung kompiliert wurde. Gegenwärtig können Raster implizit in den Geometry Datentyp umgewandelt werden, allerdings gibt diese Umwandlung die [ST_ConvexHull](#) des Rasters zurück. Diese automatische Typumwandlung kann möglicherweise in der nahen Zukunft entfernt werden; verlassen Sie sich daher bitte nicht darauf.

Typumwandlung

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

Typumwandlung nach	Verhaltensweise
Geometrie	implizit

Siehe auchChapter [9](#)**9.1.5 reclassarg**

reclassarg — Ein Zusammengesetzter Datentyp, der als Eingabewert für die Funktion "ST_Reclass" dient und die Neuklassifizierung festlegt.

Beschreibung

Ein Zusammengesetzter Datentyp, der als Eingabewert für die Funktion "ST_Reclass" dient und die Neuklassifizierung festlegt.

nband integer Die Nummer des Bandes das neu klassifiziert werden soll.

reclassexpr text Ein Ausdruck, der aus "range:map_range" Abbildungen besteht, die als Intervalle dargestellt und durch Beistriche getrennt sind. Der Ausdruck gibt an, wie die alten Zellwerte auf die neuen Zellwerte des Bandes abgebildet werden sollen. "(" bedeutet >, ")" bedeutet <, "]" < oder gleich, "[" > oder gleich

1. [a-b] = a <= x <= b
2. (a-b] = a < x <= b
3. [a-b) = a <= x < b


```
4. (a-b) = a < x < b
```

Die runden Klammern sind bei dieser Notation optional, d.h. "a-b" ist gleichbedeutend mit "(a-b)".

***pixeltype* text** Einer der unter [ST_BandPixelType](#) definierten Datentypen

***nodataval* double precision** Der Zellwert, der als NODATA/NULL betrachtet werden soll. Bei der Ausgabe von Bildern, die Transparenz unterstützen, bleibt dieser leer.

Beispiel: Band 2 in 8BUI, mit einem NODATA-Wert von 255, umgruppieren.

```
SELECT ROW(2, '0-100:1-10, 101-500:11-150, 501 - 10000: 151-254', '8BUI', 255)::reclassarg;
```

Beispiel: Band 1 in 1BB, mit einem unbestimmten NODATA-Wert, umgruppieren.

```
SELECT ROW(1, '0-100]:0, (100-255:1', '1BB', NULL)::reclassarg;
```

Siehe auch

[ST_Reclass](#)

9.1.6 summarystats

summarystats — Ein zusammengesetzter Datentyp, welcher von den Funktionen [ST_SummaryStats](#) und [ST_SummaryStatsAgg](#) zurückgegeben wird.

Beschreibung

Ein zusammengesetzter Datentyp, welcher von [ST_SummaryStats](#) und [ST_SummaryStatsAgg](#) zurückgegeben wird.

***count* integer** Die Summe aller Zellen, die für eine zusammenfassende Statistik abgezählt wurden.

***sum* double precision** Die Summe aller abgezählten Zellwerte.

***mean* double precision** Der arithmetische Mittelwert aller abgezählten Zellwerte.

***stddev* double precision** Die Standardabweichung aller abgezählten Zellwerte.

***min* double precision** Der Minimalwert aller abgezählten Zellwerte.

***max* double precision** Der Maximalwert aller abgezählten Zellwerte.

Siehe auch

[ST_SummaryStats](#), [ST_SummaryStatsAgg](#)

9.1.7 unionarg

unionarg — Ein zusammengesetzter Datentyp, der als Eingabewert für die Funktion "[ST_Union](#)" dient. Dieser Datentyp bestimmt die zu behandelnden Bänder, sowie die Verhaltensweise des UNION Operators.

Beschreibung

Ein zusammengesetzter Datentyp, der als Eingabewert für die Funktion "ST_Union" dient. Dieser Datentyp bestimmt die zu behandelnden Bänder, sowie die Verhaltensweise des UNION Operators.

nband integer Der 1-basierte Wert, welcher das Band aller Input-Raster kennzeichnet, die bearbeitet werden sollen.

uniontype text Die Variante des UNION Operators. Eine der unter **ST_Union** definierten Varianten.

Siehe auch

ST_Union

9.2 Rastermanagement

9.2.1 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, 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 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=true,
boolean num_bands=true, boolean pixel_types=true, boolean nodata_values=true , boolean out_db=true , boolean extent=true );
```

Beschreibung

Setzt die Constraints auf eine Rasterspalte. Diese werden verwendet um die Information in dem Rasterkatalog `raster_columns` anzuzeigen. Der Parameter `rastschema` bezeichnet das Tabellenschema in dem die Tabelle liegt. Die Ganzzahl `srid` verweist auf einen Eintrag in der Tabelle "spatial_ref_sys".

Der `raster2pgsql` Lader verwendet diese Funktion um Rastertabellen zu registrieren.

Gültige Bezeichnungen für Constraints: siehe Section 5.2.1 für weitere Details.

- `blocksize` bestimmt die Datenblockgröße von X und Y
- `blocksize_x` setzt die X-Kachel (die Breite der Kacheln in Pixel)
- `blocksize_y` setzt die Y-Kachel (die Höhe der Kacheln in Pixel)
- `extent` berechnet die räumliche Ausdehnung der ganzen Tabelle und setzt einen Constraint, der alle Raster auf diesen Ausschnitt beschränkt.
- `num_bands` Anzahl der Bänder
- `pixel_types` liest ein Feld mit Pixeltypen für jedes Band ein, und stellt sicher, dass alle Bänder denselben Pixeltyp haben

- `regular_blocking` setzt die Constraints für die räumliche Eindeutigkeit (keine zwei Raster dürfen räumlich ident sein) und für die Coverage-Kachel (der Raster wird an einem Coverage ausgerichtet)
- `same_alignment` stellt sicher, dass alle die selbe Ausrichtung haben, d.h. der Vergleich von zwei beliebigen Kacheln gibt `TRUE` zurück. Siehe [ST_SameAlignment](#).
- `srid` stellt sicher, dass alle die selbe SRID aufweisen
- Mehr -- alles was als Eingabe in die obere Funktion aufgeführt ist



Note

Diese Funktion leitet die Constraints von den Daten ab, die bereits in der Tabelle vorliegen. Damit Sie die Funktion anwenden können, müssen Sie zuerst die Rasterspalte erzeugen in die Sie anschließend die Daten laden.



Note

Falls Sie weitere Daten in Ihre Tabellen laden müssen, nachdem Sie bereits die Constraints gesetzt haben, können Sie DropRasterConstraints ausführen, wenn sich die räumliche Ausdehnung Ihrer Daten geändert hat.

Verfügbarkeit: 2.0.0

Beispiele: Basierend auf den Daten sämtliche Connstraints auf die Spalte setzen

```
CREATE TABLE myrasters(rid SERIAL primary key, rast raster);
INSERT INTO myrasters(rast)
SELECT ST_AddBand(ST_MakeEmptyRaster(1000, 1000, 0.3, -0.3, 2, 2, 0, 0,4326), 1, '8BSI':: ←
    text, -129, NULL);

SELECT AddRasterConstraints('myrasters'::name, 'rast'::name);

-- überprüfen, ob die Registrierung in dem View "raster_columns" korrekt ist --
SELECT srid, scale_x, scale_y, blocksize_x, blocksize_y, num_bands, pixel_types, ←
    nodata_values
    FROM raster_columns
    WHERE r_table_name = 'myrasters';

srid | scale_x | scale_y | blocksize_x | blocksize_y | num_bands | pixel_types | ←
nodata_values
-----+-----+-----+-----+-----+-----+-----+-----
4326 |      2 |      2 |      1000 |      1000 |          1 | {8BSI}      | {0}
```

Beispiele: Einen einzelnen Constraint setzen

```
CREATE TABLE public.myrasters2(rid SERIAL primary key, rast raster);
INSERT INTO myrasters2(rast)
SELECT ST_AddBand(ST_MakeEmptyRaster(1000, 1000, 0.3, -0.3, 2, 2, 0, 0,4326), 1, '8BSI':: ↵
    text, -129, NULL);

SELECT AddRasterConstraints('public'::name, 'myrasters2'::name, 'rast'::name,' ↵
    regular_blocking', 'blocksize');
-- Bestätigung--
NOTICE: Adding regular blocking constraint
NOTICE: Adding blocksize-X constraint
NOTICE: Adding blocksize-Y constraint
```

Siehe auch

Section 5.2.1, ST_AddBand, ST_MakeEmptyRaster, DropRasterConstraints, ST_BandPixelType, ST_SRID

9.2.2 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=false,
boolean num_bands=true, boolean pixel_types=true, boolean nodata_values=true, boolean out_db=true , boolean extent=true);
boolean DropRasterConstraints(name rastschema, name rasttable, name rastcolumn, text[] constraints);

```

Beschreibung

Löscht die Constraints eines PostGIS Rasters die sich auf eine Rastertabellenspalte beziehen und mit **AddRasterConstraints** hinzugefügt wurden. Nützlich um weitere Daten zu laden oder um die Daten einer Rasterspalte zu aktualisieren. Dies ist nicht notwendig, wenn Sie eine Rastertabelle oder eine Rasterspalte löschen wollen.

Zum Löschen einer Rastertabelle benutzen Sie bitte Standard-SQL:

```
DROP TABLE mytable
```

Um nur eine Rasterspalte zu löschen und den Rest der Tabelle zu belassen, verwenden Sie bitte Standard-SQL

```
ALTER TABLE mytable DROP COLUMN rast
```

Die Tabelle verschwindet aus dem `raster_columns` Katalog, wenn die Tabelle oder die Spalte gelöscht wurde. Wenn allerdings nur die Constraints gelöscht werden, so wird die Rasterspalte weiterhin im `raster_columns` Katalog aufgeführt, aber es ist keine zusätzliche Information mehr vorhanden - abgesehen vom Spalten- und Tabellennamen.

Verfügbarkeit: 2.0.0

Beispiele

```
SELECT DropRasterConstraints ('myrasters','rast');
----RESULT output ---
t

-- die Änderungen in raster_columns überprüfen --
SELECT srid, scale_x, scale_y, blocksize_x, blocksize_y, num_bands, pixel_types, ←
      nodata_values
      FROM raster_columns
      WHERE r_table_name = 'myrasters';

srid | scale_x | scale_y | blocksize_x | blocksize_y | num_bands | pixel_types| ←
nodata_values
-----+-----+-----+-----+-----+-----+-----+-----→
0 |          |         |            |            |           |            |
```

Siehe auch[AddRasterConstraints](#)**9.2.3 AddOverviewConstraints**

AddOverviewConstraints — Eine Rasterspalte als Übersicht für eine andere Rasterspalte kennzeichnen.

Synopsis

boolean **AddOverviewConstraints**(name ovschema, name ovtable, name ovcolumn, name refschema, name reftable, name refcolumn, int ovfactor);

boolean **AddOverviewConstraints**(name ovtable, name ovcolumn, name reftable, name refcolumn, int ovfactor);

Beschreibung

Fügt Constraints zu der Rasterspalte hinzu. Diese werden verwendet um die Information im `raster_overviews` Katalog anzuzeigen.

Der Parameter `ovfactor` gibt den Multiplikator für den Maßstab der Übersichtsspalte an: ein höherer "overview factor" bedingt eine niedrigere Auflösung.

Falls die Parameter `ovschema` und `refschema` weggelassen werden, so wird die erste so benannte Tabelle verwendet, die beim Durchlaufen des `search_path` gefunden wird.

Verfügbarkeit: 2.0.0

Beispiele

```
CREATE TABLE res1 AS SELECT
ST_AddBand(
  ST_MakeEmptyRaster(1000, 1000, 0, 0, 2),
  1, '8BSI'::text, -129, NULL
) r1;

CREATE TABLE res2 AS SELECT
ST_AddBand(
  ST_MakeEmptyRaster(500, 500, 0, 0, 4),
  1, '8BSI'::text, -129, NULL
) r2;

SELECT AddOverviewConstraints('res2', 'r2', 'res1', 'r1', 2);

-- überprüfen, ob die Registrierung im View "raster_overviews" korrekt ist --
SELECT o_table_name ot, o_raster_column oc,
       r_table_name rt, r_raster_column rc,
       overview_factor f
FROM raster_overviews WHERE o_table_name = 'res2';
  ot | oc |  rt  | rc | f
-----+-----+-----+-----+---
 res2 | r2 | res1 | r1 | 2
(1 row)
```

Siehe auch

Section [5.2.2](#), [DropOverviewConstraints](#), [ST_CreateOverview](#), [AddRasterConstraints](#)

9.2.4 DropOverviewConstraints

DropOverviewConstraints — Löscht die Markierung einer Rasterspalte, die festlegt dass sie als Übersicht für eine andere Spalte dient.

Synopsis

```
boolean DropOverviewConstraints(name ovschema, name ovtable, name ovcolumn);  
boolean DropOverviewConstraints(name ovtable, name ovcolumn);
```

Beschreibung

Jene Constraints einer Rasterspalte löschen, die sie als Übersicht einer anderen Rasterspalte in dem Rasterkatalog `raster_overviews` ausweisen.

Falls der Parameter `ovschema` weggelassen wird, so wird die erste so benannte Tabelle verwendet, die beim Durchlaufen des `search_path` gefunden wird.

Verfügbarkeit: 2.0.0

Siehe auch

Section [5.2.2](#), [AddOverviewConstraints](#), [DropRasterConstraints](#)

9.2.5 PostGIS_GDAL_Version

PostGIS_GDAL_Version — Gibt die von PostGIS verwendete Version der GDAL-Bibliothek aus.

Synopsis

```
text PostGIS_GDAL_Version();
```

Beschreibung

Gibt die von PostGIS verwendete Version der GDAL-Bibliothek aus. Überprüft und meldet, ob GDAL die zugehörigen Dateien findet.

Beispiele

```
SELECT PostGIS_GDAL_Version();  
       postgis_gdal_version  
-----  
GDAL 1.11dev, released 2013/04/13
```

Siehe auch

[postgis.gdal_datapath](#)

9.2.6 PostGIS_Raster_Lib_Build_Date

PostGIS_Raster_Lib_Build_Date — Gibt einen vollständigen Bericht aus, wann die Rasterbibliothek kompiliert wurde.

Synopsis

text **PostGIS_Raster_Lib_Build_Date()**;

Beschreibung

Gibt einen Bericht aus, wann die Rasterbibliothek kompiliert wurde.

Beispiele

```
SELECT PostGIS_Raster_Lib_Build_Date();
postgis_raster_lib_build_date
-----
2010-04-28 21:15:10
```

Siehe auch

[PostGIS_Raster_Lib_Version](#)

9.2.7 PostGIS_Raster_Lib_Version

PostGIS_Raster_Lib_Version — Gibt einen vollständigen Bericht über die Version und das Kompilationsdatum der Rasterbibliothek aus.

Synopsis

text **PostGIS_Raster_Lib_Version()**;

Beschreibung

Gibt einen vollständigen Bericht über die Version und das Kompilationsdatum der Rasterbibliothek aus.

Beispiele

```
SELECT PostGIS_Raster_Lib_Version();
postgis_raster_lib_version
-----
2.0.0
```

Siehe auch

[PostGIS_Lib_Version](#)

9.2.8 ST_GDALDrivers

ST_GDALDrivers — Gibt eine Liste der Rasterformate aus, die von Ihrer GDAL Bibliothek unterstützt werden. In diesen Formaten können Sie einen Raster mit **ST_AsGDALRaster** ausgeben.

Synopsis

setof record **ST_GDALDrivers**(integer OUT idx, text OUT short_name, text OUT long_name, text OUT create_options);

Beschreibung

Gibt eine Liste der Rasterformate aus - short_name, long_name und die Optionen des Urhebers - die von Ihrer GDAL-Bibliothek unterstützt werden. Verwenden Sie den "short_name" als Übergabewert für den Parameter format von **ST_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: 2.0.6, 2.1.3 - standardmäßig ist kein Treiber aktiviert, solange die GUC oder die Umgebungsvariable "gdal_enabled_drivers" nicht gesetzt sind.

Verfügbarkeit: 2.0.0 - GDAL >= 1.6.0.

Beispiele: Treiber-Liste

```
SET postgis.gdal_enabled_drivers = 'ENABLE_ALL';
SELECT short_name, long_name
FROM st_gdaldrivers()
ORDER BY short_name;
```

short_name	long_name
AAIGrid	Arc/Info ASCII Grid
ARG	Azavea Raster Grid format
DTED	DTED Elevation Raster
EHdr	ESRI .hdr Labelled
FIT	FIT Image
GIF	Graphics Interchange Format (.gif)
GPKG	GeoPackage
GS7BG	Golden Software 7 Binary Grid (.grd)
GSAG	Golden Software ASCII Grid (.grd)
GSBG	Golden Software Binary Grid (.grd)
GTiff	GeoTIFF
HF2	HF2/HFZ heightfield raster
HFA	Erdas Imagine Images (.img)
ILWIS	ILWIS Raster Map
INGR	Intergraph Raster
JPEG	JPEG JFIF
KMLSUPEROVERLAY	Kml Super Overlay
LCP	FARSITE v.4 Landscape File (.lcp)
MFF	Vexcel MFF Raster
NITF	National Imagery Transmission Format
PNG	Portable Network Graphics
R	R Object Data Store
RST	Idrisi Raster A.1
SAGA	SAGA GIS Binary Grid (.sdatt)
SRTMHGT	SRTMHGT File Format
USGSDEM	USGS Optional ASCII DEM (and CDED)
VRT	Virtual Raster
WMS	OGC Web Map Service
XPM	X11 PixMap Format
XYZ	ASCII Gridded XYZ
ZMap	ZMap Plus Grid

(31 rows)

Beispiele: Liste mit den Optionen für jeden Treiber

```
-- Ausgabe der XML-Spalte, mit den Erstellungsoptionen eines JPEGs, in eine Tabelle --
-- Sie können diese Optionen als Übergabewert für ST_AsGDALRaster verwenden
SELECT (xpath('@name', g.opt))[1]::text As oname,
       (xpath('@type', g.opt))[1]::text As otype,
       (xpath('@description', g.opt))[1]::text As descrip
FROM (SELECT unnest(xpath('/CreationOptionList/Option', create_options::xml)) As opt
FROM st_gdaldrivers()
WHERE short_name = 'JPEG') As g;
```

oname	otype	descrip
PROGRESSIVE	boolean	whether to generate a progressive JPEG
QUALITY	int	good=100, bad=0, default=75
WORLDFILE	boolean	whether to generate a worldfile
INTERNAL_MASK	boolean	whether to generate a validity mask
COMMENT	string	Comment
SOURCE_ICC_PROFILE	string	ICC profile encoded in Base64
EXIF_THUMBNAIL	boolean	whether to generate an EXIF thumbnail(overview). By default its max dimension will be 128
THUMBNAIL_WIDTH	int	Forced thumbnail width
THUMBNAIL_HEIGHT	int	Forced thumbnail height

(9 rows)

```
-- unbearbeitete XML-Ausgabe für die Erstellungsoptionen eines eoTiff --
SELECT create_options
FROM st_gdaldrivers()
WHERE short_name = 'GTiff';
```

```
<CreationOptionList>
  <Option name="COMPRESS" type="string-select">
    <Value>
>NONE</Value>
    <Value>
>LZW</Value>
    <Value>
>PACKBITS</Value>
    <Value>
>JPEG</Value>
    <Value>
>CCITTTRLE</Value>
    <Value>
>CCITTFAX3</Value>
    <Value>
>CCITTFAX4</Value>
    <Value>
>DEFLATE</Value>
  </Option>
  <Option name="PREDICTOR" type="int" description="Predictor Type"/>
  <Option name="JPEG_QUALITY" type="int" description="JPEG quality 1-100" default="75"/>
  <Option name="ZLEVEL" type="int" description="DEFLATE compression level 1-9" default ↵
    ="6"/>
  <Option name="NBITS" type="int" description="BITS for sub-byte files (1-7), sub-uint16 ↵
    (9-15), sub-uint32 (17-31)"/>
  <Option name="INTERLEAVE" type="string-select" default="PIXEL">
    <Value>
>BAND</Value>
    <Value>
>PIXEL</Value>
  </Option>
```

```

    <Option name="TILED" type="boolean" description="Switch to tiled format"/>
    <Option name="TFW" type="boolean" description="Write out world file"/>
    <Option name="RPB" type="boolean" description="Write out .RPB (RPC) file"/>
    <Option name="BLOCKXSIZE" type="int" description="Tile Width"/>
    <Option name="BLOCKYSIZE" type="int" description="Tile/Strip Height"/>
    <Option name="PHOTOMETRIC" type="string-select">
      <Value
>MINISBLACK</Value>
      <Value
>MINISWHITE</Value>
      <Value
>PALETTE</Value>
      <Value
>RGB</Value>
      <Value
>CMYK</Value>
      <Value
>YCBCR</Value>
      <Value
>CIELAB</Value>
      <Value
>ICCLAB</Value>
      <Value
>ITULAB</Value>
    </Option>
    <Option name="SPARSE_OK" type="boolean" description="Can newly created files have ↵
      missing blocks?" default="FALSE"/>
    <Option name="ALPHA" type="boolean" description="Mark first extrasample as being alpha ↵
      "/>
    <Option name="PROFILE" type="string-select" default="GDALGeoTIFF">
      <Value
>GDALGeoTIFF</Value>
      <Value
>GeoTIFF</Value>
      <Value
>BASELINE</Value>
    </Option>
    <Option name="PIXELTYPE" type="string-select">
      <Value
>DEFAULT</Value>
      <Value
>SIGNEDBYTE</Value>
    </Option>
    <Option name="BIGTIFF" type="string-select" description="Force creation of BigTIFF file ↵
      ">
      <Value
>YES</Value>
      <Value
>NO</Value>
      <Value
>IF_NEEDED</Value>
      <Value
>IF_SAFER</Value>
    </Option>
    <Option name="ENDIANNESS" type="string-select" default="NATIVE" description="Force ↵
      endianness of created file. For DEBUG purpose mostly">
      <Value
>NATIVE</Value>
      <Value
>INVERTED</Value>
      <Value
>LITTLE</Value>

```

```

    <Value
>BIG</Value>
    </Option>
    <Option name="COPY_SRC_OVERVIEWS" type="boolean" default="NO" description="Force copy ↵
      of overviews of source dataset (CreateCopy())"/>
</CreationOptionList
>

-- Ausgabe der XML-Spalte, mit den Erstellungsoptionen eines GTiff, in eine Tabelle --
SELECT (xpath('@name', g.opt))[1]::text As oname,
       (xpath('@type', g.opt))[1]::text As otype,
       (xpath('@description', g.opt))[1]::text As descrip,
       array_to_string(xpath('Value/text()', g.opt),', ' ) As vals
FROM (SELECT unnest(xpath('/CreationOptionList/Option', create_options::xml)) As opt
FROM st_gdaldrivers()
WHERE short_name = 'GTiff') As g;

```

oname	otype	descrip ↵ vals
COMPRESS	string-select	↵ NONE, LZW, ↵
PACKBITS, JPEG, CCITTRLE, CCITTFAX3, CCITTFAX4, DEFLATE		
PREDICTOR	int	Predictor Type ↵
JPEG_QUALITY	int	JPEG quality 1-100 ↵
ZLEVEL	int	DEFLATE compression level 1-9 ↵
NBITS	int	BITS for sub-byte files (1-7), sub-uint16 (9-15), sub ↵
-uint32 (17-31)		
INTERLEAVE	string-select	↵ BAND, PIXEL
TILED	boolean	Switch to tiled format ↵
TFW	boolean	Write out world file ↵
RPB	boolean	Write out .RPB (RPC) file ↵
BLOCKXSIZE	int	Tile Width ↵
BLOCKYSIZE	int	Tile/Strip Height ↵
PHOTOMETRIC	string-select	↵ MINISBLACK, ↵
MINISWHITE, PALETTE, RGB, CMYK, YCBCR, CIELAB, ICCLAB, ITULAB		
SPARSE_OK	boolean	Can newly created files have missing blocks? ↵
ALPHA	boolean	Mark first extrasample as being alpha ↵
PROFILE	string-select	↵ GDALGeoTIFF, ↵
GeoTIFF, BASELINE		
PIXELTYPE	string-select	↵ DEFAULT, ↵
SIGNEDBYTE		
BIGTIFF	string-select	Force creation of BigTIFF file ↵ YES, NO, IF_NEEDED, IF_SAFER
ENDIANNESS	string-select	Force endianness of created file. For DEBUG purpose ↵
mostly		NATIVE, INVERTED, LITTLE, BIG
COPY_SRC_OVERVIEWS	boolean	Force copy of overviews of source dataset (CreateCopy ↵

```
( ) |
(19 rows)
```

Siehe auch

[ST_AsGDALRaster](#), [ST_SRID](#), [postgis.gdal_enabled_drivers](#)

9.2.9 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);
```

Beschreibung

Änderung der SRID aller Raster in der vom Anwender angegebenen Spalte und Tabelle. Diese Funktion löscht alle entsprechenden Constraints (extent, alignment und die SRID) der Spalte bevor die SRID der Rasterspalte geändert wird.



Note

Die Daten des Rasters (Pixelwerte der Bänder) bleiben von dieser Funktion unangetastet. Es werden lediglich die Metadaten des Rasters geändert.

Verfügbarkeit: 2.1.0

Siehe auch

[UpdateGeometrySRID](#)

9.2.10 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');
```

Beschreibung

Erzeugt eine Übersichtstabelle mit skalierten Kacheln der Ursprungstabelle. Die Ausgabekacheln haben dieselbe Größe und haben die gleiche räumliche Ausdehnung wie die Eingabekacheln mit einer niedrigeren Auflösung (die Pixelgröße ist in beiden Richtungen $1/\text{factor}$ der Ursprünglichen).

Auf die Übersichtstabelle werden die Constraints gesetzt und sie steht über den Katalog `raster_overviews` zur Verfügung.

Die Optionen für den Algorithmus sind: 'NearestNeighbor', 'Bilinear', 'Cubic', 'CubicSpline' und 'Lanczos'. Siehe [GDAL Warp resampling methods](#) für weitere Details.

Verfügbarkeit: 2.2.0

Beispiel

Ausgabe mit besserer Qualität, aber langsamerer Formaterstellung

```
SELECT ST_CreateOverview('mydata.mytable'::regclass, 'rast', 2, 'Lanczos');
```

Schnellere Berechnung der Ausgabe mittels "Nearest Neighbor" (Standardeinstellung)

```
SELECT ST_CreateOverview('mydata.mytable'::regclass, 'rast', 2);
```

Siehe auch

[ST_Retile](#), [AddOverviewConstraints](#), [AddRasterConstraints](#), [Section 5.2.2](#)

9.3 Raster Constructors

9.3.1 ST_AddBand

ST_AddBand — Gibt einen Raster mit den neu hinzugefügten Band(Bändern) aus. Der Typ, der Ausgangswert und der Index für den Speicherort des Bandes kann angegeben werden. Wenn kein Index angegeben ist, wird das Band am Ende hinzugefügt.

Synopsis

- (1) raster **ST_AddBand**(raster rast, addbandarg[] addbandargset);
- (2) raster **ST_AddBand**(raster rast, integer index, text pixeltype, double precision initialvalue=0, double precision nodataval=NULL);
- (3) raster **ST_AddBand**(raster rast, text pixeltype, double precision initialvalue=0, double precision nodataval=NULL);
- (4) raster **ST_AddBand**(raster torast, raster fromrast, integer fromband=1, integer torastindex=at_end);
- (5) raster **ST_AddBand**(raster torast, raster[] fromrasts, integer fromband=1, integer torastindex=at_end);
- (6) raster **ST_AddBand**(raster rast, integer index, text outdbfile, integer[] outdbindex, double precision nodataval=NULL);
- (7) raster **ST_AddBand**(raster rast, text outdbfile, integer[] outdbindex, integer index=at_end, double precision nodataval=NULL);

Beschreibung

Gibt einen Raster mit einem an der gegebenen Position (index) neu hinzugefügten Band zurück. aus. Der Typ, der Ausgangswert und der Wert für NODATA kann übergeben werden. Wenn kein Index angegeben ist, wird das Band am Ende hinzugefügt. Wenn `fromband` nicht angegeben ist, wird Band 1 angenommen. Der Pixeltyp wird als Zeichenfolge übergeben, wie in [ST_BandPixelType](#) festgelegt. Falls ein bereits bestehender Index angegeben wird, werden alle folgenden Bänder $\geq$ diesem Index um 1 erhöht. Wenn ein größerer Ausgangswert als das Maximum des Pixeltyps angegeben ist, dann wird der Ausgangswert auf den höchsten erlaubten Wert des Pixeltyps gesetzt.

Bei der Variante, welche ein Feld von [addbandarg](#) entgegennimmt (Variante 1), ist ein bestimmter Indexwert von "addbandarg" auf den Raster zu dem Zeitpunkt bezogen, als das Band mit diesem "addbandarg" zum Raster hinzugefügt wurde. Siehe das Beispiel "Mehrere neue Bänder" unterhalb.

Bei der Variante, die ein Feld an Rastern (Variante 5) entgegennimmt, wird wenn `torast` NULL ist, das `fromband` Band eines jeden Rasters in diesem Feld, in einem neuen Raster akkumuliert.

Bei den Varianten, die ein `outdbfile` (Varianten 6 und 7) entgegennehmen, muss der Wert den vollständigen Pfad der Rasterdatei enthalten. Die Datei muss auch für die PostgreSQL-Instanz zugänglich sein.

Erweiterung: 2.1.0 - Unterstützung für "addbandarg" hinzugefügt.

Erweiterung: 2.1.0 Unterstützung für die neuen "out-db" Bänder hinzugefügt.

Beispiele: Ein einzelnes, neues Band

```
-- Ein weiteres Band vom Typ "vorzeichenlose 8-Bit Ganzzahl" mit einem Anfangswert von 200 ←
    für die Pixel
UPDATE dummy_rast
    SET rast = ST_AddBand(rast,'8BUI'::text,200)
WHERE rid = 1;
```

```
-- Erstellt einen leeren Raster mit 100x100 Einheiten, x und y der oberen linken Ecke sind ←
    jeweils 0, fügt 2 Bänder hinzu (Band 1 ist ein boolescher 0/1 Bit-Switch, Band2 ←
    beschränkt die Werte auf 0-15)
-- verwendet "addbandargs"
INSERT INTO dummy_rast(rid,rast)
    VALUES(10, ST_AddBand(ST_MakeEmptyRaster(100, 100, 0, 0, 1, -1, 0, 0, 0),
        ARRAY[
            ROW(1, '1BB'::text, 0, NULL),
            ROW(2, '4BUI'::text, 0, NULL)
        ]::addbandarg[]
    )
);
```

```
-- Ausgabe der Metadaten der Rasterbänder zur Kontrolle --
SELECT (bmd).*
FROM (SELECT ST_BandMetaData(rast,generate_series(1,2)) As bmd
    FROM dummy_rast WHERE rid = 10) AS foo;
```

```
--result --
pixeltype | nodatavalue | isoutdb | path
-----+-----+-----+-----
1BB       |             | f       |
4BUI      |             | f       |
```

```
-- Ausgabe der Metadaten des Rasters -
SELECT (rmd).width, (rmd).height, (rmd).numbands
FROM (SELECT ST_MetaData(rast) As rmd
    FROM dummy_rast WHERE rid = 10) AS foo;
```

```
-- result --
upperleftx | upperlefty | width | height | scalex | scaley | skewx | skewy | srid | ←
numbands
-----+-----+-----+-----+-----+-----+-----+-----+-----+-----
0 | 0 | 100 | 100 | 1 | -1 | 0 | 0 | 0 | ←
2
```

Beispiele: Mehrere neue Bänder

```
SELECT
    *
FROM ST_BandMetadata(
    ST_AddBand(
        ST_MakeEmptyRaster(10, 10, 0, 0, 1, -1, 0, 0, 0),
        ARRAY[
            ROW(NULL, '8BUI', 255, 0),
            ROW(NULL, '16BUI', 1, 2),
            ROW(2, '32BUI', 100, 12),
            ROW(2, '32BF', 3.14, -1)
        ]::addbandarg[]
    ),
    ARRAY[]::integer[]
```

```
);
```

bandnum	pixeltype	nodatavalue	isoutdb	path
1	8BUI	0	f	
2	32BF	-1	f	
3	32BUI	12	f	
4	16BUI	2	f	

```
-- Aggregiert das 1ste Band einer Tabelle mit ähnlichen Rastern zu einem einzelnen Raster
-- Soviele Bänder wie "test_types" und sovielen Zeilen (neuer Raster) wie Mäuse
-- ANMERKUNG: "ORDER BY test_type" wird erst ab PostgreSQL 9.0 unterstützt,
-- bei 8.4 und niedriger funktioniert dies meist, indem man die Daten in einer Unterabfrage ↵
-- sortiert (ohne Gewähr)
-- Der resultierende Raster hat ein Band für jeden "test_type", die Bänder sind ↵
-- alphabetisch nach dem "test_type" sortiert
-- Für Mausliebhaber: Bei dieser Übung werden keine Mäuse verletzt
SELECT
    mouse,
    ST_AddBand(NULL, array_agg(rast ORDER BY test_type), 1) As rast
FROM mice_studies
GROUP BY mouse;
```

Beispiele: Neues Out-db Band

```
SELECT
    *
FROM ST_BandMetadata(
    ST_AddBand(
        ST_MakeEmptyRaster(10, 10, 0, 0, 1, -1, 0, 0, 0),
        '/home/raster/mytestraster.tif'::text, NULL::int[]
    ),
    ARRAY[]::integer[]
);
```

bandnum	pixeltype	nodatavalue	isoutdb	path
1	8BUI		t	/home/raster/mytestraster.tif
2	8BUI		t	/home/raster/mytestraster.tif
3	8BUI		t	/home/raster/mytestraster.tif

Siehe auch

[ST_BandMetadata](#), [ST_BandPixelType](#), [ST_MakeEmptyRaster](#), [ST_MetaData](#), [ST_NumBands](#), [ST_Reclass](#)

9.3.2 ST_AsRaster

ST_AsRaster — Konvertiert den geometrischen Datentyp von PostGIS in einen PostGIS Raster.

Synopsis

```
raster ST_AsRaster(geometry geom, raster ref, text pixeltype, double precision value=1, double precision nodataval=0, boolean
touched=false);
raster ST_AsRaster(geometry geom, raster ref, text[] pixeltype=ARRAY['8BUI'], double precision[] value=ARRAY[1], double
precision[] nodataval=ARRAY[0], boolean touched=false);
```

```

raster ST_AsRaster(geometry geom, double precision scalex, double precision scaley, double precision gridx, double precision gridy, text pixeltype, double precision value=1, double precision nodataval=0, double precision skewx=0, double precision skewy=0, boolean touched=false);
raster ST_AsRaster(geometry geom, double precision scalex, double precision scaley, double precision gridx=NULL, double precision gridy=NULL, text[] pixeltype=ARRAY['8BUI'], double precision[] value=ARRAY[1], double precision[] nodataval=ARRAY[0], double precision skewx=0, double precision skewy=0, boolean touched=false);
raster ST_AsRaster(geometry geom, double precision scalex, double precision scaley, text pixeltype, double precision value=1, double precision nodataval=0, double precision upperleftx=NULL, double precision upperlefty=NULL, double precision skewx=0, double precision skewy=0, boolean touched=false);
raster ST_AsRaster(geometry geom, double precision scalex, double precision scaley, text[] pixeltype, double precision[] value=ARRAY[1], double precision[] nodataval=ARRAY[0], double precision upperleftx=NULL, double precision upperlefty=NULL, double precision skewx=0, double precision skewy=0, boolean touched=false);
raster ST_AsRaster(geometry geom, integer width, integer height, double precision gridx, double precision gridy, text pixeltype, double precision value=1, double precision nodataval=0, double precision skewx=0, double precision skewy=0, boolean touched=false);
raster ST_AsRaster(geometry geom, integer width, integer height, double precision gridx=NULL, double precision gridy=NULL, text[] pixeltype=ARRAY['8BUI'], double precision[] value=ARRAY[1], double precision[] nodataval=ARRAY[0], double precision skewx=0, double precision skewy=0, boolean touched=false);
raster ST_AsRaster(geometry geom, integer width, integer height, text pixeltype, double precision value=1, double precision nodataval=0, double precision upperleftx=NULL, double precision upperlefty=NULL, double precision skewx=0, double precision skewy=0, boolean touched=false);
raster ST_AsRaster(geometry geom, integer width, integer height, text[] pixeltype, double precision[] value=ARRAY[1], double precision[] nodataval=ARRAY[0], double precision upperleftx=NULL, double precision upperlefty=NULL, double precision skewx=0, double precision skewy=0, boolean touched=false);

```

Beschreibung

Konvertiert den geometrischen Datentyp von PostGIS in einen PostGIS Raster. Es gibt viele Varianten, die jeweils drei Gruppen von Möglichkeiten bieten um die Ausrichtung/alignment und die Pixelgröße des resultierenden Rasters zu bestimmen.

Die erste Gruppe besteht aus den ersten zwei Varianten und erzeugt einen Raster mit derselben Ausrichtung (`scalex`, `scaley`, `gridx` und `gridy`), dem selben Pixeltyp und NODATA Wert wie der gegebene Referenzraster. Üblicherweise wird der Referenzraster über einen Join übergeben, der aus der Tabelle mit der Geometrie und der Tabelle mit dem Referenzraster gebildet wird.

Die zweite Gruppe setzt sich aus vier Varianten zusammen und erlaubt Ihnen, die Dimensionen des Rasters über die Parameter der Pixelgröße festzulegen (`scalex` & `scaley` und `skewx` & `skewy`). Die `width` & `height` des resultierenden Rasters wird an die Ausdehnung der Geometrie angepasst. In den meisten Fällen müssen Sie eine Typumwandlung der Eingabewerte `scalex` & `scaley` von Integer nach Double Precision vornehmen, damit PostgreSQL die richtige Variante auswählt.

Die zweite Gruppe setzt sich aus vier Varianten zusammen und erlaubt Ihnen, die Dimensionen des Rasters über die Dimensionen des Rasters zu fixieren (`width` & `height`). Die Parameter der Pixelgröße (`scalex` & `scaley` und `skewx` & `skewy`) des resultierenden Rasters werden an die Ausdehnung der Geometrie angepasst.

Die ersten zwei Varianten der beiden letzten Gruppen erlauben es Ihnen, an einer beliebigen Ecke des Führungsgitters (`gridx` & `gridy`) auszurichten; und die letzten zwei Varianten nehmen die obere linke Ecke (`upperleftx` & `upperlefty`) entgegen.

Jede Variantengruppe erlaubt die Erstellung eines Rasters sowohl mit einem als auch mit mehreren Bändern. Um einen Raster mit mehreren Bändern zu erstellen, können Sie ein Feld mit Pixeltypen (`pixeltype[]`), ein Feld mit Ausgangswerten (`value`) und ein Feld mit NODATA Werten (`nodataval`) bereitstellen. Falls nicht, werden die Standardwerte - 8BUI für den Pixeltyp, 1 für den Ausgangswert und 0 für NODATA - angenommen.

Der Ausgaberraster hat dieselbe Koordinatenreferenz wie die Ausgangsgeometrie. Die einzige Ausnahme bilden Varianten mit einem Referenzraster. In diesem Fall erhält der resultierende Raster dieselbe SRID wie der Referenzraster.

Der optionale Parameter `touched` ist standardmäßig FALSE und wird auf die GDAL Rasterungsoption ALL_TOUCHED abgebildet, welche bestimmt, ob Pixel die von Linien oder Polygonen nur berührt werden, ebenfalls gerastert werden sollen; und nicht nur die Pixel auf einem Linienzug oder mit dem Mittelpunkt innerhalb des Polygons.

Dies ist insbesondere in Verbindung mit **ST_AsPNG** und der Funktionsfamilie **ST_AsGDALRaster**, für die Erstellung von JPEGs oder PNGs aus einer Geometrie direkt von der Datenbank heraus, nützlich.

Verfügbarkeit: 2.0.0 - GDAL >= 1.6.0.

**Note**

Zur Zeit können komplexe geometrische Datentypen wie Kurven, TINS und polyedrische Oberflächen nicht gerendert werden, was aber möglich sein sollte, sobald GDAL dies kann.

Beispiele: Ausgabe der Geometrien als PNG-Datei

Ein schwarzer Kreis

```
-- gibt einen schwarzen Kreiis aus, der 150 x 150 Pixel einnimmt --  
SELECT ST_AsPNG(ST_AsRaster(ST_Buffer(ST_Point(1,5),10),150, 150, '2BUI')));
```



Ein Beispiel für einen Puffer; die Grafik ist direkt in PostGIS erstellt

```
-- Die Bänder werden als RGB Bänder abgebildet - der Wert (118,154,118) ist aquamarin --  
SELECT ST_AsPNG(  
    ST_AsRaster(  
        ST_Buffer(  
            ST_GeomFromText('LINESTRING(50 50,150 150,150 50)'), 10,'join=bevel ←  
            '),  
            200,200,ARRAY['8BUI', '8BUI', '8BUI'], ARRAY[118,154,118], ARRAY ←  
            [0,0,0]));
```

Siehe auch

[ST_BandPixelType](#), [ST_Buffer](#), [ST_GDALDrivers](#), [ST_AsGDALRaster](#), [ST_AsPNG](#), [ST_AsJPEG](#), [ST_SRID](#)

9.3.3 ST_Band

ST_Band — Gibt einen oder mehrere Bänder eines bestehenden Rasters als neuen Raster aus. Nützlich um neue Raster aus bestehenden Rastern abzuleiten.

Synopsis

```
raster ST_Band(raster rast, integer[] nbands = ARRAY[1]);
raster ST_Band(raster rast, integer nband);
raster ST_Band(raster rast, text nbands, character delimiter=,);
```

Beschreibung

Gibt einen oder mehrere Bänder eines bestehenden Rasters in Form eines neuen Raster aus. Nützlich um neue Raster aus bestehenden Rastern abzuleiten, um einen Export auf ausgewählte Bänder einzuschränken, oder um die Reihenfolge der Rasterbänder neu zu ordnen. Wenn kein Band angegeben ist, wird Band 1 angenommen. Dient auch als Hilfsfunktion für verschiedene Funktionen, wie das Löschen eines Bandes.



Warning

Bei der Funktionsvariante mit `nbands` als Text, ist das Standardtrennzeichen `,`; d.h.: Sie können `'1,2,3'` abfragen und falls Sie ein anderes Trennzeichen verwenden wollen `ST_Band(rast, '1@2@3', '@')`. Wenn Sie mehrere Bänder abfragen, empfehlen wir Ihnen unbedingt die Feldvariante der Funktion zu verwenden, z.B. `ST_Band(rast, '{1,2,3}'::int[]);`, da die Variante mit der `text` Liste der Bänder in zukünftigen Versionen von PostGIS entfernt werden könnte.

Verfügbarkeit: 2.0.0

Beispiele

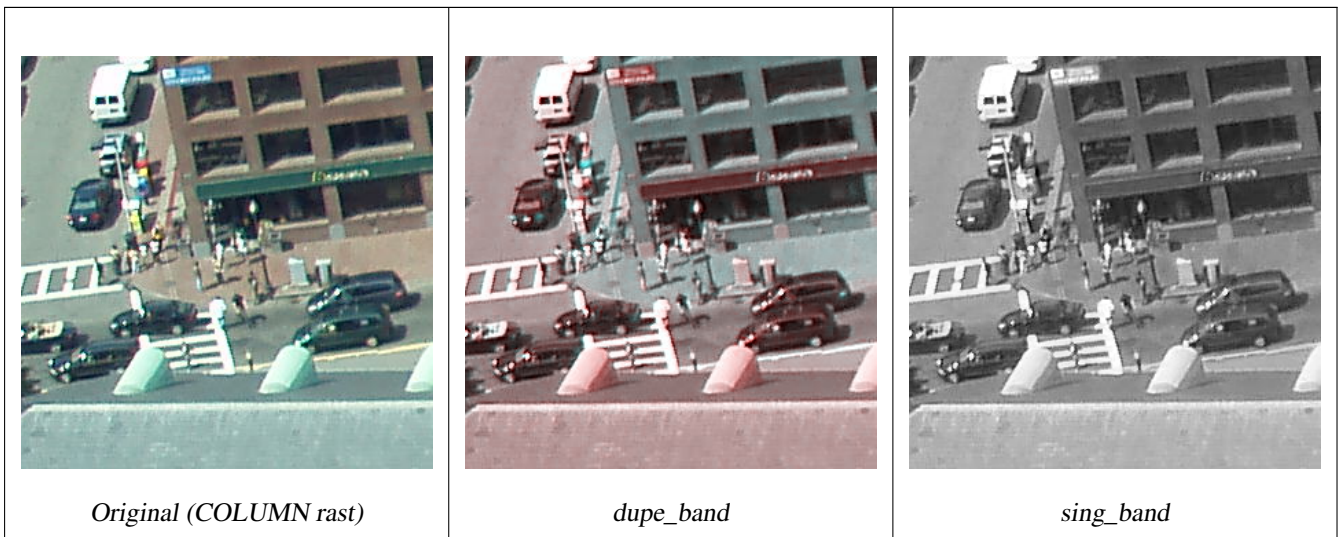
```
-- Erzeugt 2 neue Raster:: 1 enthält das Band 1 des Dummy, das Zweite enthält Band 2 des Dummy; anschließend als 2BUI neu klassifiziert
SELECT ST_NumBands(rast1) As numb1, ST_BandPixelType(rast1) As pix1,
       ST_NumBands(rast2) As numb2, ST_BandPixelType(rast2) As pix2
FROM (
  SELECT ST_Band(rast) As rast1, ST_Reclass(ST_Band(rast,3), '100-200):1, [200-254:2', '2 BUI') As rast2
  FROM dummy_rast
  WHERE rid = 2) As foo;
```

```
numb1 | pix1 | numb2 | pix2
-----+-----+-----+-----
1 | 8BUI | 1 | 2BUI
```

```
-- Gibt die Bänder 2 und 3 aus. Verwendet den Syntax zur Typumwandlung von Feldern
SELECT ST_NumBands(ST_Band(rast, '{2,3}'::int[])) As num_bands
FROM dummy_rast WHERE rid=2;
```

```
num_bands
-----
2
```

```
-- Gibt Die Bänder 2 und 3 aus. Verwendet ein Feld um die Bänder zu bestimmen
SELECT ST_NumBands(ST_Band(rast, ARRAY[2,3])) As num_bands
FROM dummy_rast
WHERE rid=2;
```



```
--Erstellt einen neuen Raster mit dem 2ten Band des Originals und zweimal dem 1sten Band,
-- sowie einen weiteren Raster mit dem dritten Band
SELECT rast, ST_Band(rast, ARRAY[2,1,1]) As dupe_band,
        ST_Band(rast, 3) As sing_band
FROM samples.than_chunked
WHERE rid=35;
```

Siehe auch

[ST_AddBand](#), [ST_NumBands](#), [ST_Reclass](#), [Chapter 9](#)

9.3.4 ST_MakeEmptyCoverage

ST_MakeEmptyCoverage — Bedeckt die georeferenzierte Fläche mit einem Gitter aus leeren Rasterkacheln.

Synopsis

raster ST_MakeEmptyCoverage(integer tilewidth, integer tileheight, integer width, integer height, double precision upperleftx, double precision upperlefty, double precision scalex, double precision scaley, double precision skewx, double precision skewy, integer srid=unknown);

Beschreibung

Erzeugt Rasterkacheln mit [ST_MakeEmptyRaster](#). Die Größe des Gitters wird über `width` & `height` angegeben. Die Kachelgröße über `tilewidth` & `tileheight`. Die abgedeckte georeferenzierte Fläche reicht vom oberen linken Eck (`upperleftx`, `upperlefty`) bis zum unteren rechten Eck (`upperleftx + width * scalex`, `upperlefty + height * scaley`).



Note

Beachten Sie bitte, dass bei Rastern "scaley" im Allgemeinen negativ und "scalex" positiv ist. Dadurch hat das untere rechte Eck einen niedrigeren Y-Wert und einen höheren X-Wert als die obere rechte Ecke.

Grundlegende Beispiele

Erzeugt ein 4x4 Gitter aus 16 Kacheln, um die WGS84 Fläche vom linken oberen Eck (22, 77) zum rechten unteren Eck (55, 33) abzudecken.

```
SELECT (ST_MetaData(tile)).* FROM ST_MakeEmptyCoverage(1, 1, 4, 4, 22, 33, (55 - 22)/(4)::float, (33 - 77)/(4)::float, 0., 0., 4326) tile;
```

upperleftx	upperlefty	width	height	scalex	scaley	skewx	skewy	srid	↔
numbands									
22	33	1	1	8.25	-11	0	0	4326	↔
	0								
30.25	33	1	1	8.25	-11	0	0	4326	↔
	0								
38.5	33	1	1	8.25	-11	0	0	4326	↔
	0								
46.75	33	1	1	8.25	-11	0	0	4326	↔
	0								
22	22	1	1	8.25	-11	0	0	4326	↔
	0								
30.25	22	1	1	8.25	-11	0	0	4326	↔
	0								
38.5	22	1	1	8.25	-11	0	0	4326	↔
	0								
46.75	22	1	1	8.25	-11	0	0	4326	↔
	0								
22	11	1	1	8.25	-11	0	0	4326	↔
	0								
30.25	11	1	1	8.25	-11	0	0	4326	↔
	0								
38.5	11	1	1	8.25	-11	0	0	4326	↔
	0								
46.75	11	1	1	8.25	-11	0	0	4326	↔
	0								
22	0	1	1	8.25	-11	0	0	4326	↔
	0								
30.25	0	1	1	8.25	-11	0	0	4326	↔
	0								
38.5	0	1	1	8.25	-11	0	0	4326	↔
	0								
46.75	0	1	1	8.25	-11	0	0	4326	↔
	0								

Siehe auch

[ST_MakeEmptyRaster](#)

9.3.5 ST_MakeEmptyRaster

ST_MakeEmptyRaster — Gibt einen leeren Raster (ohne Bänder), mit den gegebenen Dimensionen (width & height), upperleft X und Y, Pixelgröße, Rotation (scalex, scaley, skewx & skewy) und Koordinatenreferenzsystem (SRID), zurück. Wenn ein Raster übergeben wird, dann wird ein neuer Raster mit der selben Größe, Ausrichtung und SRID zurückgegeben. Wenn SRID nicht angegeben ist, wird das Koordinatenreferenzsystem auf "unknown" (0) gesetzt.

Synopsis

raster **ST_MakeEmptyRaster**(raster rast);

raster **ST_MakeEmptyRaster**(integer width, integer height, float8 upperleftx, float8 upperlefty, float8 scalex, float8 scaley,

```
float8 skewx, float8 skewy, integer srid=unknown);
raster ST_MakeEmptyRaster(integer width, integer height, float8 upperleftx, float8 upperlefty, float8 pixelsize);
```

Beschreibung

Gibt einen leeren Raster (ohne Bänder), mit den gegebenen Dimensionen (width & height), georeferenziert in geodätischen (oder geographischen) Koordinaten, oberes linkes X (upperleftx), oberes linkes Y (upperlefty), Pixelgröße, Rotation (scalex, scaley, skewx & skewy) und Koordinatenreferenzsystem (SRID), zurück.

Die letzte Version verwendet einen einzelnen Parameter um die Pixelgröße festzulegen. "scalex" wird auf diesen Übergabewert und "scaley" auf den negativen Wert davon gesetzt. "skewx" und "skewy" werden auf 0 gesetzt.

Wird ein bestehender Raster übergeben, so wird ein neuer Raster mit den gleichen Metadateneinstellungen erstellt (ohne die Bänder).

Wenn die SRID nicht angegeben ist, wird sie standardmäßig auf 0 gesetzt. Nachdem Sie einen leeren Raster erzeugt haben, werden Sie vermutlich Bänder hinzufügen und eventuell auch bearbeiten wollen. Siehe [ST_AddBand](#) um die Bänder festzulegen und [ST_SetValue](#) um Ausgangswerte für die Pixel zu setzen.

Beispiele

```
INSERT INTO dummy_rast(rid,rast)
VALUES(3, ST_MakeEmptyRaster( 100, 100, 0.0005, 0.0005, 1, 1, 0, 0, 4326) );
```

```
--verwendet einen bestehenden Raster als Vorlage für einen neuen Raster
INSERT INTO dummy_rast(rid,rast)
SELECT 4, ST_MakeEmptyRaster(rast)
FROM dummy_rast WHERE rid = 3;
```

```
-- Ausgabe der Metadaten des oben hinzugefügten Rasterss
SELECT rid, (md).*
FROM (SELECT rid, ST_MetaData(rast) As md
      FROM dummy_rast
      WHERE rid IN(3,4)) As foo;
```

```
-- output --
```

rid	upperleftx	upperlefty	width	height	scalex	scaley	skewx	skewy	srid	←
numbands										
3	0.0005	0.0005	100	100	1	1	0	0	4326	←
4	0.0005	0.0005	100	100	1	1	0	0	4326	←

Siehe auch

[ST_AddBand](#), [ST_MetaData](#), [ST_ScaleX](#), [ST_ScaleY](#), [ST_SetValue](#), [ST_SkewX](#), [ST_SkewY](#)

9.3.6 ST_Tile

ST_Tile — Gibt Raster, die aus einer Teilungsoperation des Eingaberasters resultieren, mit den gewünschten Dimensionen aus.

Synopsis

setof raster **ST_Tile**(raster rast, int[] nband, integer width, integer height, boolean padwithnodata=FALSE, double precision nodataval=NULL);

setof raster **ST_Tile**(raster rast, integer nband, integer width, integer height, boolean padwithnodata=FALSE, double precision nodataval=NULL);

setof raster **ST_Tile**(raster rast, integer width, integer height, boolean padwithnodata=FALSE, double precision nodataval=NULL);

Beschreibung

Gibt Raster, die aus einer Teilungsoperation des Eingaberasters resultieren, mit den gewünschten Dimensionen aus.

Wenn `padwithnodata = FALSE`, dann können die Eckkacheln auf der rechten Seite und auf der unteren Seite andere Dimensionen als die übrigen Kacheln aufweisen. Wenn `padwithnodata = TRUE`, dann haben alle Kacheln dieselbe Dimension und die Eckkacheln werden eventuell mit NODATA Werten aufgefüllt. Wenn für die Rasterbänder keine NODATA Werte festgelegt sind, können Sie diese mit `nodataval` spezifizieren.



Note

Wenn das Band des Eingaberasters "out-of-db" ist, dann ist das entsprechende Band des Ausgaberrasters auch "out-of-db".

Verfügbarkeit: 2.1.0

Beispiele

```
WITH foo AS (
  SELECT ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(3, 3, 0, 0, 1, -1, 0, 0, 0), 1, '8 ←
    BUI', 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, '8 ←
    BUI', 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, '8 ←
    BUI', 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, '8 ←
    BUI', 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, '8 ←
    BUI', 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, '8 ←
    BUI', 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, '8 ←
    BUI', 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, '8 ←
    BUI', 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, '8 ←
    BUI', 9, 0), 2, '8BUI', 90, 0) AS rast
), bar AS (
  SELECT ST_Union(rast) AS rast FROM foo
), baz AS (
  SELECT ST_Tile(rast, 3, 3, TRUE) AS rast FROM bar
)
SELECT
  ST_DumpValues(rast)
FROM baz;

      st_dumpvalues
-----
```

```
(1,"{{1,1,1},{1,1,1},{1,1,1}}")
(2,"{{10,10,10},{10,10,10},{10,10,10}}")
(1,"{{2,2,2},{2,2,2},{2,2,2}}")
(2,"{{20,20,20},{20,20,20},{20,20,20}}")
(1,"{{3,3,3},{3,3,3},{3,3,3}}")
(2,"{{30,30,30},{30,30,30},{30,30,30}}")
(1,"{{4,4,4},{4,4,4},{4,4,4}}")
(2,"{{40,40,40},{40,40,40},{40,40,40}}")
(1,"{{5,5,5},{5,5,5},{5,5,5}}")
(2,"{{50,50,50},{50,50,50},{50,50,50}}")
(1,"{{6,6,6},{6,6,6},{6,6,6}}")
(2,"{{60,60,60},{60,60,60},{60,60,60}}")
(1,"{{7,7,7},{7,7,7},{7,7,7}}")
(2,"{{70,70,70},{70,70,70},{70,70,70}}")
(1,"{{8,8,8},{8,8,8},{8,8,8}}")
(2,"{{80,80,80},{80,80,80},{80,80,80}}")
(1,"{{9,9,9},{9,9,9},{9,9,9}}")
(2,"{{90,90,90},{90,90,90},{90,90,90}}")
(18 rows)
```

```
WITH foo AS (
    SELECT ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(3, 3, 0, 0, 1, -1, 0, 0, 0), 1, '8 ←
        BUI', 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, '8 ←
        BUI', 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, '8 ←
        BUI', 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, '8 ←
        BUI', 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, '8 ←
        BUI', 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, '8 ←
        BUI', 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, '8 ←
        BUI', 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, '8 ←
        BUI', 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, '8 ←
        BUI', 9, 0), 2, '8BUI', 90, 0) AS rast
), bar AS (
    SELECT ST_Union(rast) AS rast FROM foo
), baz AS (
    SELECT ST_Tile(rast, 3, 3, 2) AS rast FROM bar
)
SELECT
    ST_DumpValues(rast)
FROM baz;
```

```
st_dumpvalues
```

```
-----
(1,"{{10,10,10},{10,10,10},{10,10,10}}")
(1,"{{20,20,20},{20,20,20},{20,20,20}}")
(1,"{{30,30,30},{30,30,30},{30,30,30}}")
(1,"{{40,40,40},{40,40,40},{40,40,40}}")
(1,"{{50,50,50},{50,50,50},{50,50,50}}")
(1,"{{60,60,60},{60,60,60},{60,60,60}}")
(1,"{{70,70,70},{70,70,70},{70,70,70}}")
(1,"{{80,80,80},{80,80,80},{80,80,80}}")
(1,"{{90,90,90},{90,90,90},{90,90,90}}")
```

```
(9 rows)
```

Siehe auch

[ST_Union](#), [ST_Retile](#)

9.3.7 ST_Retile

ST_Retile — Gibt konfigurierte Kacheln eines beliebig gekachelten Rastercoverage aus.

Synopsis

SETOF raster **ST_Retile**(regclass tab, name col, geometry ext, float8 sfx, float8 sfy, int tw, int th, text algo='NearestNeighbor');

Beschreibung

Gibt die Kacheln in einem bestimmten Maßstab (*sfx*, *sfy*), maximaler Größe (*tw*, *th*) und räumlicher Ausdehnung (*ext*) mit den Daten des angegebenen Raster-Coverages (*tab*, *col*) zurück.

Die Optionen für den Algorithmus sind: 'NearestNeighbor', 'Bilinear', 'Cubic', 'CubicSpline' und 'Lanczos'. Siehe [GDAL Warp resampling methods](#) für weitere Details.

Verfügbarkeit: 2.2.0

Siehe auch

[ST_CreateOverview](#)

9.3.8 ST_FromGDALRaster

ST_FromGDALRaster — Erzeugt einen Raster aus einer von GDAL unterstützten Rasterdatei.

Synopsis

raster **ST_FromGDALRaster**(bytea gdaldata, integer srid=NULL);

Beschreibung

Erzeugt einen Raster aus einer von GDAL unterstützten Rasterdatei. *gdaldata* hat den Datentyp BYTEA und den Inhalt der GDAL Rasterdatei.

Wenn *srid* NULL ist, dann versucht die Funktion die SRID aus dem GDAL Raster automatisch zuzuweisen. Wenn *srid* angegeben ist, überschreibt dieser Wert jede automatisch zugewiesene SRID.

Verfügbarkeit: 2.1.0

Beispiele

```
WITH foo AS (
    SELECT ST_AsPNG(ST_AddBand(ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(2, 2, 0, 0, ←
        0.1, -0.1, 0, 0, 4326), 1, '8BUI', 1, 0), 2, '8BUI', 2, 0), 3, '8BUI', 3, 0)) AS ←
        png
),
bar AS (
    SELECT 1 AS rid, ST_FromGDALRaster(png) AS rast FROM foo
    UNION ALL
    SELECT 2 AS rid, ST_FromGDALRaster(png, 3310) AS rast FROM foo
)
SELECT
    rid,
    ST_Metadata(rast) AS metadata,
    ST_SummaryStats(rast, 1) AS stats1,
    ST_SummaryStats(rast, 2) AS stats2,
    ST_SummaryStats(rast, 3) AS stats3
FROM bar
ORDER BY rid;
```

rid	metadata	stats1	stats2	stats3
1	(0,0,2,2,1,-1,0,0,0,3)	(4,4,1,0,1,1)	(4,8,2,0,2,2)	(4,12,3,0,3,3)
2	(0,0,2,2,1,-1,0,0,3310,3)	(4,4,1,0,1,1)	(4,8,2,0,2,2)	(4,12,3,0,3,3)

(2 rows)

Siehe auch

[ST_AsGDALRaster](#)

9.4 Zugriffsfunktionen auf Raster

9.4.1 ST_GeoReference

ST_GeoReference — Gibt die Metadaten der Georeferenzierung, die sich üblicherweise in einem sogenannten "World File befinden, im GDAL oder ESRI Format aus. Die Standardeinstellung ist GDAL.

Synopsis

```
text ST_GeoReference(raster rast, text format=GDAL);
```

Beschreibung

Gibt die Metadaten der Georeferenzierung, die sich üblicherweise in einem sogenannten "World File befinden, inklusive "Carriage Return" im GDAL oder ESRI Format aus. Die Standardeinstellung ist GDAL. Der Datentyp ist die Zeichenfolge 'GDAL' oder 'ESRI'.

Die Formate unterscheiden sich wie folgt:

GDAL:

```
scalex
skewy
skewx
scaley
```

```
upperleftx
upperlefty
```

ESRI:

```
scalex
skewy
skewx
scaley
upperleftx + scalex*0.5
upperlefty + scaley*0.5
```

Beispiele

```
SELECT ST_GeoReference(rast, 'ESRI') As esri_ref, ST_GeoReference(rast, 'GDAL') As gdal_ref
FROM dummy_rast WHERE rid=1;
```

esri_ref		gdal_ref
2.0000000000		2.0000000000
0.0000000000	:	0.0000000000
0.0000000000	:	0.0000000000
3.0000000000	:	3.0000000000
1.5000000000	:	0.5000000000
2.0000000000	:	0.5000000000

Siehe auch

[ST_SetGeoReference](#), [ST_ScaleX](#), [ST_ScaleY](#)

9.4.2 ST_Height

ST_Height — Gibt die Höhe des Rasters in Pixel aus.

Synopsis

integer **ST_Height**(raster rast);

Beschreibung

Gibt die Höhe des Rasters aus.

Beispiele

```
SELECT rid, ST_Height(rast) As rastheight
FROM dummy_rast;
```

rid		rastheight
1		20
2		5

Siehe auch[ST_Width](#)**9.4.3 ST_IsEmpty**

ST_IsEmpty — Gibt TRUE zurück, wenn der Raster leer ist (width = 0 and height = 0). Andernfalls wird FALSE zurückgegeben.

Synopsis

boolean **ST_IsEmpty**(raster rast);

Beschreibung

Gibt TRUE zurück, wenn der Raster leer ist (width = 0 and height = 0). Andernfalls wird FALSE zurückgegeben.

Verfügbarkeit: 2.0.0

Beispiele

```
SELECT ST_IsEmpty(ST_MakeEmptyRaster(100, 100, 0, 0, 0, 0, 0, 0))
st_isempty |
-----+
f          |

SELECT ST_IsEmpty(ST_MakeEmptyRaster(0, 0, 0, 0, 0, 0, 0, 0))
st_isempty |
-----+
t          |
```

Siehe auch[ST_HasNoBand](#)**9.4.4 ST_MemSize**

ST_MemSize — Gibt den Platzbedarf des Rasters (in Byte) aus.

Synopsis

integer **ST_MemSize**(raster rast);

Beschreibung

Gibt den Platzbedarf des Rasters (in Byte) aus.

Dies Funktion ist eine schöne Ergänzung zu den in PostgreSQL eingebauten Funktionen `pg_column_size`, `pg_size_pretty`, `pg_relation_size` und `pg_total_relation_size`.



Note
pg_relation_size, das die Größe einer Tabelle in Byte angibt kann niedrigere Werte liefern als ST_MemSize. Dies kann vorkommen, da pg_relation_size den TOAST-Speicher der Tabellen nicht mitrechnet und eine große Geometrie in TOAST-Tabellen gespeichert wird. pg_column_size kann einen niedrigeren Wert anzeigen, da es die komprimierte Dateigröße ausgibt.
pg_total_relation_size - schließt die Tabelle, die TOAST-Tabellen und die Indizes mit ein.

Verfügbarkeit: 2.2.0

Beispiele

```
SELECT ST_MemSize(ST_AsRaster(ST_Buffer(ST_Point(1,5),10,1000),150, 150, '8BUI')) As rast_mem;

rast_mem
-----
22568
```

Siehe auch

9.4.5 ST_MetaData

ST_MetaData — Gibt die wesentlichen Metadaten eines Rasterobjektes, wie Zellgröße, Rotation (Versatz) etc. aus

Synopsis

record ST_MetaData(raster rast);

Beschreibung

Gibt die grundlegenden Metadaten eines Rasters aus, wie Pixelgröße, Rotation (skew), obere, untere linke, etc.. Die zurückgegebenen Spalten sind: upperleftx | upperlefty | width | height | scalex | scaley | skewx | skewy | srid | numbands

Beispiele

```
SELECT rid, (foo.md).*
FROM (SELECT rid, ST_MetaData(rast) As md
FROM dummy_rast) As foo;
```

rid	upperleftx	upperlefty	width	height	scalex	scaley	skewx	skewy	srid	numbands
1	0	0	0.5	0.5	10	20	2	3	0	
2	3427927.75	5793244	5	5	0.05	-0.05			0	

Siehe auch

ST_BandMetaData, ST_NumBands

9.4.6 ST_NumBands

ST_NumBands — Gibt die Anzahl der Bänder des Rasters aus.

Synopsis

integer **ST_NumBands**(raster rast);

Beschreibung

Gibt die Anzahl der Bänder des Rasters aus.

Beispiele

```
SELECT rid, ST_NumBands(rast) As numbands
FROM dummy_rast;
```

rid	numbands
1	0
2	3

Siehe auch

[ST_Value](#)

9.4.7 ST_PixelHeight

ST_PixelHeight — Gibt die Pixelhöhe in den Einheiten des Koordinatenreferenzsystem aus.

Synopsis

double precision **ST_PixelHeight**(raster rast);

Beschreibung

Gibt die Höhe eines Pixels in den Einheiten des Koordinatenreferenzsystem aus. Beim üblichen Fall ohne Versatz/skew, entspricht die Pixelhöhe dem Maßstabsverhältnis zwischen den geometrischen Koordinaten und den Rasterpixeln.

Siehe [ST_PixelWidth](#) für eine schematische Darstellung der Verhältnisse.

Beispiele: Raster ohne Versatz/skew

```
SELECT ST_Height(rast) As rastheight, ST_PixelHeight(rast) As pixheight,
       ST_ScaleX(rast) As scalex, ST_ScaleY(rast) As scaley, ST_SkewX(rast) As skewx,
       ST_SkewY(rast) As skewy
FROM dummy_rast;
```

rastheight	pixheight	scalex	scaley	skewx	skewy
20	3	2	3	0	0
5	0.05	0.05	-0.05	0	0

Beispiele: Raster mit einem Versatz ungleich 0

```
SELECT ST_Height(rast) As rastheight, ST_PixelHeight(rast) As pixheight,
       ST_ScaleX(rast) As scalex, ST_ScaleY(rast) As scaley, ST_SkewX(rast) As skewx,
       ST_SkewY(rast) As skewy
FROM (SELECT ST_SetSKew(rast,0.5,0.5) As rast
      FROM dummy_rast) As skewed;
```

rastheight	pixheight	scalex	scaley	skewx	skewy
20	3.04138126514911	2	3	0.5	0.5
5	0.502493781056044	0.05	-0.05	0.5	0.5

Siehe auch

[ST_PixelWidth](#), [ST_ScaleX](#), [ST_ScaleY](#), [ST_SkewX](#), [ST_SkewY](#)

9.4.8 ST_PixelWidth

ST_PixelWidth — Gibt die Pixelbreite in den Einheiten des Koordinatenreferenzsystems aus.

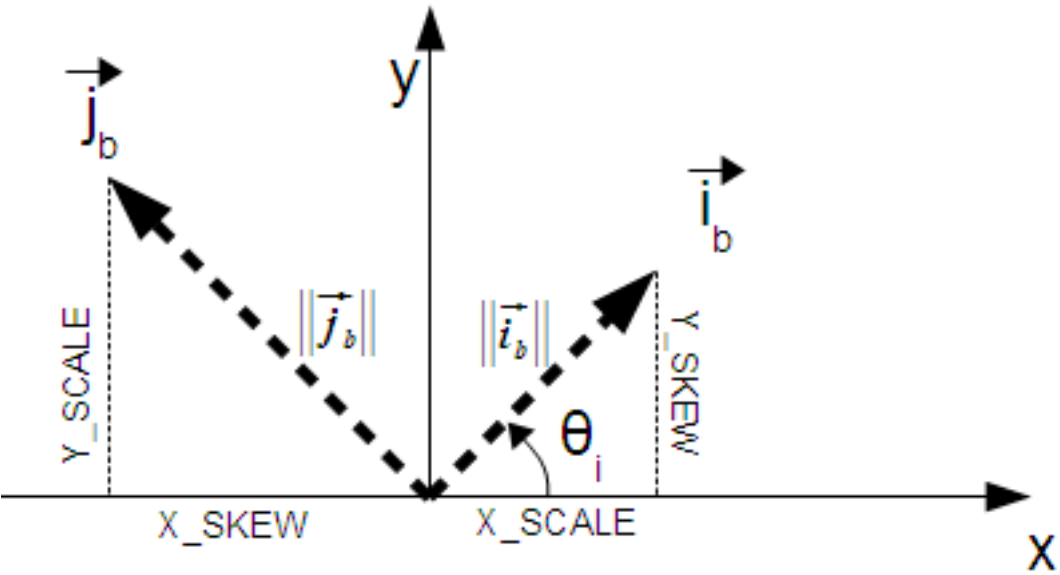
Synopsis

double precision **ST_PixelWidth**(raster rast);

Beschreibung

Gibt die Pixelbreite in den Einheiten des Koordinatenreferenzsystems aus. Beim üblichen Fall ohne Versatz entspricht die Pixelbreite dem Maßstabsverhältnis zwischen den geometrischen Koordinaten und den Rasterpixel.

Das folgende Schema zeigt die Beziehungen:



Pixelbreite: Pixelgröße in "i"-Richtung
Pixelhöhe: Pixelgröße in "j"-Richtung

Beispiele: Raster ohne Versatz/skew

```
SELECT ST_Width(rast) As rastwidth, ST_PixelWidth(rast) As pixwidth,
       ST_ScaleX(rast) As scalex, ST_ScaleY(rast) As scaley, ST_SkewX(rast) As skewx,
       ST_SkewY(rast) As skewy
FROM dummy_rast;
```

rastwidth	pixwidth	scalex	scaley	skewx	skewy
10	2	2	3	0	0
5	0.05	0.05	-0.05	0	0

Beispiele: Raster mit einem Versatz ungleich 0

```
SELECT ST_Width(rast) As rastwidth, ST_PixelWidth(rast) As pixwidth,
       ST_ScaleX(rast) As scalex, ST_ScaleY(rast) As scaley, ST_SkewX(rast) As skewx,
       ST_SkewY(rast) As skewy
FROM (SELECT ST_SetSkew(rast,0.5,0.5) As rast
FROM dummy_rast) As skewed;
```

rastwidth	pixwidth	scalex	scaley	skewx	skewy
10	2.06155281280883	2	3	0.5	0.5
5	0.502493781056044	0.05	-0.05	0.5	0.5

Siehe auch

[ST_PixelHeight](#), [ST_ScaleX](#), [ST_ScaleY](#), [ST_SkewX](#), [ST_SkewY](#)

9.4.9 ST_ScaleX

ST_ScaleX — Gibt die X-Komponente der Pixelbreite in den Einheiten des Koordinatenreferenzsystems aus.

Synopsis

```
float8 ST_ScaleX(raster rast);
```

Beschreibung

Gibt die X-Komponente der Pixelbreite in den Einheiten des Koordinatenreferenzsystems aus. Siehe [World File](#) für weitere Details.

Änderung: 2.0.0 In WKTRaster Versionen wurde dies als `ST_PixelSizeX` bezeichnet.

Beispiele

```
SELECT rid, ST_ScaleX(rast) As rastpixwidth
FROM dummy_rast;
```

rid	rastpixwidth
1	2
2	0.05

Siehe auch[ST_Width](#)**9.4.10 ST_ScaleY**

ST_ScaleY — Gibt die Y-Komponente der Pixelhöhe in den Einheiten des Koordinatenreferenzsystems aus.

Synopsis

```
float8 ST_ScaleY(raster rast);
```

Beschreibung

Gibt die Y-Komponente der Pixelhöhe in den Einheiten des Koordinatenreferenzsystems aus. Kann negative Werte annehmen. Siehe [World File](#) für weitere Details.

Änderung: 2.0.0. Versionen von WKTRaster haben dies als "ST_PixelSizeY" bezeichnet.

Beispiele

```
SELECT rid, ST_ScaleY(rast) As rastpixheight
FROM dummy_rast;
```

rid	rastpixheight
1	3
2	-0.05

Siehe auch[ST_Height](#)**9.4.11 ST_RasterToWorldCoord**

ST_RasterToWorldCoord — Gibt die obere linke Ecke des Rasters in geodätischem X und Y (Länge und Breite) für eine gegebene Spalte und Zeile aus. Spalte und Zeile wird von 1 aufwärts gezählt.

Synopsis

```
record ST_RasterToWorldCoord(raster rast, integer xcolumn, integer yrow);
```

Beschreibung

Gibt die obere linke Ecke einer Spalte und Zeile als geometrisches X und Y (als geographische Länge und Breite) aus. Die zurückgegebenen X und Y sind in den Einheiten des georeferenzierten Rasters. Die Nummerierung der Spalten und Zeilen beginnt mit 1. Allerdings, wenn für einen der Parameter 0, eine negative Zahl, oder eine höhere Zahl als die betreffende Größe des Rasters übergeben wird, dann werden Koordinaten außerhalb des Rasters ausgegeben; dabei wird angenommen, dass das Gitter des Rasters auch außerhalb der Begrenzung des Rasters angewendet werden kann.

Verfügbarkeit: 2.1.0

Beispiele

```
-- Raster ohne Versatz/skew
SELECT
    rid,
    (ST_RasterToWorldCoord(rast,1, 1)).*,
    (ST_RasterToWorldCoord(rast,2, 2)).*
FROM dummy_rast
```

rid	longitude	latitude	longitude	latitude
1	0.5	0.5	2.5	3.5
2	3427927.75	5793244	3427927.8	5793243.95

```
-- Raster mit Versatz/skew
SELECT
    rid,
    (ST_RasterToWorldCoord(rast, 1, 1)).*,
    (ST_RasterToWorldCoord(rast, 2, 3)).*
FROM (
    SELECT
        rid,
        ST_SetSkew(rast, 100.5, 0) As rast
    FROM dummy_rast
) As foo
```

rid	longitude	latitude	longitude	latitude
1	0.5	0.5	203.5	6.5
2	3427927.75	5793244	3428128.8	5793243.9

Siehe auch

[ST_RasterToWorldCoordX](#), [ST_RasterToWorldCoordY](#), [ST_SetSkew](#)

9.4.12 ST_RasterToWorldCoordX

ST_RasterToWorldCoordX — Gibt die geodätische X Koordinate links oberhalb des Rasters, der Spalte und der Zeile aus. Die Nummerierung der Spalten und Zeilen beginnt mit 1.

Synopsis

```
float8 ST_RasterToWorldCoordX(raster rast, integer xcolumn);
float8 ST_RasterToWorldCoordX(raster rast, integer xcolumn, integer yrow);
```

Beschreibung

Gibt die obere linke X-Koordinate einer Rasterzeile in den geometrischen Einheiten des georeferenzierten Rasters aus. Die Nummerierung der Spalten und Zeilen beginnt mit 1. Wenn Sie eine negative Zahl oder eine höhere Zahl als die Anzahl der Spalten des Rasters angeben, dann bekommen Sie die Koordinaten links außerhalb und rechts außerhalb des Rasters zurück; dabei wird angenommen, dass der Versatz und die Pixelgröße gleich wie bei dem ausgewählten Raster sind.



Note

Bei Rastern ohne Versatz, ist die Angabe der X-Spalte ausreichend. Bei Rastern mit Versatz ist die georeferenzierte Koordinate eine Funktion von "ST_ScaleX", "ST_SkewX", der Zeile und der Spalte. Wenn Sie bei einem Raster mit Versatz nur die X-Spalte angeben, dann wird eine Fehlermeldung ausgegeben.

Änderung: 2.1.0 Vorgängerversionen haben dies als "ST_Raster2WorldCoordX" bezeichnet.

Beispiele

```
-- Bei einem Raster ohne Versatz ist die Angabe der Spalte ausreichend
SELECT rid, ST_RasterToWorldCoordX(rast,1) As xlcoord,
        ST_RasterToWorldCoordX(rast,2) As x2coord,
        ST_ScaleX(rast) As pixelx
FROM dummy_rast;
```

rid	xlcoord	x2coord	pixelx
1	0.5	2.5	2
2	3427927.75	3427927.8	0.05

```
-- Lasst es Uns nur so zum Spass drehen
SELECT rid, ST_RasterToWorldCoordX(rast, 1, 1) As xlcoord,
        ST_RasterToWorldCoordX(rast, 2, 3) As x2coord,
        ST_ScaleX(rast) As pixelx
FROM (SELECT rid, ST_SetSkew(rast, 100.5, 0) As rast FROM dummy_rast) As foo;
```

rid	xlcoord	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](#)

9.4.13 ST_RasterToWorldCoordY

ST_RasterToWorldCoordY — Gibt die geodätische Y Koordinate links oberhalb des Rasters, der Spalte und der Zeile aus. Die Nummerierung der Spalten und Zeilen beginnt mit 1.

Synopsis

```
float8 ST_RasterToWorldCoordY(raster rast, integer yrow);
float8 ST_RasterToWorldCoordY(raster rast, integer xcolumn, integer yrow);
```

Beschreibung

Gibt die obere linke Y-Koordinate einer Rasterspalte in den Einheiten des georeferenzierten Rasters aus. Die Nummerierung der Spalten und Zeilen beginnt mit 1. Wenn Sie eine negative Zahl oder eine höhere Zahl als die Anzahl der Spalten/Zeilen des Rasters angeben, dann bekommen Sie die Koordinaten außerhalb des Rasters zurück; dabei wird angenommen, dass Versatz und Pixelgröße gleich wie bei der ausgewählten Rasterkachel sind.



Note

Bei Rastern ohne Versatz ist die Angabe der Y-Spalte ausreichend. Bei Rastern mit Versatz entspricht die georeferenzierte Koordinate einer Funktion von ST_ScaleY, ST_SkewY, Zeile und Spalte. Wenn Sie bei einem Raster mit Versatz nur die Y-Spalte angeben, wird eine Fehlermeldung ausgegeben.

Änderung: 2.1.0 In Vorgängerversionen wurde dies als ST_Raster2WorldCoordY bezeichnet

Beispiele

```
-- Bei einem Raster ohne Versatz ist die Angabe der Zeile ausreichend
SELECT rid, ST_RasterToWorldCoordY(rast,1) As ylcoord,
       ST_RasterToWorldCoordY(rast,3) As y2coord,
       ST_ScaleY(rast) As pixely
FROM dummy_rast;
```

rid	ylcoord	y2coord	pixely
1	0.5	6.5	3
2	5793244	5793243.9	-0.05

```
-- zum Spaß führen wir einen Versatz ein
SELECT rid, ST_RasterToWorldCoordY(rast,1,1) As ylcoord,
       ST_RasterToWorldCoordY(rast,2,3) As y2coord,
       ST_ScaleY(rast) As pixely
FROM (SELECT rid, ST_SetSkew(rast,0,100.5) As rast FROM dummy_rast) As foo;
```

rid	ylcoord	y2coord	pixely
1	0.5	107	3
2	5793244	5793344.4	-0.05

Siehe auch

[ST_ScaleY](#), [ST_RasterToWorldCoordX](#), [ST_SetSkew](#), [ST_SkewY](#)

9.4.14 ST_Rotation

ST_Rotation — Gibt die Rotation des Rasters im Bogenmaß aus.

Synopsis

```
float8 ST_Rotation(raster rast);
```

Beschreibung

Gibt die einheitliche Rotation des Rasters in Radiant aus. Wenn der Raster keine einheitliche Rotation aufweist wird NaN zurückgegeben. Siehe [World File](#) für weitere Details.

Beispiele

```
SELECT rid, ST_Rotation(ST_SetScale(ST_SetSkew(rast, sqrt(2)), sqrt(2))) as rot FROM dummy_rast;
```

rid	rot
1	0.785398163397448
2	0.785398163397448

Siehe auch

[ST_SetRotation](#), [ST_SetScale](#), [ST_SetSkew](#)

9.4.15 ST_SkewX

ST_SkewX — Gibt den georeferenzierten Versatz in X-Richtung (oder den Rotationsparameter) aus.

Synopsis

float8 ST_SkewX(raster rast);

Beschreibung

Gibt den georeferenzierten Versatz in X-Richtung (oder den Rotationsparameter) aus. Siehe [World File](#) für weitere Details.

Beispiele

```
SELECT rid, ST_SkewX(rast) As skewx, ST_SkewY(rast) As skewy,
       ST_GeoReference(rast) as georef
FROM dummy_rast;
```

rid	skewx	skewy	georef
1	0	0	2.0000000000 : 0.0000000000 : 0.0000000000 : 3.0000000000 : 0.5000000000 : 0.5000000000 :
2	0	0	0.0500000000 : 0.0000000000 : 0.0000000000 : -0.0500000000 : 3427927.7500000000 : 5793244.0000000000

Siehe auch

[ST_GeoReference](#), [ST_SkewY](#), [ST_SetSkew](#)

9.4.16 ST_SkewY

ST_SkewY — Gibt den georeferenzierten Versatz in Y-Richtung (oder den Rotationsparameter) aus.

Synopsis

float8 ST_SkewY(raster rast);

Beschreibung

Gibt den georeferenzierten Versatz in Y-Richtung (oder den Rotationsparameter) aus. Siehe [World File](#) für weitere Details.

Beispiele

```
SELECT rid, ST_SkewX(rast) As skewx, ST_SkewY(rast) As skewy,
       ST_GeoReference(rast) as georef
FROM dummy_rast;
```

rid	skewx	skewy	georef
1	0	0	2.0000000000 : 0.0000000000 : 0.0000000000 : 3.0000000000 : 0.5000000000 : 0.5000000000 :
2	0	0	0.0500000000 : 0.0000000000 : 0.0000000000 : -0.0500000000 : 3427927.7500000000 : 5793244.0000000000

Siehe auch

[ST_GeoReference](#), [ST_SkewX](#), [ST_SetSkew](#)

9.4.17 ST_SRID

ST_SRID — Gibt den Identifikator des Koordinatenreferenzsystems des Rasters aus, das in der Tabelle "spatial_ref_sys" definiert ist.

Synopsis

```
integer ST_SRID(raster rast);
```

Beschreibung

Gibt den Identifikator des Koordinatenreferenzsystems des Rasters aus, das in der Tabelle "spatial_ref_sys" definiert ist.



Note
Ab PostGIS 2.0 ist die SRID eines nicht georeferenzierten Rasters/Geometrie 0 anstelle wie früher -1.

Beispiele

```
SELECT ST_SRID(rast) As srid
FROM dummy_rast WHERE rid=1;
```

srid
0

Siehe auch

Section [4.3.1](#), [ST_SRID](#)

9.4.18 ST_Summary

ST_Summary — Gibt eine textliche Zusammenfassung des Rasterinhalts zurück.

Synopsis

text **ST_Summary**(raster rast);

Beschreibung

Gibt eine textliche Zusammenfassung des Rasterinhalts zurück

Verfügbarkeit: 2.1.0

Beispiele

```
SELECT ST_Summary(
    ST_AddBand(
        ST_AddBand(
            ST_AddBand(
                ST_MakeEmptyRaster(10, 10, 0, 0, 1, -1, 0, 0, 0)
                , 1, '8BUI', 1, 0
            )
            , 2, '32BF', 0, -9999
        )
        , 3, '16BSI', 0, NULL
    )
);
```

st_summary

```
-----
Raster of 10x10 pixels has 3 bands and extent of BOX(0 -10,10 0)+
band 1 of pixtype 8BUI is in-db with NODATA value of 0      +
band 2 of pixtype 32BF is in-db with NODATA value of -9999  +
band 3 of pixtype 16BSI is in-db with no NODATA value
(1 row)
```

Siehe auch

[ST_MetaData](#), [ST_BandMetaData](#), [ST_Summary](#) [ST_Extent](#)

9.4.19 ST_UpperLeftX

ST_UpperLeftX — Gibt die obere linke X-Koordinate des Rasters im Koordinatenprojektionssystem aus.

Synopsis

float8 **ST_UpperLeftX**(raster rast);

Beschreibung

Gibt die obere linke X-Koordinate des Rasters im Koordinatenprojektionssystem aus.

Beispiele

```
SELECT rid, ST_UpperLeftX(rast) As ulx
FROM dummy_rast;
```

rid	ulx
1	0.5
2	3427927.75

Siehe auch

[ST_UpperLeftY](#), [ST_GeoReference](#), [Box3D](#)

9.4.20 ST_UpperLeftY

ST_UpperLeftY — Gibt die obere linke Y-Koordinate des Rasters im Koordinatenprojektionssystem aus.

Synopsis

float8 **ST_UpperLeftY**(raster rast);

Beschreibung

Gibt die obere linke Y-Koordinate des Rasters im Koordinatenprojektionssystem aus.

Beispiele

```
SELECT rid, ST_UpperLeftY(rast) As uly
FROM dummy_rast;
```

rid	uly
1	0.5
2	5793244

Siehe auch

[ST_UpperLeftX](#), [ST_GeoReference](#), [Box3D](#)

9.4.21 ST_Width

ST_Width — Gibt die Breite des Rasters in Pixel aus.

Synopsis

integer **ST_Width**(raster rast);

Beschreibung

Gibt die Breite des Rasters in Pixel aus.

Beispiele

```
SELECT ST_Width(rast) As rastwidth
FROM dummy_rast WHERE rid=1;

rastwidth
-----
10
```

Siehe auch

[ST_Height](#)

9.4.22 ST_WorldToRasterCoord

ST_WorldToRasterCoord — Gibt für ein geometrisches X und Y (geographische Länge und Breite) oder für eine Punktgeometrie im Koordinatenreferenzsystem des Rasters, die obere linke Ecke als Spalte und Zeile aus.

Synopsis

```
record ST_WorldToRasterCoord(raster rast, geometry pt);
record ST_WorldToRasterCoord(raster rast, double precision longitude, double precision latitude);
```

Beschreibung

Gibt für ein geometrisches X und Y (geographische Länge und Breite), oder für eine Punktgeometrie, die obere linke Ecke als Spalte und Zeile aus. Diese Funktion funktioniert auch dann, wenn sich X und Y, bzw. die Punktgeometrie außerhalb der Ausdehnung des Rasters befindet. X und Y müssen im Koordinatenreferenzsystem des Rasters angegeben werden.

Verfügbarkeit: 2.1.0

Beispiele

```
SELECT
    rid,
    (ST_WorldToRasterCoord(rast, 3427927.8, 20.5)) .*,
    (ST_WorldToRasterCoord(rast, ST_GeomFromText('POINT(3427927.8 20.5)', ST_SRID(rast))) ↔
    ) .*
FROM dummy_rast;
```

rid	columnx	rowy	columnx	rowy
1	1713964	7	1713964	7
2	2	115864471	2	115864471

Siehe auch

[ST_WorldToRasterCoordX](#), [ST_WorldToRasterCoordY](#), [ST_RasterToWorldCoordX](#), [ST_RasterToWorldCoordY](#), [ST_SRID](#)

9.4.23 ST_WorldToRasterCoordX

ST_WorldToRasterCoordX — Gibt für eine Punktgeometrie (pt) oder eine globale X- und Y-Koordinate (xw, yw) die Rasterspalte im globalen Koordinatenreferenzsystem des Rasters aus.

Synopsis

```
integer ST_WorldToRasterCoordX(raster rast, geometry pt);
integer ST_WorldToRasterCoordX(raster rast, double precision xw);
integer ST_WorldToRasterCoordX(raster rast, double precision xw, double precision yw);
```

Beschreibung

Gibt für eine Punktgeometrie (pt) oder eine globale X- und Y-Koordinate (xw, yw) die Rasterspalte aus. Es werden ein Punkt oder sowohl "xw" als auch "yw" in globalen Koordinaten benötigt, wenn der Raster einen Versatz aufweist. Wenn der Raster keinen Versatz aufweist, ist "xw" ausreichend. Die globalen Koordinaten sind im Koordinatenreferenzsystem des Rasters.

Änderung: 2.1.0 In Vorgängerversionen wurde dies als ST_World2RasterCoordX bezeichnet

Beispiele

```
SELECT rid, ST_WorldToRasterCoordX(rast,3427927.8) As xcoord,
          ST_WorldToRasterCoordX(rast,3427927.8,20.5) As xcoord_xwyw,
          ST_WorldToRasterCoordX(rast,ST_GeomFromText('POINT(3427927.8 20.5)',ST_SRID ←
          (rast))) As ptxcoord
FROM dummy_rast;
```

rid	xcoord	xcoord_xwyw	ptxcoord
1	1713964	1713964	1713964
2	1	1	1

Siehe auch

[ST_RasterToWorldCoordX](#), [ST_RasterToWorldCoordY](#), [ST_SRID](#)

9.4.24 ST_WorldToRasterCoordY

ST_WorldToRasterCoordY — Gibt für eine Punktgeometrie (pt) oder eine globale X- und Y-Koordinate (xw, yw) die Rasterzeile im globalen Koordinatenreferenzsystem des Rasters aus.

Synopsis

```
integer ST_WorldToRasterCoordY(raster rast, geometry pt);
integer ST_WorldToRasterCoordY(raster rast, double precision xw);
integer ST_WorldToRasterCoordY(raster rast, double precision xw, double precision yw);
```

Beschreibung

Gibt für eine Punktgeometrie (pt) oder eine globale X- und Y-Koordinate (xw, yw) die Rasterzeile aus. Es werden ein Punkt oder sowohl "xw" als auch "yw" in globalen Koordinaten benötigt, wenn der Raster einen Versatz aufweist. Wenn der Raster keinen Versatz aufweist, ist "xw" ausreichend. Die globalen Koordinaten sind im Koordinatenreferenzsystem des Rasters.

Änderung: 2.1.0 In Vorgängerversionen wurde dies als ST_World2RasterCoordY bezeichnet

Beispiele

```
SELECT rid, ST_WorldToRasterCoordY(rast,20.5) As ycoord,
        ST_WorldToRasterCoordY(rast,3427927.8,20.5) As ycoord_xwyw,
        ST_WorldToRasterCoordY(rast,ST_GeomFromText('POINT(3427927.8 20.5)',ST_SRID ←
        (rast))) As ptycoord
FROM dummy_rast;
```

rid	ycoord	ycoord_xwyw	ptycoord
1	7	7	7
2	115864471	115864471	115864471

Siehe auch

[ST_RasterToWorldCoordX](#), [ST_RasterToWorldCoordY](#), [ST_SRID](#)

9.5 Zugriffsfunktionen auf Rasterbänder

9.5.1 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

record **ST_BandMetaData**(raster rast, integer bandnum=1);

Beschreibung

Gibt die grundlegenden Metadaten eines Rasterbandes aus. Die zurückgegebenen Spalten sind pixeltype | nodatavalue | isoutdb | path.



Note

Wenn der Raster keine Bänder beinhaltet wird ein Fehler gemeldet.



Note

Wenn für das Band kein NODATA-Wert vergeben wurde, wird der NODATA-Wert auf NULL gesetzt.

Beispiele

```
SELECT rid, (foo.md).*
FROM (SELECT rid, ST_BandMetaData(rast,1) As md
FROM dummy_rast WHERE rid=2) As foo;
```

rid	pixeltype	nodatavalue	isoutdb	path
2	8BUI	0	f	

Siehe auch

[ST_MetaData](#), [ST_BandPixelType](#)

9.5.2 ST_BandNoDataValue

ST_BandNoDataValue — Gibt den NODATA Wert des gegebenen Bandes aus. Wenn der Parameter "bandnum" nicht angegeben ist, wird das 1ste Band angenommen.

Synopsis

double precision **ST_BandNoDataValue**(raster rast, integer bandnum=1);

Beschreibung

Gibt den NODATA Wert des Bandes aus.

Beispiele

```
SELECT ST_BandNoDataValue(rast,1) As bnval1,
       ST_BandNoDataValue(rast,2) As bnval2, ST_BandNoDataValue(rast,3) As bnval3
FROM dummy_rast
WHERE rid = 2;
```

bnval1	bnval2	bnval3
0	0	0

Siehe auch

[ST_NumBands](#)

9.5.3 ST_BandIsNoData

ST_BandIsNoData — Gibt TRUE aus, wenn das Band ausschließlich aus NODATA Werten besteht.

Synopsis

boolean **ST_BandIsNoData**(raster rast, integer band, boolean forceChecking=true);
 boolean **ST_BandIsNoData**(raster rast, boolean forceChecking=true);

Beschreibung

Gibt TRUE aus, wenn das Band ausschließlich aus NODATA Werten besteht. Wenn kein Band angegeben ist, wird Band 1 angenommen. Wenn der letzte Übergabewert TRUE ist, dann wird das gesamte Band, Pixel für Pixel, überprüft. Sonst gibt die Funktion nur den Wert der Flag "isnodata" für das Band aus. Wenn dieser Parameter nicht gesetzt wurde, wird der Standardwert "FALSE" angenommen.

Verfügbarkeit: 2.0.0

**Note**

Wenn die Flag geändert wurde (d.h.: das Ergebnis unterscheidet sich wenn man TRUE als letzten Parameter angibt, von jenem bei dem dieser Parameter nicht verwendet wird), sollten Sie den Raster aktualisieren um diese Flag auf TRUE zu setzen, indem Sie `ST_SetBandIsNodata()` anwenden, oder `ST_SetBandNodataValue()` mit TRUE als letzten Übergabewert ausführen. Siehe [ST_SetBandIsNoData](#).

Beispiele

```
-- Erstellt eine Dummy-Tabelle mit einer Rasterspalte
create table dummy_rast (rid integer, rast raster);

-- Einen Raster mit zwei Bändern hinzufügen, ein Pixel/Band. Beim ersten Band - NODATA Wert ←
  = Pixelwert = 3.
-- Beim zweiten Band - NODATA Wert = 1, Pixelwert = 4
insert into dummy_rast values(1,
(
'01' -- little endian (uint8 ndr)
||
'0000' -- version (uint16 0)
||
'0200' -- nBands (uint16 0)
||
'17263529ED684A3F' -- scaleX (float64 0.000805965234044584)
||
'F9253529ED684ABF' -- scaleY (float64 -0.00080596523404458)
||
'1C9F33CE69E352C0' -- ipX (float64 -75.5533328537098)
||
'718F0E9A27A44840' -- ipY (float64 49.2824585505576)
||
'ED50EB853EC32B3F' -- skewX (float64 0.000211812383858707)
||
'7550EB853EC32B3F' -- skewY (float64 0.000211812383858704)
||
'E6100000' -- SRID (int32 4326)
||
'0100' -- width (uint16 1)
||
'0100' -- height (uint16 1)
||
'6' -- hasnodatavalue and isnodata value set to true.
||
'2' -- first band type (4BUI)
||
'03' -- novalue==3
||
'03' -- pixel(0,0)==3 (same that nodata)
||
'0' -- hasnodatavalue set to false
||
'5' -- second band type (16BSI)
||
'0D00' -- novalue==13
||
'0400' -- pixel(0,0)==4
)::raster
);

select st_bandisnodata(rast, 1) from dummy_rast where rid = 1; -- Expected true
```

```
select st_bandisnodata(rast, 2) from dummy_rast where rid = 1; -- Expected false
```

Siehe auch

[ST_BandNoDataValue](#), [ST_NumBands](#), [ST_SetBandNoDataValue](#), [ST_SetBandIsNoData](#)

9.5.4 ST_BandPath

ST_BandPath — Gibt den Dateipfad aus, unter dem das Band im Dateisystem gespeichert ist. Wenn "bandnum" nicht angegeben ist, wird 1 angenommen.

Synopsis

text **ST_BandPath**(raster rast, integer bandnum=1);

Beschreibung

Gibt den Dateipfad des Bandes im System aus. Wenn man die Funktion mit einem "in-db"-Rasterband aufruft, wird eine Fehlermeldung ausgegeben.

Beispiele

Siehe auch

9.5.5 ST_BandPixelType

ST_BandPixelType — Gibt den Pixeltyp des angegebenen Bandes aus. Wenn der Parameter "bandnum" nicht angegeben ist, wird das 1ste Band angenommen.

Synopsis

text **ST_BandPixelType**(raster rast, integer bandnum=1);

Beschreibung

Gibt den NODATA Wert des Bandes aus.

Im Folgenden die 11 unterstützten Pixeltypen

- 1BB - 1-Bit Boolean
- 2BUI - 2-Bit vorzeichenlose Ganzzahl
- 4BUI - 4-Bit vorzeichenlose Ganzzahl
- 8BSI - 8-Bit Ganzzahl
- 8BUI - 8-Bit vorzeichenlose Ganzzahl
- 16BSI - 16-Bit Ganzzahl

- 16BUI - 16-bit vorzeichenlose Ganzzahl
- 32BSI - 32-Bit Ganzzahl
- 32BUI - 32-Bit vorzeichenlose Ganzzahl
- 32BF - 32-Bit Float
- 64BF - 64-Bit Float

Beispiele

```
SELECT ST_BandPixelType(rast,1) As btype1,
       ST_BandPixelType(rast,2) As btype2, ST_BandPixelType(rast,3) As btype3
FROM dummy_rast
WHERE rid = 2;
```

btype1	btype2	btype3
8BUI	8BUI	8BUI

Siehe auch

[ST_NumBands](#)

9.5.6 ST_HasNoBand

ST_HasNoBand — Gibt TRUE aus, wenn kein Band mit der angegebenen Bandnummer existiert. Gibt den Pixeltyp des angegebenen Bandes aus. Wenn keine Bandnummer angegeben ist, wird das 1ste Band angenommen.

Synopsis

boolean **ST_HasNoBand**(raster rast, integer bandnum=1);

Beschreibung

Gibt TRUE aus, wenn kein Band mit der angegebenen Bandnummer existiert. Gibt den Pixeltyp des angegebenen Bandes aus. Wenn keine Bandnummer angegeben ist, wird das 1ste Band angenommen.

Verfügbarkeit: 2.0.0

Beispiele

```
SELECT rid, ST_HasNoBand(rast) As hb1, ST_HasNoBand(rast,2) as hb2,
       ST_HasNoBand(rast,4) as hb4, ST_NumBands(rast) As numbands
FROM dummy_rast;
```

rid	hb1	hb2	hb4	numbands
1	t	t	t	0
2	f	f	t	3

Siehe auch

[ST_NumBands](#)

9.6 Zugriffsfunktionen und Änderungsmethoden für Rasterpixel

9.6.1 ST_PixelAsPolygon

ST_PixelAsPolygon — Gibt die Polygoneometrie aus, die das Pixel einer bestimmten Zeile und Spalte begrenzt.

Synopsis

geometry **ST_PixelAsPolygon**(raster rast, integer columnx, integer rowy);

Beschreibung

Gibt die Polygoneometrie aus, die das Pixel einer bestimmten Zeile und Spalte begrenzt.

Verfügbarkeit: 2.0.0

Beispiele

```
-- die Polygone der Rasterpixel ausgeben
SELECT i,j, ST_AsText(ST_PixelAsPolygon(foo.rast, i,j)) As blpgeom
FROM dummy_rast As foo
      CROSS JOIN generate_series(1,2) As i
      CROSS JOIN generate_series(1,1) As j
WHERE rid=2;
```

i	j	blpgeom
1	1	POLYGON((3427927.75 5793244,3427927.8 5793244,3427927.8 5793243.95,...
2	1	POLYGON((3427927.8 5793244,3427927.85 5793244,3427927.85 5793243.95, ..

Siehe auch

[ST_DumpAsPolygons](#), [ST_PixelAsPolygons](#), [ST_PixelAsPoint](#), [ST_PixelAsPoints](#), [ST_PixelAsCentroid](#), [ST_PixelAsCentroids](#), [ST_Intersection](#), [ST_AsText](#)

9.6.2 ST_PixelAsPolygons

ST_PixelAsPolygons — Gibt die umhüllende Polygoneometrie, den Zellwert, sowie die X- und Y-Rasterkoordinate für jedes Pixel aus.

Synopsis

setof record **ST_PixelAsPolygons**(raster rast, integer band=1, boolean exclude_nodata_value=TRUE);

Beschreibung

Gibt die umhüllende Polygoneometrie, den Zellwert (double precision), sowie die X- und Y-Rasterkoordinate (Ganzzahl) für jedes Pixel aus.

**Note**

ST_PixelAsPolygons gibt eine Polygongeometrie pro Pixel zurück. Dies unterscheidet sich von ST_DumpAsPolygons, wo jede Geometrie einen oder mehrere Pixel mit gleichem Zellwert darstellt.

**Note**

Wenn exclude_nodata_value = TRUE, werden nur Pixel ohne NODATA Werte als Polygone ausgegeben.

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

```
-- ein Rasterpixelpolygon ausgeben
SELECT (gv).x, (gv).y, (gv).val, ST_AsText((gv).geom) geom
FROM (SELECT ST_PixelAsPolygons(
          ST_SetValue(ST_SetValue(ST_AddBand(ST_MakeEmptyRaster(2, 2, 0, 0, 0.001, ←
          -0.001, 0.001, 0.001, 4269),
                                '8BUI'::text, 1, 0),
                                2, 2, 10),
          1, 1, NULL)
) gv
) foo;
```

x	y	val	geom
1	1		POLYGON((0 0,0.001 0.001,0.002 0,0.001 -0.001,0 0))
1	2	1	POLYGON((0.001 -0.001,0.002 0,0.003 -0.001,0.002 -0.002,0.001 -0.001))
2	1	1	POLYGON((0.001 0.001,0.002 0.002,0.003 0.001,0.002 0,0.001 0.001))
2	2	10	POLYGON((0.002 0,0.003 0.001,0.004 0,0.003 -0.001,0.002 0))

Siehe auch

[ST_DumpAsPolygons](#), [ST_PixelAsPolygon](#), [ST_PixelAsPoint](#), [ST_PixelAsPoints](#), [ST_PixelAsCentroid](#), [ST_PixelAsCentroids](#), [ST_AsText](#)

9.6.3 ST_PixelAsPoint

ST_PixelAsPoint — Gibt eine Punktgeometrie der oberen linken Ecke des Rasters zurück.

Synopsis

geometry **ST_PixelAsPoint**(raster rast, integer columnx, integer rowy);

Beschreibung

Gibt eine Punktgeometrie der oberen linken Ecke des Rasters zurück.

Verfügbarkeit: 2.1.0

Beispiele

```
SELECT ST_AsText(ST_PixelAsPoint(rast, 1, 1)) FROM dummy_rast WHERE rid = 1;

  st_astext
-----
POINT(0.5 0.5)
```

Siehe auch

[ST_DumpAsPolygons](#), [ST_PixelAsPolygon](#), [ST_PixelAsPolygons](#), [ST_PixelAsPoints](#), [ST_PixelAsCentroid](#), [ST_PixelAsCentroids](#)

9.6.4 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

geometry **ST_PixelAsPoints**(raster rast, integer band=1, boolean exclude_nodata_value=TRUE);

Beschreibung

Gibt eine Punktgeometrie für jedes Pixel des Rasterbandes zurück, zusammen mit dem Zellwert und den X- und Y-Rasterkoordinaten eines jeden Pixels. Die Koordinaten der Punkte entsprechen dem oberen linken Eck der Pixel.



Note

Wenn `exclude_nodata_value = TRUE`, dann werden nur jene Pixel als Punkte zurückgegeben deren Zellwerte nicht NODATA sind.

Verfügbarkeit: 2.1.0

Änderung: 2.1.1 Die Verhaltensweise von "exclude_nodata_value" wurde geändert.

Beispiele

```
SELECT x, y, val, ST_AsText(geom) FROM (SELECT (ST_PixelAsPoints(rast, 1)).* FROM ↵
dummy_rast WHERE rid = 2) foo;
```

x	y	val	st_astext
1	1	253	POINT(3427927.75 5793244)
2	1	254	POINT(3427927.8 5793244)
3	1	253	POINT(3427927.85 5793244)
4	1	254	POINT(3427927.9 5793244)
5	1	254	POINT(3427927.95 5793244)
1	2	253	POINT(3427927.75 5793243.95)
2	2	254	POINT(3427927.8 5793243.95)
3	2	254	POINT(3427927.85 5793243.95)
4	2	253	POINT(3427927.9 5793243.95)
5	2	249	POINT(3427927.95 5793243.95)
1	3	250	POINT(3427927.75 5793243.9)
2	3	254	POINT(3427927.8 5793243.9)

```

3 | 3 | 254 | POINT(3427927.85 5793243.9)
4 | 3 | 252 | POINT(3427927.9 5793243.9)
5 | 3 | 249 | POINT(3427927.95 5793243.9)
1 | 4 | 251 | POINT(3427927.75 5793243.85)
2 | 4 | 253 | POINT(3427927.8 5793243.85)
3 | 4 | 254 | POINT(3427927.85 5793243.85)
4 | 4 | 254 | POINT(3427927.9 5793243.85)
5 | 4 | 253 | POINT(3427927.95 5793243.85)
1 | 5 | 252 | POINT(3427927.75 5793243.8)
2 | 5 | 250 | POINT(3427927.8 5793243.8)
3 | 5 | 254 | POINT(3427927.85 5793243.8)
4 | 5 | 254 | POINT(3427927.9 5793243.8)
5 | 5 | 254 | POINT(3427927.95 5793243.8)

```

Siehe auch

[ST_DumpAsPolygons](#), [ST_PixelAsPolygon](#), [ST_PixelAsPolygons](#), [ST_PixelAsPoint](#), [ST_PixelAsCentroid](#), [ST_PixelAsCentroids](#)

9.6.5 ST_PixelAsCentroid

ST_PixelAsCentroid — Gibt den geometrischen Schwerpunkt (Punktgeometrie) der Fläche aus, die durch das Pixel repräsentiert wird.

Synopsis

geometry **ST_PixelAsCentroid**(raster rast, integer x, integer y);

Beschreibung

Gibt den geometrischen Schwerpunkt (Punktgeometrie) der Fläche aus, die durch das Pixel repräsentiert wird.

Verfügbarkeit: 2.1.0

Beispiele

```

SELECT ST_AsText(ST_PixelAsCentroid(rast, 1, 1)) FROM dummy_rast WHERE rid = 1;

 st_astext
-----
POINT(1.5 2)

```

Siehe auch

[ST_DumpAsPolygons](#), [ST_PixelAsPolygon](#), [ST_PixelAsPolygons](#), [ST_PixelAsPoint](#), [ST_PixelAsPoints](#), [ST_PixelAsCentroids](#)

9.6.6 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

geometry **ST_PixelAsCentroids**(raster rast, integer band=1, boolean exclude_nodata_value=TRUE);

Beschreibung

Gibt den geometrischen Schwerpunkt (Punktgeometrie) für jedes Pixel des Rasterbandes zurück, zusammen mit dem Zellwert und den X- und Y-Rasterkoordinaten eines jeden Pixels. Die Koordinaten der Punkte entsprechen dem geometrischen Schwerpunkt der Pixel.



Note

Wenn `exclude_nodata_value = TRUE`, dann werden nur jene Pixel als Punkte zurückgegeben deren Zellwerte nicht NODATA sind.

Verfügbarkeit: 2.1.0

Änderung: 2.1.1 Die Verhaltensweise von "exclude_nodata_value" wurde geändert.

Beispiele

```
--LATERAL JOIN - benötigt PostgreSQL 9.3+
SELECT x, y, val, ST_AsText(geom)
      FROM (SELECT dp.* FROM dummy_rast, LATERAL ST_PixelAsCentroids(rast, 1) AS dp WHERE ←
            rid = 2) foo;
```

x	y	val	st_astext
1	1	253	POINT(3427927.775 5793243.975)
2	1	254	POINT(3427927.825 5793243.975)
3	1	253	POINT(3427927.875 5793243.975)
4	1	254	POINT(3427927.925 5793243.975)
5	1	254	POINT(3427927.975 5793243.975)
1	2	253	POINT(3427927.775 5793243.925)
2	2	254	POINT(3427927.825 5793243.925)
3	2	254	POINT(3427927.875 5793243.925)
4	2	253	POINT(3427927.925 5793243.925)
5	2	249	POINT(3427927.975 5793243.925)
1	3	250	POINT(3427927.775 5793243.875)
2	3	254	POINT(3427927.825 5793243.875)
3	3	254	POINT(3427927.875 5793243.875)
4	3	252	POINT(3427927.925 5793243.875)
5	3	249	POINT(3427927.975 5793243.875)
1	4	251	POINT(3427927.775 5793243.825)
2	4	253	POINT(3427927.825 5793243.825)
3	4	254	POINT(3427927.875 5793243.825)
4	4	254	POINT(3427927.925 5793243.825)
5	4	253	POINT(3427927.975 5793243.825)
1	5	252	POINT(3427927.775 5793243.775)
2	5	250	POINT(3427927.825 5793243.775)
3	5	254	POINT(3427927.875 5793243.775)
4	5	254	POINT(3427927.925 5793243.775)
5	5	254	POINT(3427927.975 5793243.775)

Siehe auch

[ST_DumpAsPolygons](#), [ST_PixelAsPolygon](#), [ST_PixelAsPolygons](#), [ST_PixelAsPoint](#), [ST_PixelAsPoints](#), [ST_PixelAsCentroid](#)

9.6.7 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);
double precision ST_Value(raster rast, integer x, integer y, boolean exclude_nodata_value=true);
double precision ST_Value(raster rast, integer band, integer x, integer y, boolean exclude_nodata_value=true);
```

Beschreibung

Gibt den Zellwert eines Pixels aus, das über columnx und rowy oder durch einen 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 `TRUE` gesetzt ist, werden nur die Pixel ohne `nodata` Wert betrachtet. Wenn `exclude_nodata_value` auf `FALSE` gesetzt ist, dann werden alle Pixel berücksichtigt.

Erweiterung: 2.0.0 Der optionale Übergabewert "exclude_nodata_value" wurde hinzugefügt.

Beispiele

```
-- gibt die Rasterwerte an bestimmten Punkten einer PostGIS Geometrie aus
-- die SRID der Geometrie und des Rasters muss übereinstimmen
SELECT rid, ST_Value(rast, foo.pt_geom) As b1pval, ST_Value(rast, 2, foo.pt_geom) As b2pval
FROM dummy_rast CROSS JOIN (SELECT ST_SetSRID(ST_Point(3427927.77, 5793243.76), 0) As
    pt_geom) As foo
WHERE rid=2;
```

rid	b1pval	b2pval
2	252	79

```
-- allgemeines, fiktives Beispiel, das eine echte Tabelle verwendet
SELECT rid, ST_Value(rast, 3, sometable.geom) As b3pval
FROM sometable
WHERE ST_Intersects(rast, sometable.geom);
```

```
SELECT rid, ST_Value(rast, 1, 1, 1) As b1pval,
    ST_Value(rast, 2, 1, 1) As b2pval, ST_Value(rast, 3, 1, 1) As b3pval
FROM dummy_rast
WHERE rid=2;
```

rid	b1pval	b2pval	b3pval
2	253	78	70

```
--- Alle Werte aller Pixel in den Bändern 1,2,3 auslesen ---
SELECT x, y, ST_Value(rast, 1, x, y) As b1val,
    ST_Value(rast, 2, x, y) As b2val, ST_Value(rast, 3, x, y) As b3val
FROM dummy_rast CROSS JOIN
generate_series(1, 1000) As x CROSS JOIN generate_series(1, 1000) As y
WHERE rid = 2 AND x <= ST_Width(rast) AND y <= ST_Height(rast);
```

x	y	b1val	b2val	b3val
1	1	253	78	70
1	2	253	96	80
1	3	250	99	90
1	4	251	89	77
1	5	252	79	62
2	1	254	98	86
2	2	254	118	108
:				
:				

--- Alle Werte der Bänder 1,2,3 so wie oben auslesen und zusätzlich den oberen linken Punkt ↵
für jedes Pixel als Punktgeometrie ausgeben --

```
SELECT ST_AsText(ST_SetSRID(
    ST_Point(ST_UpperLeftX(rast) + ST_ScaleX(rast)*x,
            ST_UpperLeftY(rast) + ST_ScaleY(rast)*y),
    ST_SRID(rast))) As uplpt
, ST_Value(rast, 1, x, y) As b1val,
  ST_Value(rast, 2, x, y) As b2val, ST_Value(rast, 3, x, y) As b3val
FROM dummy_rast CROSS JOIN
generate_series(1,1000) As x CROSS JOIN generate_series(1,1000) As y
WHERE rid = 2 AND x <= ST_Width(rast) AND y <= ST_Height(rast);
```

uplpt	b1val	b2val	b3val
POINT(3427929.25 5793245.5)	253	78	70
POINT(3427929.25 5793247)	253	96	80
POINT(3427929.25 5793248.5)	250	99	90
:			

--- Ein Polygon erzeugen, dass sich aus der Vereinigung aller Pixel ergibt,
-- die in einen bestimmten Wertebereich fallen und ein bestimmtes Polygon schneiden ↵
--

```
SELECT ST_AsText(ST_Union(pixpolyg)) As shadow
FROM (SELECT ST_Translate(ST_MakeEnvelope(
    ST_UpperLeftX(rast), ST_UpperLeftY(rast),
    ST_UpperLeftX(rast) + ST_ScaleX(rast),
    ST_UpperLeftY(rast) + ST_ScaleY(rast), 0
    ), ST_ScaleX(rast)*x, ST_ScaleY(rast)*y
    ) As pixpolyg, ST_Value(rast, 2, x, y) As b2val
FROM dummy_rast CROSS JOIN
generate_series(1,1000) As x CROSS JOIN generate_series(1,1000) As y
WHERE rid = 2
AND x <= ST_Width(rast) AND y <= ST_Height(rast)) As foo
WHERE
ST_Intersects(
    pixpolyg,
    ST_GeomFromText('POLYGON((3427928 5793244,3427927.75 5793243.75,3427928 ↵
    5793243.75,3427928 5793244))',0)
    ) AND b2val != 254;
```

```
shadow
-----
MULTIPOLYGON(((3427928 5793243.9,3427928 5793243.85,3427927.95 5793243.85,3427927.95 ↵
5793243.9,
3427927.95 5793243.95,3427928 5793243.95,3427928.05 5793243.95,3427928.05 ↵
5793243.9,3427928 5793243.9)),((3427927.95 5793243.9,3427927.95 579324
3.85,3427927.9 5793243.85,3427927.85 5793243.85,3427927.85 5793243.9,3427927.9 ↵
```

```

5793243.9,3427927.9 5793243.95,
3427927.95 5793243.95,3427927.95 5793243.9)), ((3427927.85 5793243.75,3427927.85  ←
5793243.7,3427927.8 5793243.7,3427927.8 5793243.75
,3427927.8 5793243.8,3427927.8 5793243.85,3427927.85 5793243.85,3427927.85  ←
5793243.8,3427927.85 5793243.75)),
((3427928.05 5793243.75,3427928.05 5793243.7,3427928 5793243.7,3427927.95  ←
5793243.7,3427927.95 5793243.75,3427927.95 5793243.8,3427
927.95 5793243.85,3427928 5793243.85,3427928 5793243.8,3427928.05 5793243.8,
3427928.05 5793243.75)), ((3427927.95 5793243.75,3427927.95 5793243.7,3427927.9  ←
5793243.7,3427927.85 5793243.7,
3427927.85 5793243.75,3427927.85 5793243.8,3427927.85 5793243.85,3427927.9 5793243.85,
3427927.95 5793243.85,3427927.95 5793243.8,3427927.95 5793243.75)))

```

```

--- Die Überprüfung sämtlicher Pixel einer großen Rasterkachel kann viel Zeit in Anspruch  ←
nehmen.
--- Sie können die Geschwindigkeit, zu Kosten einiger Größenordnungen an Genauigkeit  ←
beträchtlich steigern,
-- indem Sie Pixel mit dem optionalen Parameter "step" von "generate_series" auswählen.
-- Das nächste Beispiel gleicht dem Vorigen, aber es überprüft nur 1 von jeweils 4 (2x2)  ←
Pixel und setzt das zuletzt untersuchte Pixel als Wert für die nachfolgenden 3 Pixel  ←
ein

```

```

SELECT ST_AsText(ST_Union(pixpolyg)) As shadow
FROM (SELECT ST_Translate(ST_MakeEnvelope(
    ST_UpperLeftX(rast), ST_UpperLeftY(rast),
    ST_UpperLeftX(rast) + ST_ScaleX(rast)*2,
    ST_UpperLeftY(rast) + ST_ScaleY(rast)*2, 0
    ), ST_ScaleX(rast)*x, ST_ScaleY(rast)*y
    ) As pixpolyg, ST_Value(rast, 2, x, y) As b2val
    FROM dummy_rast CROSS JOIN
generate_series(1,1000,2) As x CROSS JOIN generate_series(1,1000,2) As y
WHERE rid = 2
    AND x <= ST_Width(rast) AND y <= ST_Height(rast) ) As foo
WHERE
    ST_Intersects(
        pixpolyg,
        ST_GeomFromText('POLYGON((3427928 5793244,3427927.75 5793243.75,3427928  ←
5793243.75,3427928 5793244))',0)
    ) AND b2val != 254;

shadow
-----
MULTIPOLYGON(((3427927.9 5793243.85,3427927.8 5793243.85,3427927.8 5793243.95,
3427927.9 5793243.95,3427928 5793243.95,3427928.1 5793243.95,3427928.1 5793243.85,3427928  ←
5793243.85,3427927.9 5793243.85)),
((3427927.9 5793243.65,3427927.8 5793243.65,3427927.8 5793243.75,3427927.8  ←
5793243.85,3427927.9 5793243.85,
3427928 5793243.85,3427928 5793243.75,3427928.1 5793243.75,3427928.1 5793243.65,3427928  ←
5793243.65,3427927.9 5793243.65)))

```

Siehe auch

[ST_SetValue](#), [ST_DumpAsPolygons](#), [ST_NumBands](#), [ST_PixelAsPolygon](#), [ST_ScaleX](#), [ST_ScaleY](#), [ST_UpperLeftX](#), [ST_UpperLeftY](#), [ST_SRID](#), [ST_AsText](#), [ST_Point](#), [ST_MakeEnvelope](#), [ST_Intersects](#), [ST_Intersection](#)

9.6.8 ST_NearestValue

ST_NearestValue — Gibt den nächstgelegenen nicht NODATA Wert eines bestimmten Pixels aus, das über "columnx" und "rowy" oder durch eine Punktgeometrie - im gleichen Koordinatenreferenzsystem wie der Raster - ausgewählt wird.

Synopsis

```
double precision ST_NearestValue(raster rast, integer bandnum, geometry pt, boolean exclude_nodata_value=true);
double precision ST_NearestValue(raster rast, geometry pt, boolean exclude_nodata_value=true);
double precision ST_NearestValue(raster rast, integer bandnum, integer columnx, integer rowy, boolean exclude_nodata_value=true);
double precision ST_NearestValue(raster rast, integer columnx, integer rowy, boolean exclude_nodata_value=true);
```

Beschreibung

Gibt den nächstgelegenen, nicht-NODATA Wert eines bestimmten Pixels aus, das über "columnx" und "rowy", oder durch einen geometrischen Punkt angegeben wird. Falls das angegebene Pixel den Wert NODATA hat, findet die Funktion das nächstgelegene Pixel, das nicht den Wert NODATA hat.

Die Bandnummern beginnen mit 1 und wenn keine Bandnummer angegeben ist, dann wird für bandnum 1 angenommen. Wenn `exclude_nodata_value` auf FALSE gesetzt ist, werden auch die Pixel mit einem nodata Wert einbezogen. Wenn `exclude_nodata_value` nicht übergeben wird, dann wird er über die Metadaten des Rasters ausgelesen.

Verfügbarkeit: 2.1.0



Note

ST_NearestValue ist eine Alternative zu ST_Value.

Beispiele

```
-- Pixel 2x2 hat einen Wert
SELECT
    ST_Value(rast, 2, 2) AS value,
    ST_NearestValue(rast, 2, 2) AS nearestvalue
FROM (
    SELECT
        ST_SetValue(
            ST_SetValue(
                ST_SetValue(
                    ST_SetValue(
                        ST_SetValue(
                            ST_AddBand(
                                ST_MakeEmptyRaster(5, 5, ↵
                                    -2, 2, 1, -1, 0, 0, 0),
                                '8BUI'::text, 1, 0
                            ),
                            1, 1, 0.
                        ),
                        2, 3, 0.
                    ),
                    3, 5, 0.
                ),
                4, 2, 0.
            ),
            5, 4, 0.
        ) AS rast
    ) AS foo

value | nearestvalue
-----+-----
1 | 1
```

```
-- Pixel 2x3 ist NODATA
SELECT
  ST_Value(rast, 2, 3) AS value,
  ST_NearestValue(rast, 2, 3) AS nearestvalue
FROM (
  SELECT
    ST_SetValue(
      ST_SetValue(
        ST_SetValue(
          ST_SetValue(
            ST_SetValue(
              ST_AddBand(
                ST_MakeEmptyRaster(5, 5, ←
                  -2, 2, 1, -1, 0, 0, 0),
                '8BUI'::text, 1, 0
              ),
              1, 1, 0.
            ),
            2, 3, 0.
          ),
          3, 5, 0.
        ),
        4, 2, 0.
      ),
      5, 4, 0.
    ) AS rast
  ) AS foo

value | nearestvalue
-----+-----
      |          1
```

Siehe auch

[ST_Neighborhood](#), [ST_Value](#)

9.6.9 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=true);
```

Beschreibung

Gibt für ein Pixel eines Bandes die ringsherum liegenden nicht-`NODATA` Werte in einem 2D-Feld mit Double Precision zurück. Das Pixel kann entweder über columnX und rowY, oder über einen geometrischen Punkt der im selben Koordinatenreferenzsys-

tem wie der Raster vorliegt übergeben werden. Die Parameter `distanceX` und `distanceY` bestimmen die Anzahl der Pixel auf der X- und Y-Achse, um das festgelegte Pixel herum; z.B.: Sie möchten alle Werte innerhalb einer Entfernung von 3 Pixel entlang der X-Achse und einer Entfernung von 2 Pixel entlang der Y-Achse, um das Pixel von Interesse herum. Der Wert im Zentrum des 2-D Feldes ist der Wert jenes Pixels, das über `columnX` und `rowY` oder über einen geometrischen Punkt übergeben wurde.

Die Bandnummern beginnen mit 1 und wenn keine Bandnummer angegeben ist, dann wird für `bandnum` 1 angenommen. Wenn `exclude_nodata_value` auf `FALSE` gesetzt ist, werden auch die Pixel mit einem `nodata` Wert einbezogen. Wenn `exclude_nodata_value` nicht übergeben wird, dann wird er über die Metadaten des Rasters ausgelesen.

**Note**

Die Anzahl der Elemente entlang jeder Achse des zurückgegebenen 2D-Feldes ergibt sich aus $2 * (\text{distanceX} | \text{distanceY}) + 1$. So ergibt sich bei einer `distanceX` und einer `distanceY` von je 1, ein Feld von 3x3.

**Note**

Das 2-D Feld kann einer beliebigen, integrierten Funktion zur Weiterverarbeitung übergeben werden, wie z.B. an `ST_Min4ma`, `ST_Sum4ma`, `ST_Mean4ma`.

Verfügbarkeit: 2.1.0

Beispiele

```
-- Pixel 2x2 hat einen Wert
SELECT
    ST_Neighborhood(rast, 2, 2, 1, 1)
FROM (
    SELECT
        ST_SetValues(
            ST_AddBand(
                ST_MakeEmptyRaster(5, 5, -2, 2, 1, -1, 0, 0, 0),
                '8BUI'::text, 1, 0
            ),
            1, 1, 1, ARRAY[
                [0, 1, 1, 1, 1],
                [1, 1, 1, 0, 1],
                [1, 0, 1, 1, 1],
                [1, 1, 1, 1, 0],
                [1, 1, 0, 1, 1]
            ]::double precision[],
            1
        ) AS rast
    ) AS foo

    st_neighborhood
-----
{{NULL,1,1},{1,1,NULL},{1,1,1}}
```

```
-- Pixel 2x3 ist NODATA
SELECT
    ST_Neighborhood(rast, 2, 3, 1, 1)
FROM (
    SELECT
        ST_SetValues(
            ST_AddBand(
                ST_MakeEmptyRaster(5, 5, -2, 2, 1, -1, 0, 0, 0),
```

```

        '8BUI'::text, 1, 0
    ),
    1, 1, 1, ARRAY[
        [0, 1, 1, 1, 1],
        [1, 1, 1, 0, 1],
        [1, 0, 1, 1, 1],
        [1, 1, 1, 1, 0],
        [1, 1, 0, 1, 1]
    ]::double precision[],
    1
) AS rast
) AS foo

    st_neighborhood
-----
{{1,1,1},{1,NULL,1},{1,1,1}}

-- Pixel 3x3 hat einen Wert
-- exclude_nodata_value = FALSE
SELECT
    ST_Neighborhood(rast, 3, 3, 1, 1, false)
FROM (
    ST_SetValues(
        ST_AddBand(
            ST_MakeEmptyRaster(5, 5, -2, 2, 1, -1, 0, 0, 0),
            '8BUI'::text, 1, 0
        ),
        1, 1, 1, ARRAY[
            [0, 1, 1, 1, 1],
            [1, 1, 1, 0, 1],
            [1, 0, 1, 1, 1],
            [1, 1, 1, 1, 0],
            [1, 1, 0, 1, 1]
        ]::double precision[],
        1
    ) AS rast
) AS foo

    st_neighborhood
-----
{{1,0,1},{1,1,1},{0,1,1}}

```

Siehe auch

[ST_NearestValue](#), [ST_Min4ma](#), [ST_Max4ma](#), [ST_Sum4ma](#), [ST_Mean4ma](#), [ST_Range4ma](#), [ST_Distinct4ma](#), [ST_StdDev4ma](#)

9.6.10 ST_SetValue

ST_SetValue — Setzt den Wert für ein Pixel eines Bandes, das über columnx und rowy festgelegt wird, oder für die Pixel die eine bestimmte Geometrie schneiden, und gibt den veränderten Raster zurück. Die Bandnummerierung beginnt mit 1; wenn die Bandnummer nicht angegeben ist, wird 1 angenommen.

Synopsis

```

raster ST_SetValue(raster rast, integer bandnum, geometry geom, double precision newvalue);
raster ST_SetValue(raster rast, geometry geom, double precision newvalue);
raster ST_SetValue(raster rast, integer bandnum, integer columnx, integer rowy, double precision newvalue);
raster ST_SetValue(raster rast, integer columnx, integer rowy, double precision newvalue);

```

Beschreibung

Setzt die Werte bestimmter Pixel eines Bandes auf einen neuen Wert und gibt den veränderten Raster zurück. Die Pixel können über die Rasterzeile und die Rasterspalte, oder über eine Geometrie festgelegt werden. Wenn die Bandnummer nicht angegeben ist, wird 1 angenommen.

Erweiterung: 2.1.0 Die geometrische Variante von `ST_SetValue()` unterstützt nun jeden geometrischen Datentyp, nicht nur `POINT`. Die geometrische Variante ist ein Adapter für die `geomval[]` Variante von `ST_SetValues()`

Beispiele

```
-- Beispiel mit Geometrie
SELECT (foo.geomval).val, ST_AsText(ST_Union((foo.geomval).geom))
FROM (SELECT ST_DumpAsPolygons(
          ST_SetValue(rast,1,
                     ST_Point(3427927.75, 5793243.95),
                     50)
        ) As geomval
FROM dummy_rast
where rid = 2) As foo
WHERE (foo.geomval).val < 250
GROUP BY (foo.geomval).val;
```

val	st_astext
50	POLYGON((3427927.75 5793244,3427927.75 5793243.95,3427927.8 579324 ...
249	POLYGON((3427927.95 5793243.95,3427927.95 5793243.85,3427928 57932 ...

```
-- Den veränderten Raster speichern --
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](#)

9.6.11 ST_SetValues

`ST_SetValues` — Gibt einen Raster zurück, der durch das Setzen der Werte eines bestimmten Bandes verändert wurde.

Synopsis

```
raster ST_SetValues(raster rast, integer nband, integer columnx, integer rowy, double precision[][] newvalueset, boolean[][]
noset=NULL, boolean keepnodata=FALSE);
raster ST_SetValues(raster rast, integer nband, integer columnx, integer rowy, double precision[][] newvalueset, double precision
nosetvalue, boolean keepnodata=FALSE);
raster ST_SetValues(raster rast, integer nband, integer columnx, integer rowy, integer width, integer height, double precision
newvalue, boolean keepnodata=FALSE);
raster ST_SetValues(raster rast, integer columnx, integer rowy, integer width, integer height, double precision newvalue, boolean
keepnodata=FALSE);
raster ST_SetValues(raster rast, integer nband, geomval[] geomvalset, boolean keepnodata=FALSE);
```

Beschreibung

Gibt einen Raster zurück, der durch das Setzen bestimmter Pixel auf einen neuen Wert (neue Werte) für das ausgewiesene Band verändert wurde.

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 Variante 3 mit der Ausnahme, dass die Pixel des ersten Bandes von `rast` gesetzt werden.

Bei der Variante 5 wird ein Feld von `geomval` verwendet um die Pixel zu bestimmen. Wenn die gesamte Geometrie des Feldes vom Datentyp `POINT` oder `MULTIPOINT` ist, verwendet die Funktion Länge und Breite eines jeden Punktes um den Wert eines Pixels direkt zu setzen. Andernfalls wird die Geometrie in Raster umgewandelt über die dann in einem Schritt iteriert wird. Siehe das Beispiel mit der Variante 5.

Verfügbarkeit: 2.1.0

Beispiele: Variante 1

```
/*
ST_SetValues() zeigt folgendes Verhalten...

+ - + - + - +      + - + - + - +
| 1 | 1 | 1 | 1 |    | 1 | 1 | 1 | 1 |
+ - + - + - +      + - + - + - +
| 1 | 1 | 1 | 1 |    =
>   | 1 | 9 | 9 |
+ - + - + - +      + - + - + - +
| 1 | 1 | 1 | 1 |    | 1 | 9 | 9 |
+ - + - + - +      + - + - + - +
*/
SELECT
    (poly).x,
    (poly).y,
    (poly).val
FROM (
    SELECT
        ST_PixelAsPolygons(
            ST_SetValues(
                ST_AddBand(
                    ST_MakeEmptyRaster(3, 3, 0, 0, 1, -1, 0, 0, 0),
                    1, '8BUI', 1, 0
                ),
                1, 2, 2, ARRAY[[9, 9], [9, 9]]::double precision[][]
            )
        ) AS poly
) foo
ORDER BY 1, 2;

x | y | val
```

```

----+----+-----
1 | 1 | 1
1 | 2 | 1
1 | 3 | 1
2 | 1 | 1
2 | 2 | 9
2 | 3 | 9
3 | 1 | 1
3 | 2 | 9
3 | 3 | 9

```

```

/*
ST_SetValues() zeigt folgendes Verhalten...

```

```

+ - + - + - +      + - + - + - +
| 1 | 1 | 1 |      | 9 | 9 | 9 |
+ - + - + - +      + - + - + - +
| 1 | 1 | 1 |      =
> | 9 | 1 | 9 |
+ - + - + - +      + - + - + - +
| 1 | 1 | 1 |      | 9 | 9 | 9 |
+ - + - + - +      + - + - + - +

```

```

*/
SELECT
    (poly).x,
    (poly).y,
    (poly).val
FROM (
    SELECT
        ST_PixelAsPolygons(
            ST_SetValues(
                ST_AddBand(
                    ST_MakeEmptyRaster(3, 3, 0, 0, 1, -1, 0, 0, 0),
                    1, '8BUI', 1, 0
                ),
                1, 1, 1, ARRAY[[9, 9, 9], [9, NULL, 9], [9, 9, 9]]::double ↵
                    precision[][])
            ) AS poly
        ) foo
ORDER BY 1, 2;

```

```

x | y | val
----+----+-----
1 | 1 | 9
1 | 2 | 9
1 | 3 | 9
2 | 1 | 9
2 | 2 |
2 | 3 | 9
3 | 1 | 9
3 | 2 | 9
3 | 3 | 9

```

```

/*
ST_SetValues() zeigt folgendes Verhalten...

```

```

+ - + - + - +      + - + - + - +
| 1 | 1 | 1 |      | 9 | 9 | 9 |
+ - + - + - +      + - + - + - +
| 1 | 1 | 1 |      =

```

```

>      | 1 |      | 9 |
+ - + - + - +      + - + - + - +
| 1 | 1 | 1 |      | 9 | 9 | 9 |
+ - + - + - +      + - + - + - +
*/
SELECT
    (poly).x,
    (poly).y,
    (poly).val
FROM (
SELECT
    ST_PixelAsPolygons(
        ST_SetValues(
            ST_AddBand(
                ST_MakeEmptyRaster(3, 3, 0, 0, 1, -1, 0, 0, 0),
                1, '8BUI', 1, 0
            ),
            1, 1, 1,
            ARRAY[[9, 9, 9], [9, NULL, 9], [9, 9, 9]]::double precision ←
                [],
            ARRAY[[false], [true]]::boolean[][])
        ) AS poly
) foo
ORDER BY 1, 2;

 x | y | val
---+---+-----
 1 | 1 |   9
 1 | 2 |   1
 1 | 3 |   9
 2 | 1 |   9
 2 | 2 |
 2 | 3 |   9
 3 | 1 |   9
 3 | 2 |   9
 3 | 3 |   9

```

```

/*
ST_SetValues() zeigt folgendes Verhalten...

+ - + - + - +      + - + - + - +
|   | 1 | 1 |      |   | 9 | 9 |
+ - + - + - +      + - + - + - +
| 1 | 1 | 1 |      =
>      | 1 |      | 9 |
+ - + - + - +      + - + - + - +
| 1 | 1 | 1 |      | 9 | 9 | 9 |
+ - + - + - +      + - + - + - +
*/
SELECT
    (poly).x,
    (poly).y,
    (poly).val
FROM (
SELECT
    ST_PixelAsPolygons(
        ST_SetValues(
            ST_SetValue(
                ST_AddBand(
                    ST_MakeEmptyRaster(3, 3, 0, 0, 1, -1, 0, 0, 0),
                    1, '8BUI', 1, 0

```

```

),
1, 1, 1, NULL
),
1, 1, 1,
ARRAY[[9, 9, 9], [9, NULL, 9], [9, 9, 9]]::double precision ←
[],
ARRAY[[false], [true]]::boolean[],
TRUE
)
) AS poly
) foo
ORDER BY 1, 2;

 x | y | val
---+---+-----
 1 | 1 | 
 1 | 2 |  1
 1 | 3 |  9
 2 | 1 |  9
 2 | 2 | 
 2 | 3 |  9
 3 | 1 |  9
 3 | 2 |  9
 3 | 3 |  9

```

Beispiele: Variante 2

```

/*
ST_SetValues() zeigt folgendes Verhalten...

+ - + - + - +      + - + - + - +
| 1 | 1 | 1 |      | 1 | 1 | 1 |
+ - + - + - +      + - + - + - +
| 1 | 1 | 1 |      =
> | 1 | 9 | 9 |
+ - + - + - +      + - + - + - +
| 1 | 1 | 1 |      | 1 | 9 | 9 |
+ - + - + - +      + - + - + - +
*/
SELECT
    (poly).x,
    (poly).y,
    (poly).val
FROM (
SELECT
    ST_PixelAsPolygons(
        ST_SetValues(
            ST_AddBand(
                ST_MakeEmptyRaster(3, 3, 0, 0, 1, -1, 0, 0, 0),
                1, '8BUI', 1, 0
            ),
            1, 1, 1, ARRAY[[-1, -1, -1], [-1, 9, 9], [-1, 9, 9]]::double ←
                precision[], -1
        )
    ) AS poly
) foo
ORDER BY 1, 2;

 x | y | val
---+---+-----

```

```

1 | 1 | 1
1 | 2 | 1
1 | 3 | 1
2 | 1 | 1
2 | 2 | 9
2 | 3 | 9
3 | 1 | 1
3 | 2 | 9
3 | 3 | 9

```

```

/*
Diese Beispiel ähnelt dem Vorigen. An Stelle von nosetvalue = -1, ist nosetvalue = NULL ←
gesetzt

```

```

ST_SetValues() zeigt folgendes Verhalten...

```

```

+ - + - + - +      + - + - + - +
| 1 | 1 | 1 |      | 1 | 1 | 1 |
+ - + - + - +      + - + - + - +
| 1 | 1 | 1 |      =
> | 1 | 9 | 9 |
+ - + - + - +      + - + - + - +
| 1 | 1 | 1 |      | 1 | 9 | 9 |
+ - + - + - +      + - + - + - +

```

```

*/
SELECT
    (poly).x,
    (poly).y,
    (poly).val
FROM (
    SELECT
        ST_PixelAsPolygons(
            ST_SetValues(
                ST_AddBand(
                    ST_MakeEmptyRaster(3, 3, 0, 0, 1, -1, 0, 0, 0),
                    1, '8BUI', 1, 0
                ),
                1, 1, 1, ARRAY[[NULL, NULL, NULL], [NULL, 9, 9], [NULL, 9, 9]]::
                    double precision[[]], NULL::double precision
            )
        ) AS poly
    ) foo
ORDER BY 1, 2;

```

```

x | y | val
---+---+-----
1 | 1 | 1
1 | 2 | 1
1 | 3 | 1
2 | 1 | 1
2 | 2 | 9
2 | 3 | 9
3 | 1 | 1
3 | 2 | 9
3 | 3 | 9

```

Beispiele: Variante 3

```

/*
ST_SetValues() führt folgendes aus...

```



```

+ - + - + - +      + - + - + - +
| 1 | 1 | 1 |      | 1 | 1 | 1 |
+ - + - + - +      + - + - + - +
| 1 | 1 | 1 |      =
>   | 1 | 9 | 9 |
+ - + - + - +      + - + - + - +
| 1 | 1 | 1 |      | 1 | 9 | 9 |
+ - + - + - +      + - + - + - +
*/
SELECT
    (poly).x,
    (poly).y,
    (poly).val
FROM (
SELECT
    ST_PixelAsPolygons(
        ST_SetValues(
            ST_AddBand(
                ST_MakeEmptyRaster(3, 3, 0, 0, 1, -1, 0, 0, 0),
                1, '8BUI', 1, 0
            ),
            1, 2, 2, 2, 2, 9
        )
    ) AS poly
) foo
ORDER BY 1, 2;

```

```

  x | y | val
-----+-----
  1 | 1 |   1
  1 | 2 |   1
  1 | 3 |   1
  2 | 1 |   1
  2 | 2 |   9
  2 | 3 |   9
  3 | 1 |   1
  3 | 2 |   9
  3 | 3 |   9

```

```

/*
ST_SetValues() führt folgendes aus...

+ - + - + - +      + - + - + - +
| 1 | 1 | 1 |      | 1 | 1 | 1 |
+ - + - + - +      + - + - + - +
| 1 |   | 1 |      =
>   | 1 |   | 9 |
+ - + - + - +      + - + - + - +
| 1 | 1 | 1 |      | 1 | 9 | 9 |
+ - + - + - +      + - + - + - +
*/
SELECT
    (poly).x,
    (poly).y,
    (poly).val
FROM (
SELECT
    ST_PixelAsPolygons(
        ST_SetValues(
            ST_SetValue(
                ST_AddBand(
                    ST_MakeEmptyRaster(3, 3, 0, 0, 1, -1, 0, 0, 0),

```

```

        1, '8BUI', 1, 0
    ),
    1, 2, 2, NULL
),
1, 2, 2, 2, 2, 9, TRUE
)
) AS poly
) foo
ORDER BY 1, 2;
```

x	y	val
1	1	1
1	2	1
1	3	1
2	1	1
2	2	
2	3	9
3	1	1
3	2	9
3	3	9

Beispiele: Variante 5

```
WITH foo AS (
    SELECT 1 AS rid, ST_AddBand(ST_MakeEmptyRaster(5, 5, 0, 0, 1, -1, 0, 0, 0), 1, '8 BUI', 0, 0) AS rast
), bar AS (
    SELECT 1 AS gid, 'SRID=0;POINT(2.5 -2.5)::geometry geom UNION ALL
    SELECT 2 AS gid, 'SRID=0;POLYGON((1 -1, 4 -1, 4 -4, 1 -4, 1 -1))::geometry geom UNION ALL
    SELECT 3 AS gid, 'SRID=0;POLYGON((0 0, 5 0, 5 -1, 1 -1, 1 -4, 0 -4, 0 0))::geometry geom UNION ALL
    SELECT 4 AS gid, 'SRID=0;MULTIPOINT(0 0, 4 4, 4 -4)::geometry
)
SELECT
    rid, gid, ST_DumpValues(ST_SetValue(rast, 1, geom, gid))
FROM foo t1
CROSS JOIN bar t2
ORDER BY rid, gid;
```

rid	gid	st_dumpvalues
1	1	(1, "{NULL,NULL,NULL,NULL,NULL},{NULL,NULL,NULL,NULL,NULL},{NULL,NULL,1,NULL,NULL},{NULL,NULL,NULL,NULL,NULL},{NULL,NULL,NULL,NULL,NULL}}")
1	2	(1, "{NULL,NULL,NULL,NULL,NULL},{NULL,2,2,2,NULL},{NULL,2,2,2,NULL},{NULL,2,2,2,NULL},{NULL,NULL,NULL,NULL}}")
1	3	(1, "{3,3,3,3,3},{3,NULL,NULL,NULL,NULL},{3,NULL,NULL,NULL,NULL},{3,NULL,NULL,NULL,NULL},{NULL,NULL,NULL,NULL}}")
1	4	(1, "{4,NULL,NULL,NULL,NULL},{NULL,NULL,NULL,NULL,NULL},{NULL,NULL,NULL,NULL,NULL},{NULL,NULL,NULL,NULL,4}}")

(4 rows)

Das folgende Beispiel zeigt, dass bestehende "geomvals" durch ein Feld mit "geomvals" später überschrieben werden können

```
WITH foo AS (
    SELECT 1 AS rid, ST_AddBand(ST_MakeEmptyRaster(5, 5, 0, 0, 1, -1, 0, 0, 0), 1, '8 BUI', 0, 0) AS rast
), bar AS (
```

```

SELECT 1 AS gid, 'SRID=0;POINT(2.5 -2.5)::geometry geom UNION ALL
SELECT 2 AS gid, 'SRID=0;POLYGON((1 -1, 4 -1, 4 -4, 1 -4, 1 -1))::geometry geom ↵
UNION ALL
SELECT 3 AS gid, 'SRID=0;POLYGON((0 0, 5 0, 5 -1, 1 -1, 1 -4, 0 -4, 0 0))':: ↵
geometry geom UNION ALL
SELECT 4 AS gid, 'SRID=0;MULTIPOINT(0 0, 4 4, 4 -4)::geometry
)
SELECT
    t1.rid, t2.gid, t3.gid, ST_DumpValues(ST_SetValues(rast, 1, ARRAY[ROW(t2.geom, t2. ↵
gid), ROW(t3.geom, t3.gid)]::geomval[]))
FROM foo t1
CROSS JOIN bar t2
CROSS JOIN bar t3
WHERE t2.gid = 1
      AND t3.gid = 2
ORDER BY t1.rid, t2.gid, t3.gid;

rid | gid | gid | st_dumpvalues
-----+-----+-----+-----
1 | 1 | 2 | (1,"{NULL,NULL,NULL,NULL,NULL},{NULL,2,2,2,NULL},{NULL,2,2,2,NULL},{ ↵
NULL,2,2,2,NULL},{NULL,NULL,NULL,NULL,NULL}}")
(1 row)

```

Dieses Beispiel ist das Gegenteil des vorigen Beispiels

```

WITH foo AS (
    SELECT 1 AS rid, ST_AddBand(ST_MakeEmptyRaster(5, 5, 0, 0, 1, -1, 0, 0, 0), 1, '8 ↵
BUI', 0, 0) AS rast
), bar AS (
    SELECT 1 AS gid, 'SRID=0;POINT(2.5 -2.5)::geometry geom UNION ALL
    SELECT 2 AS gid, 'SRID=0;POLYGON((1 -1, 4 -1, 4 -4, 1 -4, 1 -1))::geometry geom ↵
UNION ALL
    SELECT 3 AS gid, 'SRID=0;POLYGON((0 0, 5 0, 5 -1, 1 -1, 1 -4, 0 -4, 0 0))':: ↵
geometry geom UNION ALL
    SELECT 4 AS gid, 'SRID=0;MULTIPOINT(0 0, 4 4, 4 -4)::geometry
)
SELECT
    t1.rid, t2.gid, t3.gid, ST_DumpValues(ST_SetValues(rast, 1, ARRAY[ROW(t2.geom, t2. ↵
gid), ROW(t3.geom, t3.gid)]::geomval[]))
FROM foo t1
CROSS JOIN bar t2
CROSS JOIN bar t3
WHERE t2.gid = 2
      AND t3.gid = 1
ORDER BY t1.rid, t2.gid, t3.gid;

rid | gid | gid | st_dumpvalues
-----+-----+-----+-----
1 | 2 | 1 | (1,"{NULL,NULL,NULL,NULL,NULL},{NULL,2,2,2,NULL},{NULL,2,1,2,NULL},{ ↵
NULL,2,2,2,NULL},{NULL,NULL,NULL,NULL,NULL}}")
(1 row)

```

Siehe auch

[ST_Value](#), [ST_SetValue](#), [ST_PixelAsPolygons](#)

9.6.12 ST_DumpValues

ST_DumpValues — Gibt die Werte eines bestimmten Bandes als 2-dimensionales Feld aus.

Synopsis

setof record **ST_DumpValues**(raster rast , integer[] nband=NULL , boolean exclude_nodata_value=true);
double precision[][] **ST_DumpValues**(raster rast , integer nband , boolean exclude_nodata_value=true);

Beschreibung

Gibt die Werte eines bestimmten Bandes als 2-dimensionales Feld (der erste Index entspricht der Zeile, der zweite der Spalte) aus. Wenn nband NULL oder nicht gegeben ist, werden alle Rasterbänder abgearbeitet.

Verfügbarkeit: 2.1.0

Beispiele

```
WITH foo AS (  
    SELECT ST_AddBand(ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(3, 3, 0, 0, 1, -1, 0, 0, ↵  
        0), 1, '8BUI'::text, 1, 0), 2, '32BF'::text, 3, -9999), 3, '16BSI', 0, 0) AS ↵  
        rast  
)  
SELECT  
    (ST_DumpValues(rast)).*  
FROM foo;
```

nband	valarray
1	{{1,1,1},{1,1,1},{1,1,1}}
2	{{3,3,3},{3,3,3},{3,3,3}}
3	{{NULL,NULL,NULL},{NULL,NULL,NULL},{NULL,NULL,NULL}}

(3 rows)

```
WITH foo AS (  
    SELECT ST_AddBand(ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(3, 3, 0, 0, 1, -1, 0, 0, ↵  
        0), 1, '8BUI'::text, 1, 0), 2, '32BF'::text, 3, -9999), 3, '16BSI', 0, 0) AS ↵  
        rast  
)  
SELECT  
    (ST_DumpValues(rast, ARRAY[3, 1])).*  
FROM foo;
```

nband	valarray
3	{{NULL,NULL,NULL},{NULL,NULL,NULL},{NULL,NULL,NULL}}
1	{{1,1,1},{1,1,1},{1,1,1}}

(2 rows)

```
WITH foo AS (  
    SELECT ST_SetValue(ST_AddBand(ST_MakeEmptyRaster(3, 3, 0, 0, 1, -1, 0, 0, 0), 1, '8 ↵  
        BUI', 1, 0), 1, 2, 5) AS rast  
)  
SELECT  
    (ST_DumpValues(rast, 1))[2][1]  
FROM foo;
```

st_dumpvalues

```
-----
          5
(1 row)
```

Siehe auch

[ST_Value](#), [ST_SetValue](#), [ST_SetValues](#)

9.6.13 ST_PixelOfValue

ST_PixelOfValue — Gibt die columnx- und rowy-Koordinaten jener Pixel aus, deren Zellwert gleich dem gesuchten Wert ist.

Synopsis

```
setof record ST_PixelOfValue( raster rast , integer nband , double precision[] search , boolean exclude_nodata_value=true );
setof record ST_PixelOfValue( raster rast , double precision[] search , boolean exclude_nodata_value=true );
setof record ST_PixelOfValue( raster rast , integer nband , double precision search , boolean exclude_nodata_value=true );
setof record ST_PixelOfValue( raster rast , double precision search , boolean exclude_nodata_value=true );
```

Beschreibung

Gibt die columnx- und rowy-Koordinaten jener Pixel aus, deren Zellwert gleich dem gesuchten Wert ist. Wenn kein Band angegeben ist, wird Band 1 angenommen.

Verfügbarkeit: 2.1.0

Beispiele

```
SELECT
    (pixels).*
FROM (
    SELECT
        ST_PixelOfValue(
            ST_SetValue(
                ST_SetValue(
                    ST_SetValue(
                        ST_SetValue(
                            ST_SetValue(
                                ST_AddBand(
                                    ST_MakeEmptyRaster ←
                                        (5, 5, -2, 2, 1, ←
                                        -1, 0, 0, 0),
                                    '8BUI'::text, 1, 0
                                ),
                                1, 1, 0
                            ),
                            2, 3, 0
                        ),
                        3, 5, 0
                    ),
                    4, 2, 0
                ),
                5, 4, 255
            ),
            1, ARRAY[1, 255]) AS pixels
```

```
) AS foo
```

```

val | x | y
-----+-----+-----
  1 | 1 | 2
  1 | 1 | 3
  1 | 1 | 4
  1 | 1 | 5
  1 | 2 | 1
  1 | 2 | 2
  1 | 2 | 4
  1 | 2 | 5
  1 | 3 | 1
  1 | 3 | 2
  1 | 3 | 3
  1 | 3 | 4
  1 | 4 | 1
  1 | 4 | 3
  1 | 4 | 4
  1 | 4 | 5
  1 | 5 | 1
  1 | 5 | 2
  1 | 5 | 3
255 | 5 | 4
  1 | 5 | 5

```

9.7 Raster Editoren

9.7.1 ST_SetGeoReference

ST_SetGeoReference — Georeferenziert einen Raster über 6 Parameter in einem einzigen Aufruf. Die Zahlen müssen durch Leerzeichen getrennt sein. Die Funktion akzeptiert die Eingabe im Format von 'GDAL' und von 'ESRI'. Der Standardwert ist GDAL.

Synopsis

```

raster ST_SetGeoReference(raster rast, text georefcoords, text format=GDAL);
raster ST_SetGeoReference(raster rast, double precision upperleftx, double precision upperlefty, double precision scalex, double precision scaley, double precision skewx, double precision skewy);

```

Beschreibung

Georeferenziert einen Raster über 6 Parameter in einem einzigen Aufruf. Die Funktion akzeptiert die Eingabe im Format von 'GDAL' und von 'ESRI'. Der Standardwert ist GDAL. Wenn die 6 Parameter nicht angegeben sind, wird NULL zurückgegeben.

Die Formate unterscheiden sich wie folgt:

GDAL:

```
scalex skewy skewx scaley upperleftx upperlefty
```

ESRI:

```
scalex skewy skewx scaley upperleftx + scalex*0.5 upperlefty + scaley*0.5
```



Note
Wenn der Raster out-db Bänder aufweist, dann kann eine Änderung der Georeferenzierung zu einem unkorrekten Zugriff auf die extern gespeicherten Daten des Bandes führen.

Erweiterung: 2.1.0 ST_SetGeoReference(raster, double precision, ...) Variante hinzugefügt

Beispiele

```
WITH foo AS (
    SELECT ST_MakeEmptyRaster(5, 5, 0, 0, 1, -1, 0, 0, 0) AS rast
)
SELECT
    0 AS rid, (ST_Metadatas(rast)).*
FROM foo
UNION ALL
SELECT
    1, (ST_Metadatas(ST_SetGeoReference(rast, '10 0 0 -10 0.1 0.1', 'GDAL'))).*
FROM foo
UNION ALL
SELECT
    2, (ST_Metadatas(ST_SetGeoReference(rast, '10 0 0 -10 5.1 -4.9', 'ESRI'))).*
FROM foo
UNION ALL
SELECT
    3, (ST_Metadatas(ST_SetGeoReference(rast, 1, 1, 10, -10, 0.001, 0.001))).*
FROM foo
```

rid	upperleftx	upperlefty	width	height	scalex	scaley	skewx			
	skewy	srid	numbands					←		
0		0		0	5	5	1	-1	0	←
1	0	0	0.1	0.1	5	5	10	-10	0	←
2	0.09999999999999996	0.09999999999999996			5	5	10	-10	0	←
3	0.001	0	1	1	5	5	10	-10	0.001	←

Siehe auch

[ST_GeoReference](#), [ST_ScaleX](#), [ST_ScaleY](#), [ST_UpperLeftX](#), [ST_UpperLeftY](#)

9.7.2 ST_SetRotation

ST_SetRotation — Bestimmt die Rotation des Rasters in Radiant.

Synopsis

float8 ST_SetRotation(raster rast, float8 rotation);

Beschreibung

Einheitliche Rotation des Rasters. Die Rotation wird in Radiant angegeben. Siehe [World File](#) für mehr Details.

Beispiele

```
SELECT
  ST_ScaleX(rast1), ST_ScaleY(rast1), ST_SkewX(rast1), ST_SkewY(rast1),
  ST_ScaleX(rast2), ST_ScaleY(rast2), ST_SkewX(rast2), ST_SkewY(rast2)
FROM (
  SELECT ST_SetRotation(rast, 15) AS rast1, rast as rast2 FROM dummy_rast
) AS foo;
```

st_scalex	st_scaley	st_skewx	st_skewy	
st_scalex	st_scaley	st_skewx	st_skewy	
-1.51937582571764	-2.27906373857646	1.95086352047135	1.30057568031423	↔
2	3	0	0	
-0.0379843956429411	-0.0379843956429411	0.0325143920078558	0.0325143920078558	↔
0.05	-0.05	0	0	

Siehe auch

[ST_Rotation](#), [ST_ScaleX](#), [ST_ScaleY](#), [ST_SkewX](#), [ST_SkewY](#)

9.7.3 ST_SetScale

ST_SetScale — Setzt die X- und Y-Größe der Pixel in den Einheiten des Koordinatenreferenzsystems. Entweder eine Zahl pro Pixel oder Breite und Höhe.

Synopsis

```
raster ST_SetScale(raster rast, float8 xy);
raster ST_SetScale(raster rast, float8 x, float8 y);
```

Beschreibung

Setzt die X- und Y-Größe der Pixel in den Einheiten des Koordinatenreferenzsystems. Entweder eine Zahl pro Pixel oder Breite und Höhe. X und Y werden als gleich groß angenommen, wenn nur eine Zahl übergeben wird.



Note

ST_SetScale unterscheidet sich von **ST_Rescale** dadurch, dass bei **ST_SetScale** der Raster nicht skaliert wird um mit der Rasterausdehnung übereinzustimmen. Es werden nur die Metadaten (oder die Georeferenz) des Rasters geändert, um eine ursprünglich falsch angegebene Skalierung zu korrigieren. **ST_Rescale** berechnet die Breite und Höhe eines Rasters so, dass er mit der geographischen Ausdehnung des Rasters übereinstimmt. **ST_SetScale** verändert weder die Breite noch die Höhe des Rasters.

Änderung: 2.0.0. Versionen von WKTRaster haben dies als "ST_SetPixelSizeY" bezeichnet. Dies wurde mit 2.0.0 geändert.

Beispiele

```
UPDATE dummy_rast
  SET rast = ST_SetScale(rast, 1.5)
WHERE rid = 2;
```

```
SELECT ST_ScaleX(rast) As pixx, ST_ScaleY(rast) As pixy, Box3D(rast) As newbox
FROM dummy_rast
```



```
WHERE rid = 2;
```

pixx	pixy	newbox
1.5	1.5	BOX(3427927.75 5793244 0, 3427935.25 5793251.5 0)

```
UPDATE dummy_rast
```

```
    SET rast = ST_SetScale(rast, 1.5, 0.55)
```

```
WHERE rid = 2;
```

```
SELECT ST_ScaleX(rast) As pixx, ST_ScaleY(rast) As pixy, Box3D(rast) As newbox
FROM dummy_rast
WHERE rid = 2;
```

pixx	pixy	newbox
1.5	0.55	BOX(3427927.75 5793244 0,3427935.25 5793247 0)

Siehe auch

[ST_ScaleX](#), [ST_ScaleY](#), [Box3D](#)

9.7.4 ST_SetSkew

ST_SetSkew — Setzt den georeferenzierten X- und Y-Versatz (oder den Rotationsparameter). Wenn nur ein Wert übergeben wird, werden X und Y auf den selben Wert gesetzt.

Synopsis

```
raster ST_SetSkew(raster rast, float8 skewxy);
raster ST_SetSkew(raster rast, float8 skewx, float8 skewy);
```

Beschreibung

Setzt den georeferenzierten X- und Y-Versatz (oder den Rotationsparameter). Wenn nur ein Wert übergeben wird, werden X und Y auf den selben Wert gesetzt. Siehe [World File](#) für weitere Details.

Beispiele

```
-- Beispiel 1
UPDATE dummy_rast SET rast = ST_SetSkew(rast,1,2) WHERE rid = 1;
SELECT rid, ST_SkewX(rast) As skewx, ST_SkewY(rast) As skewy,
       ST_GeoReference(rast) as georef
FROM dummy_rast WHERE rid = 1;
```

rid	skewx	skewy	georef
1	1	2	2.0000000000 : 2.0000000000 : 1.0000000000 : 3.0000000000 : 0.5000000000 : 0.5000000000

```
-- Beispiel 2 Beide auf die gleiche Zahl setzen:
UPDATE dummy_rast SET rast = ST_SetSkew(rast,0) WHERE rid = 1;
SELECT rid, ST_SkewX(rast) As skewx, ST_SkewY(rast) As skewy,
       ST_GeoReference(rast) as georef
FROM dummy_rast WHERE rid = 1;
```

rid	skewx	skewy	georef
1	0	0	2.0000000000 : 0.0000000000 : 0.0000000000 : 3.0000000000 : 0.5000000000 : 0.5000000000

Siehe auch

[ST_GeoReference](#), [ST_SetGeoReference](#), [ST_SkewX](#), [ST_SkewY](#)

9.7.5 ST_SetSRID

ST_SetSRID — Setzt die SRID eines Rasters auf einen bestimmten Ganzzahlwert. Die SRID wird in der Tabelle "spatial_ref_sys" definiert.

Synopsis

raster **ST_SetSRID**(raster rast, integer srid);

Beschreibung

Weist der SRID des Rasters einen bestimmten Ganzzahlwert zu.



Note

Diese Funktion führt keine Koordinatentransformation des Rasters durch - sie setzt nur die Metadaten, welche das Koordinatenreferenzsystem definieren, in dem der Raster vorliegt. Nützlich für spätere Koordinatentransformationen.

Siehe auch

Section [4.3.1](#), [ST_SRID](#)

9.7.6 ST_SetUpperLeft

ST_SetUpperLeft — Setzt den Wert der oberen linken Ecke des Rasters auf die projizierten X- und Y-Koordinaten.

Synopsis

raster **ST_SetUpperLeft**(raster rast, double precision x, double precision y);

Beschreibung

Setzt den Wert der oberen linken Ecke des Rasters auf die projizierten X- und Y-Koordinaten.

Beispiele

```
SELECT ST_SetUpperLeft(rast, -71.01, 42.37)
FROM dummy_rast
WHERE rid = 2;
```

Siehe auch

[ST_UpperLeftX](#), [ST_UpperLeftY](#)

9.7.7 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=NearestNeighbour, 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=NearestNeighbour, double precision maxerr=0.125, boolean usescale=true);
raster **ST_Resample**(raster rast, raster ref, boolean usescale, text algorithm=NearestNeighbour, double precision maxerr=0.125);

Beschreibung

Skaliert einen Raster mit einem bestimmten Algorithmus, neuen Dimensionen (width & height), einer Gitterecke (gridx & gridy) und über Parameter zur Georeferenzierung des Rasters (scalex, scaley, skewx & skewy), die angegeben oder von einem anderen Raster übernommen werden können. Wenn Sie einen Referenzraster verwenden, müssen beide Raster die selbe SRID haben.

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.

Wenn `maxerr` nicht angegeben ist, wird der maximale Fehler mit 0.125 Prozent angesetzt.



Note

Siehe [GDAL Warp resampling methods](#) für mehr Details.

Verfügbarkeit: 2.0.0 benötigt GDAL 1.6.1+

Änderung: 2.1.0 Der Parameter SRID wurde entfernt. Varianten mit einem Referenzraster setzen nicht länger die SRID des Referenzrasters ein. Verwenden Sie bitte `ST_Transform()` um einen Raster umzuprojizieren. Funktioniert mit Raster ohne SRID.

Beispiele

```
SELECT
    ST_Width(orig) AS orig_width,
    ST_Width(reduce_100) AS new_width
FROM (
    SELECT
        rast AS orig,
        ST_Resample(rast,100,100) AS reduce_100
    FROM aerials.boston
    WHERE ST_Intersects(rast,
        ST_Transform(
            ST_MakeEnvelope(-71.128, 42.2392,-71.1277, 42.2397, 4326),26986)
        )
    LIMIT 1
) AS foo;
```

orig_width	new_width
200	100

Siehe auch

[ST_Rescale](#), [ST_Resize](#), [ST_Transform](#)

9.7.8 ST_Rescale

ST_Rescale — Skaliert einen Raster indem lediglich der Maßstab (oder die Pixelgröße) angepasst wird. Neue Pixelwerte werden über NearestNeighbor, bilinear, kubisch, CubicSpline oder mit dem Lanczos-Filter errechnet. Die Standardeinstellung ist NearestNeighbor.

Synopsis

```
raster ST_Rescale(raster rast, double precision scalexy, text algorithm=NearestNeighbour, double precision maxerr=0.125);
raster ST_Rescale(raster rast, double precision scalex, double precision scaley, text algorithm=NearestNeighbour, double precision maxerr=0.125);
```

Beschreibung

Skaliert einen Raster indem lediglich der Maßstab (oder die Pixelgröße) 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.

`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](#)

Wenn `maxerr` nicht angegeben ist, wird der maximale Fehler mit 0.125 Prozent angesetzt.



Note

Siehe [GDAL Warp resampling methods](#) für mehr Details.

**Note**

ST_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+

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.

```
-- die ursprüngliche Pixelgröße des Rasters
SELECT ST_PixelWidth(ST_AddBand(ST_MakeEmptyRaster(100, 100, 0, 0, 0.001, -0.001, 0, 0, ↵
    4269), '8BUI'::text, 1, 0)) width

width
-----
0.001

-- die Pixelgröße des skalierten Rasters
SELECT ST_PixelWidth(ST_Rescale(ST_AddBand(ST_MakeEmptyRaster(100, 100, 0, 0, 0.001, ↵
    -0.001, 0, 0, 4269), '8BUI'::text, 1, 0), 0.0015)) width

width
-----
0.0015
```

Siehe auch

ST_Resize, **ST_Resample**, **ST_SetScale**, **ST_ScaleX**, **ST_ScaleY**, **ST_Transform**

9.7.9 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=NearestNeighbour, double precision maxerr=0.125);
raster ST_Reskew(raster rast, double precision skewx, double precision skewy, text algorithm=NearestNeighbour, double precision maxerr=0.125);
```

Beschreibung

Skaliert einen Raster, indem lediglich der Versatz (oder Rotationsparameter) angepasst wird. Neue Pixelwerte werden über NearestNeighbor, bilinear, kubisch, CubicSpline oder mit dem Lanczos-Filter errechnet. Die Standardeinstellung "NearestNeighbor" ist am schnellsten, ergibt aber die schlechteste Interpolation.

`skewx` und `skewy` legen den neuen Versatz fest.

Die Ausdehnung des Zielrasters erfasst die Ausdehnung des Ausgangsrasters.

Wenn `maxerr` nicht angegeben ist, wird der maximale Fehler mit 0.125 Prozent angesetzt.



Note

Siehe [GDAL Warp resampling methods](#) für mehr Details.



Note

`ST_Reskew` unterscheidet sich von `ST_SetSkew` darin, dass `ST_SetSkew` den Raster nicht skaliert um mit der Ausdehnung des Ausgangsrasters übereinzustimmen. `ST_SetSkew` ändert lediglich die Metadaten (oder die Georeferenz) des Rasters, um einen ursprünglich falsch angegebenen Versatz zu korrigieren. `ST_Reskew` ergibt einen Raster mit geänderter Breite und Höhe, da die Berechnung so durchgeführt wird, dass die geographische 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.

```
-- der ursprüngliche Raster, nicht rotiert
SELECT ST_Rotation(ST_AddBand(ST_MakeEmptyRaster(100, 100, 0, 0, 0.001, -0.001, 0, 0, 4269) ←
, '8BUI'::text, 1, 0));

-- result
0

-- die Rotation des versetzten Rasters
SELECT ST_Rotation(ST_Reskew(ST_AddBand(ST_MakeEmptyRaster(100, 100, 0, 0, 0.001, -0.001, ←
0, 0, 4269), '8BUI'::text, 1, 0), 0.0015));

-- result
-0.982793723247329
```

Siehe auch

[ST_Resample](#), [ST_Rescale](#), [ST_SetSkew](#), [ST_SetRotation](#), [ST_SkewX](#), [ST_SkewY](#), [ST_Transform](#)

9.7.10 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=NearestNeighbour, 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=NearestNeighbour, double precision maxerr=0.125);
raster ST_SnapToGrid(raster rast, double precision gridx, double precision gridy, double precision scalexy, text algorithm=NearestNeig
double precision maxerr=0.125);
```

Beschreibung

Skaliert einen Raster durch Fangen an einem Führungsgitter, welches durch einen beliebige Pixeleckpunkt (`gridx` & `gridy`) und eine optionale Pixelgröße (`scalex` & `scaley`) definiert ist. Neue Pixelwerte werden über `NearestNeighbor`, `bilinear`, `kubisch`, `CubicSpline` oder mit dem `Lanczos`-Filter errechnet. Die Standardeinstellung "`NearestNeighbor`" ist am schnellsten, ergibt aber die schlechteste Interpolation.

`gridx` und `gridy` bestimmen einen beliebigen Pixeleckpunkt des neuen Gitters. Dies muss nicht unbedingt die obere linke Ecke des Zielrasters sein, darf aber nicht innerhalb oder am Rand der Ausdehnung des Zielrasters liegen.

Optional können Sie die Pixelgröße des neuen Gitters mit `scalex` und `scaley` festlegen.

Die Ausdehnung des Zielrasters erfasst die Ausdehnung des Ausgangsrasters.

Wenn `maxerr` nicht angegeben ist, wird der maximale Fehler mit 0.125 Prozent angesetzt.



Note

Siehe [GDAL Warp resampling methods](#) für mehr Details.



Note

Verwenden Sie bitte [ST_Resample](#), wenn Sie eine bessere Kontrolle über die Gitterparameter benötigen.

Verfügbarkeit: 2.0.0 benötigt GDAL 1.6.1+

Änderung: 2.1.0 Funktioniert jetzt auch mit Raster ohne SRID

Beispiele

Ein einfaches Beispiel, bei dem ein Raster an einem sich geringfügig unterscheidenden Gitter gefangen wird.

```
-- das obere linke X des ursprünglichen Rasters
SELECT ST_UpperLeftX(ST_AddBand(ST_MakeEmptyRaster(10, 10, 0, 0, 0.001, -0.001, 0, 0, 4269) ←
, '8BUI'::text, 1, 0));
-- result
0

-- das obere linke X des Rasters nach dem Fangen
SELECT ST_UpperLeftX(ST_SnapToGrid(ST_AddBand(ST_MakeEmptyRaster(10, 10, 0, 0, 0.001, ←
-0.001, 0, 0, 4269), '8BUI'::text, 1, 0), 0.0002, 0.0002));

--result
-0.0008
```

Siehe auch

[ST_Resample](#), [ST_Rescale](#), [ST_UpperLeftX](#), [ST_UpperLeftY](#)

9.7.11 ST_Resize

`ST_Resize` — Ändert die Zellgröße - width/height - eines Rasters

Synopsis

raster **ST_Resize**(raster rast, integer width, integer height, text algorithm=NearestNeighbor, double precision maxerr=0.125);
 raster **ST_Resize**(raster rast, double precision percentwidth, double precision percentheight, text algorithm=NearestNeighbor, double precision maxerr=0.125);
 raster **ST_Resize**(raster rast, text width, text height, text algorithm=NearestNeighbor, double precision maxerr=0.125);

Beschreibung

Passt die Größe des Rasters an eine neue Breite/Höhe an. Die neue Breite/Höhe kann durch die genaue Anzahl der Pixel, oder durch einen Prozentsatz der Breite/Höhe des Rasters, angegeben werden. Die Ausdehnung des Zielrasters ist mit der Ausdehnung des Ausgangsrasters ident.

Neue Pixelwerte werden über NearestNeighbor, bilinear, kubisch, CubicSpline oder mit dem Lanczos-Filter errechnet. Die Standardeinstellung "NearestNeighbor" ist am schnellsten, erzeugt aber auch die schlechteste Interpolation.

Variante 1 erwartet die tatsächliche width/height des Ausgaberrasters.

Variante 2 erwartet Dezimalwerte zwischen null (0) und eins (1), welche das Verhältnis zur Pixelbreite und zur Pixelhöhe des Eingaberasters angeben.

Variante 3 nimmt entweder die tatsächliche Breite/Höhe des Zielrasters oder einen Prozentsatz der Breite/Höhe des Ausgangsrasters als Zeichenfolge ("20%") entgegen.

Verfügbarkeit: 2.1.0 benötigt GDAL 1.6.1+

Beispiele

```
WITH foo AS (
SELECT
    1 AS rid,
    ST_Resize(
        ST_AddBand(
            ST_MakeEmptyRaster(1000, 1000, 0, 0, 1, -1, 0, 0, 0)
            , 1, '8BUI', 255, 0
        )
        , '50%', '500') AS rast
UNION ALL
SELECT
    2 AS rid,
    ST_Resize(
        ST_AddBand(
            ST_MakeEmptyRaster(1000, 1000, 0, 0, 1, -1, 0, 0, 0)
            , 1, '8BUI', 255, 0
        )
        , 500, 100) AS rast
UNION ALL
SELECT
    3 AS rid,
    ST_Resize(
        ST_AddBand(
            ST_MakeEmptyRaster(1000, 1000, 0, 0, 1, -1, 0, 0, 0)
            , 1, '8BUI', 255, 0
        )
        , 0.25, 0.9) AS rast
), bar AS (
    SELECT rid, ST_Metadata(rast) AS meta, rast FROM foo
)
SELECT rid, (meta).* FROM bar
```


rid	upperleftx	upperlefty	width	height	scalex	scaley	skewx	skewy	srid	←
numbands										
1	0	0	500	500	1	-1	0	0	0	←
	1									
2	0	0	500	100	1	-1	0	0	0	←
	1									
3	0	0	250	900	1	-1	0	0	0	←
	1									
(3 rows)										

Siehe auch

[ST_Resample](#), [ST_Rescale](#), [ST_Reskew](#), [ST_SnapToGrid](#)

9.7.12 ST_Transform

ST_Transform — Projiziert einen Raster von einem bekannten Koordinatenreferenzsystem in ein anderes bekanntes Koordinatenreferenzsystem um. Die Optionen für die Skalierung sind `NearestNeighbor`, `Bilinear`, `Cubisch`, `CubicSpline` und der Lanczos-Filter, die Standardeinstellung ist `NearestNeighbor`.

Synopsis

```
raster ST_Transform(raster rast, integer srid, text algorithm=NearestNeighbor, double precision maxerr=0.125, double precision
scalex, double precision scaley);
raster ST_Transform(raster rast, integer srid, double precision scalex, double precision scaley, text algorithm=NearestNeighbor,
double precision maxerr=0.125);
raster ST_Transform(raster rast, raster alignto, text algorithm=NearestNeighbor, double precision maxerr=0.125);
```

Beschreibung

Projiziert einen Raster von einem bekannten Koordinatenreferenzsystem in ein anderes bekanntes Koordinatenreferenzsystem um. Verwendet den angegebenen Algorithmus für die Interpolation der Pixelwerte. Wenn kein Algorithmus angegeben ist, wird 'NearestNeighbor' verwendet. Wenn "maxerr" nicht angegeben ist, wird ein Prozentsatz von 0.125 angenommen.

Die Optionen für den Algorithmus sind: 'NearestNeighbor', 'Bilinear', 'Cubic', 'CubicSpline' und 'Lanczos'. Siehe [GDAL Warp resampling methods](#) für weitere Details.

`ST_Transform` wird oft mit `ST_SetSRID()` verwechselt. `ST_Transform` ändert die Koordinaten der Geometrie tatsächlich (und die Pixelwerte mittels "resampling") von einem Koordinatenreferenzsystem auf ein anderes, während `ST_SetSRID()` nur den Identifikator "SRID" des Rasters ändert.

Anders als die anderen Varianten, benötigt Variante 3 einen Referenzraster für den Parameter `alignto`. Der Raster wird in das Koordinatenreferenzsystem (SRID) des Referenzrasters transformiert und an dem Referenzraster ausgerichtet (`ST_SameAlignment = TRUE`).

Note



Falls Sie herausfinden, dass die Koordinatentransformation nicht richtig unterstützt wird, müssen Sie vermutlich die Umgebungsvariable `PROJSO` auf jene Projektionsbibliothek ".so" oder ".dll" setzen, die von Ihrer PostGIS Installation verwendet wird. Hierbei muss nur der Name der Datei angegeben werden. Auf Windows zum Beispiel würden Sie unter Control Panel -> System -> Environment Variables eine Systemvariable mit dem Namen `PROJSO` hinzufügen und auf `libproj.dll` setzen (falls Sie Proj 4.6.1 verwenden). Nach dieser Änderung müssen Sie den PostgreSQL-Dienst/Dämon neu starten.

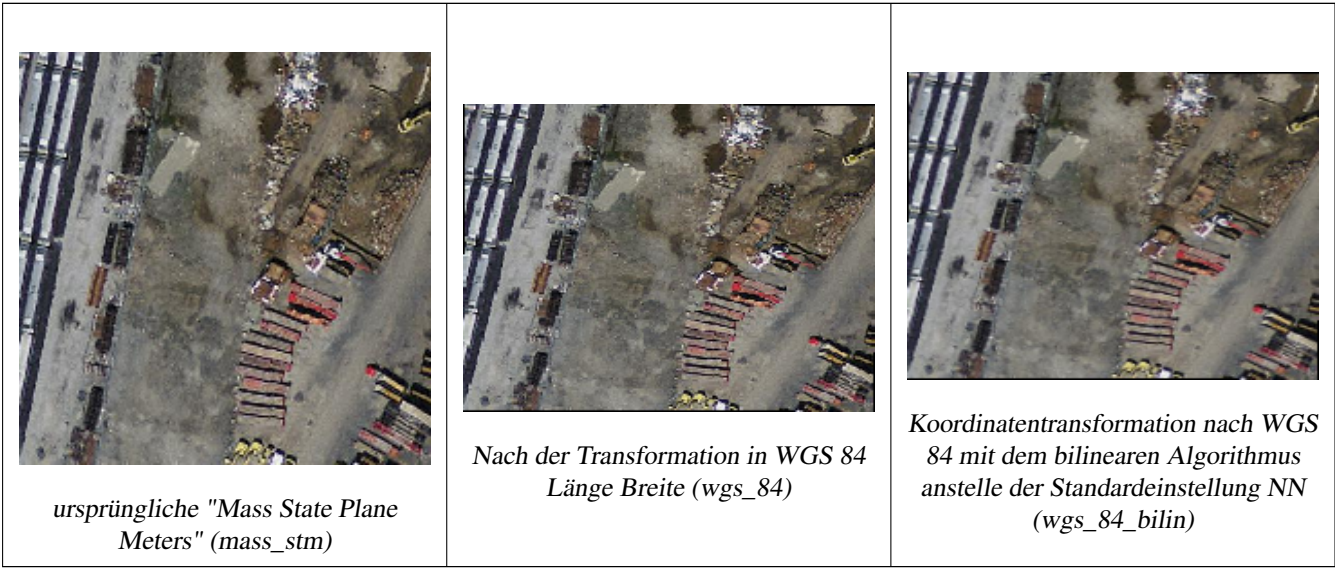
Verfügbarkeit: 2.0.0 benötigt GDAL 1.6.1+

Erweiterung: 2.1.0 Variante `ST_Transform(rast, alignto)` hinzugefügt

Beispiele

```
SELECT ST_Width(mass_stm) As w_before, ST_Width(wgs_84) As w_after,
       ST_Height(mass_stm) As h_before, ST_Height(wgs_84) As h_after
FROM
  ( SELECT rast As mass_stm, ST_Transform(rast,4326) As wgs_84
    , ST_Transform(rast,4326, 'Bilinear') AS wgs_84_bilin
    FROM aerials.o_2_boston
      WHERE ST_Intersects(rast,
        ST_Transform(ST_MakeEnvelope(-71.128, 42.2392,-71.1277, 42.2397, 4326),26986) )
    LIMIT 1) As foo;
```

w_before	w_after	h_before	h_after
200	228	200	170



Beispiele: Variante 3

Im Folgenden wird der Unterschied zwischen der Verwendung von ST_Transform(raster, srid) und ST_Transform(raster, align)to) gezeigt

```
WITH foo AS (
  SELECT 0 AS rid, ST_AddBand(ST_MakeEmptyRaster(2, 2, -500000, 600000, 100, -100, 0, 0, 2163), 1, '16BUI', 1, 0) AS rast UNION ALL
  SELECT 1, ST_AddBand(ST_MakeEmptyRaster(2, 2, -499800, 600000, 100, -100, 0, 0, 2163), 1, '16BUI', 2, 0) AS rast UNION ALL
  SELECT 2, ST_AddBand(ST_MakeEmptyRaster(2, 2, -499600, 600000, 100, -100, 0, 0, 2163), 1, '16BUI', 3, 0) AS rast UNION ALL

  SELECT 3, ST_AddBand(ST_MakeEmptyRaster(2, 2, -500000, 599800, 100, -100, 0, 0, 2163), 1, '16BUI', 10, 0) AS rast UNION ALL
  SELECT 4, ST_AddBand(ST_MakeEmptyRaster(2, 2, -499800, 599800, 100, -100, 0, 0, 2163), 1, '16BUI', 20, 0) AS rast UNION ALL
  SELECT 5, ST_AddBand(ST_MakeEmptyRaster(2, 2, -499600, 599800, 100, -100, 0, 0, 2163), 1, '16BUI', 30, 0) AS rast UNION ALL

  SELECT 6, ST_AddBand(ST_MakeEmptyRaster(2, 2, -500000, 599600, 100, -100, 0, 0, 2163), 1, '16BUI', 100, 0) AS rast UNION ALL
```

```

SELECT 7, ST_AddBand(ST_MakeEmptyRaster(2, 2, -499800, 599600, 100, -100, 0, 0, ↵
2163), 1, '16BUI', 200, 0) AS rast UNION ALL
SELECT 8, ST_AddBand(ST_MakeEmptyRaster(2, 2, -499600, 599600, 100, -100, 0, 0, ↵
2163), 1, '16BUI', 300, 0) AS rast
), bar AS (
SELECT
    ST_Transform(rast, 4269) AS alignto
FROM foo
LIMIT 1
), baz AS (
SELECT
    rid,
    rast,
    ST_Transform(rast, 4269) AS not_aligned,
    ST_Transform(rast, alignto) AS aligned
FROM foo
CROSS JOIN bar
)
SELECT
    ST_SameAlignment(rast) AS rast,
    ST_SameAlignment(not_aligned) AS not_aligned,
    ST_SameAlignment(aligned) AS aligned
FROM baz

rast | not_aligned | aligned
-----+-----+-----
t   | f           | t

```

Siehe auch

[ST_Transform](#), [ST_SetSRID](#)

9.8 Editoren für Rasterbänder

9.8.1 ST_SetBandNoDataValue

ST_SetBandNoDataValue — Setzt den NODATA Wert eines Bandes. Wenn kein Band angegeben ist, wird Band 1 angenommen. Falls ein Band keinen NODATA Wert aufweisen soll, übergeben Sie bitte für den Parameter "nodatavalue" NULL.

Synopsis

```

raster ST_SetBandNoDataValue(raster rast, double precision nodatavalue);
raster ST_SetBandNoDataValue(raster rast, integer band, double precision nodatavalue, boolean forcechecking=false);

```

Beschreibung

Setzt den NODATA Wert eines Bandes. Wenn kein Band angegeben ist, wird Band 1 angenommen. Dies beeinflusst die Ergebnisse von [ST_Polygon](#), [ST_DumpAsPolygons](#) und die Funktionen [ST_PixelAs...](#)().

Beispiele

```

-- den NODATA-Wert nur für das erste Band setzen
UPDATE dummy_rast
    SET rast = ST_SetBandNoDataValue(rast,1, 254)
WHERE rid = 2;

-- Den NODATA Wert der Bänder 1,2,3 ändern
UPDATE dummy_rast
    SET rast =
        ST_SetBandNoDataValue(
            ST_SetBandNoDataValue(
                ST_SetBandNoDataValue(
                    rast,1, 254)
                ,2,99),
            3,108)
    WHERE rid = 2;

-- NODATA Werte ausblenden. Dadurch wird gewährleistet, dass sämtliche Pixel von allen ↵
    Funktionsberechnungen herangezogen werden
UPDATE dummy_rast
    SET rast = ST_SetBandNoDataValue(rast,1, NULL)
WHERE rid = 2;

```

Siehe auch

[ST_BandNoDataValue](#), [ST_NumBands](#)

9.8.2 ST_SetBandIsNoData

`ST_SetBandIsNoData` — Setzt die Flag "isnodata" für das Band auf TRUE.

Synopsis

raster `ST_SetBandIsNoData`(raster rast, integer band=1);

Beschreibung

Setzt die Flag "isnodata" des Bandes auf TRUE. Wenn kein Band angegeben ist, wird Band 1 angenommen. Diese Funktion sollte nur aufgerufen werden, wenn sich die Flag verändert hat. Dies ist der Fall, wenn sich die Ergebnisse von [ST_BandIsNoData](#) unterscheiden, da einmal mit TRUE als letzten Übergabewert und ein anderes mal ohne diesen aufgerufen wurde.

Verfügbarkeit: 2.0.0

Beispiele

```

-- Eine dummy Tabelle mit einer Rasterspalte erstellen
create table dummy_rast (rid integer, rast raster);

-- Einen Raster mit zwei Bändern hinzufügen, ein Pixel pro Band. Im ersten Band, ↵
    nodatavalue = pixel value = 3.
-- Im zweiten Band, nodatavalue = 13, pixel value = 4
insert into dummy_rast values(1,
(
'01' -- little endian (uint8 ndr)
||
'0000' -- version (uint16 0)

```

```

||
'0200' -- nBands (uint16 0)
||
'17263529ED684A3F' -- scaleX (float64 0.000805965234044584)
||
'F9253529ED684ABF' -- scaleY (float64 -0.00080596523404458)
||
'1C9F33CE69E352C0' -- ipX (float64 -75.5533328537098)
||
'718F0E9A27A44840' -- ipY (float64 49.2824585505576)
||
'ED50EB853EC32B3F' -- skewX (float64 0.000211812383858707)
||
'7550EB853EC32B3F' -- skewY (float64 0.000211812383858704)
||
'E6100000' -- SRID (int32 4326)
||
'0100' -- width (uint16 1)
||
'0100' -- height (uint16 1)
||
'4' -- hasnodatavalue set to true, isnodata value set to false (when it should be true)
||
'2' -- first band type (4BUI)
||
'03' -- novalue==3
||
'03' -- pixel(0,0)==3 (same that nodata)
||
'0' -- hasnodatavalue set to false
||
'5' -- second band type (16BSI)
||
'0D00' -- novalue==13
||
'0400' -- pixel(0,0)==4
)::raster
);

select st_bandisnodata(rast, 1) from dummy_rast where rid = 1; -- Expected false
select st_bandisnodata(rast, 1, TRUE) from dummy_rast where rid = 1; -- Expected true

-- Die Flag "isnodata" ist versaut. Wir setzen sie auf TRUE
update dummy_rast set rast = st_setbandisnodata(rast, 1) where rid = 1;

select st_bandisnodata(rast, 1) from dummy_rast where rid = 1; -- Expected true

```

Siehe auch

[ST_BandNoDataValue](#), [ST_NumBands](#), [ST_SetBandNoDataValue](#), [ST_BandIsNoData](#)

9.9 Rasterband Statistik und Analytik

9.9.1 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);
bigint ST_Count(text rastertable, text rastercolumn, integer nband=1, boolean exclude_nodata_value=true);
bigint ST_Count(text rastertable, text rastercolumn, boolean exclude_nodata_value);
```

Beschreibung

Gibt die Anzahl der Pixel für ein Band eines Rasters oder eines Raster-Coverage zurück. Wenn kein Band angegeben ist, wird für nband 1 vorausgesetzt.



Note

Wenn der Parameter `exclude_nodata_value` auf TRUE gesetzt ist, werden nur die Pixel des Rasters gezählt, deren Werte ungleich `nodata` sind. Setzen Sie bitte `exclude_nodata_value` auf FALSE um die Anzahl aller Pixel zu erhalten

Verfügbarkeit: 2.0.0



Warning

Ab 2.0.0 sind die Varianten von `ST_Count(rastertable, rastercolumn, ...)` überholt. Verwenden Sie bitte stattdessen `ST_CountAgg`.

Beispiele

```
-- das Beispiel zählt einmal die Pixel ohne den Wert 249 und einmal alle Pixel. --
SELECT rid, ST_Count(ST_SetBandNoDataValue(rast,249)) As exclude_nodata,
       ST_Count(ST_SetBandNoDataValue(rast,249),false) As include_nodata
FROM dummy_rast WHERE rid=2;
```

rid	exclude_nodata	include_nodata
2	23	25

Siehe auch

[ST_CountAgg](#), [ST_SummaryStats](#), [ST_SetBandNoDataValue](#)

9.9.2 ST_CountAgg

ST_CountAgg — Aggregatfunktion. Gibt die Anzahl der Pixel in einem bestimmten Band der Raster aus. Wenn kein Band angegeben ist, wird Band 1 angenommen. Wenn "exclude_nodata_value" TRUE ist, werden nur die Pixel ohne NODATA Werte gezählt.

Synopsis

```
bigint ST_CountAgg(raster rast, integer nband, boolean exclude_nodata_value, double precision sample_percent);
bigint ST_CountAgg(raster rast, integer nband, boolean exclude_nodata_value);
bigint ST_CountAgg(raster rast, boolean exclude_nodata_value);
```

Beschreibung

Gibt die Anzahl der Pixel in einem bestimmten Band der Raster aus. Wenn kein Band angegeben ist, wird `nband` standardmäßig auf 1 gesetzt.

Wenn der Parameter `exclude_nodata_value` auf `TRUE` gesetzt ist, werden nur die Pixel des Rasters gezählt, deren Werte ungleich `NODATA` sind. Setzen Sie bitte `exclude_nodata_value` auf `FALSE` um die Anzahl aller Pixel zu erhalten

Standardmäßig werden alle Pixel abgetastet. Um eine schnellere Rückmeldung zu bekommen, können Sie den Parameter `sample_percent` auf einen Wert zwischen null (0) und eins (1) setzen.

Verfügbarkeit: 2.2.0

Beispiele

```
WITH foo AS (
    SELECT
        rast.rast
    FROM (
        SELECT ST_SetValue(
            ST_SetValue(
                ST_SetValue(
                    ST_AddBand(
                        ST_MakeEmptyRaster(10, 10, 10, 10, 2, 2, 0, 0,
                        0,0)
                        , 1, '64BF', 0, 0
                    )
                    , 1, 1, 1, -10
                )
                , 1, 5, 4, 0
            )
            , 1, 5, 5, 3.14159
        ) AS rast
    ) AS rast
    FULL JOIN (
        SELECT generate_series(1, 10) AS id
    ) AS id
    ON 1 = 1
)
SELECT
    ST_CountAgg(rast, 1, TRUE)
FROM foo;

 st_countagg
-----
          20
(1 row)
```

Siehe auch

[ST_Count](#), [ST_SummaryStats](#), [ST_SetBandNoDataValue](#)

9.9.3 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);
 SETOF record **ST_Histogram**(text rastertable, text rastercolumn, integer nband, integer bins, boolean right);
 SETOF record **ST_Histogram**(text rastertable, text rastercolumn, integer nband, boolean exclude_nodata_value, integer bins, boolean right);
 SETOF record **ST_Histogram**(text rastertable, text rastercolumn, integer nband=1, boolean exclude_nodata_value=true, integer bins=autocomputed, double precision[] width=NULL, boolean right=false);
 SETOF record **ST_Histogram**(text rastertable, text rastercolumn, integer nband=1, integer bins, double precision[] width=NULL, boolean right=false);

Beschreibung

Gibt Datensätze aus, die das Minimum, das Maximum, die Anzahl und die Prozent der Werte des Rasterbandes für jede Klasse enthalten. Wenn kein Band angegeben ist, wird `nband` standardmäßig auf 1 gesetzt.



Note

Standardmäßig werden nur jene Pixelwerte berücksichtigt, die nicht den Wert `NODATA` haben. Setzen Sie bitte `exclude_nodata_value` auf `FALSE` um die Anzahl sämtlicher Pixel zu erhalten.

width double precision[] Breite: ein Feld das die Breite für jede Kategorie/Klasse angibt. Wenn die Anzahl der Klassen größer ist als die Anzahl der Breiten, werden die Breiten wiederholt.

Beispiel: 9 Klassen, die Breiten sind [a, b, c] und werden als [a, b, c, a, b, c, a, b, c] übergeben.

bins integer Anzahl der Klassen - die Anzahl der Datensätze die von der Funktion ausgegeben werden. Wenn die Anzahl der Klassen nicht angegeben ist, wird sie automatisch berechnet.

right boolean Berechnet das Histogramm von rechts anstatt von links (Standardwert). Dies ändert das Auswahlkriterium für einen Wert `x` von [a,b) auf (a,b].

Verfügbarkeit: 2.0.0

Beispiel: Einzelne Rasterkachel - Berechnung des Histogramm für die Bänder 1, 2, 3 und automatische Einteilung der Wertemenge in Klassen

```
SELECT band, (stats).*
FROM (SELECT rid, band, ST_Histogram(rast, band) As stats
      FROM dummy_rast CROSS JOIN generate_series(1,3) As band
      WHERE rid=2) As foo;
```

band	min	max	count	percent
1	249	250	2	0.08

1		250		251		2		0.08
1		251		252		1		0.04
1		252		253		2		0.08
1		253		254		18		0.72
2		78		113.2		11		0.44
2		113.2		148.4		4		0.16
2		148.4		183.6		4		0.16
2		183.6		218.8		1		0.04
2		218.8		254		5		0.2
3		62		100.4		11		0.44
3		100.4		138.8		5		0.2
3		138.8		177.2		4		0.16
3		177.2		215.6		1		0.04
3		215.6		254		4		0.16

Beispiel: Nur Band 2 und 6 Klassen

```
SELECT (stats).*
FROM (SELECT rid, ST_Histogram(rast, 2,6) As stats
      FROM dummy_rast
      WHERE rid=2) As foo;
```

min		max		count		percent
78		107.333333		9		0.36
107.333333		136.666667		6		0.24
136.666667		166		0		0
166		195.333333		4		0.16
195.333333		224.666667		1		0.04
224.666667		254		5		0.2

(6 rows)

-- Das gleiche Beispiel wie vorher, aber der Wertebereich der Pixel ist für jede Klasse explizit angegeben. ↵

```
SELECT (stats).*
FROM (SELECT rid, ST_Histogram(rast, 2,6,ARRAY[0.5,1,4,100,5]) As stats
      FROM dummy_rast
      WHERE rid=2) As foo;
```

min		max		count		percent
78		78.5		1		0.08
78.5		79.5		1		0.04
79.5		83.5		0		0
83.5		183.5		17		0.0068
183.5		188.5		0		0
188.5		254		6		0.003664

(6 rows)

Siehe auch

[ST_Count](#), [ST_SummaryStats](#), [ST_SummaryStatsAgg](#)

9.9.4 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);
SETOF record ST_Quantile(text rastertable, text rastercolumn, integer nband=1, boolean exclude_nodata_value=true, double
precision[] quantiles=NULL);
SETOF record ST_Quantile(text rastertable, text rastercolumn, integer nband, double precision[] quantiles);

```

Beschreibung

Berechnet die Quantile eines Rasters oder einer Rastercoverage Tabelle im Kontext von Stichproben oder Bevölkerung. Dadurch kann untersucht werden, ob ein Wert bei 25%, 50% oder 75% Perzentil des Rasters liegt.



Note

Wenn `exclude_nodata_value` auf `FALSE` gesetzt ist, werden auch die Pixel mit NODATA Werten gezählt.

Verfügbarkeit: 2.0.0

Beispiele

```

UPDATE dummy_rast SET rast = ST_SetBandNoDataValue(rast,249) WHERE rid=2;
--Das Beispiel berücksichtigt nur die Pixel von Band 1, die nicht den Wert 249 haben und in ↵
den genannten Quantilen liegen --

```

```

SELECT (pvq).*
FROM (SELECT ST_Quantile(rast, ARRAY[0.25,0.75]) As pvq
      FROM dummy_rast WHERE rid=2) As foo
ORDER BY (pvq).quantile;

```

quantile	value
0.25	253
0.75	254

```

SELECT ST_Quantile(rast, 0.75) As value
FROM dummy_rast WHERE rid=2;

```

value
254

```

--ein Beispiel aus der Praxis. Die Quantile aller Pixel in Band 2 die eine Geometrie ↵
schneiden
SELECT rid, (ST_Quantile(rast,2)).* As pvc
FROM o_4_boston
WHERE ST_Intersects(rast,
ST_GeomFromText('POLYGON((224486 892151,224486 892200,224706 892200,224706 ↵
892151,224486 892151))',26986)

```

```
)
ORDER BY value, quantile,rid
;
```

rid	quantile	value
1	0	0
2	0	0
14	0	1
15	0	2
14	0.25	37
1	0.25	42
15	0.25	47
2	0.25	50
14	0.5	56
1	0.5	64
15	0.5	66
2	0.5	77
14	0.75	81
15	0.75	87
1	0.75	94
2	0.75	106
14	1	199
1	1	244
2	1	255
15	1	255

Siehe auch

[ST_Count](#), [ST_SummaryStats](#), [ST_SummaryStatsAgg](#), [ST_SetBandNoDataValue](#)

9.9.5 ST_SummaryStats

ST_SummaryStats — Gibt eine zusammenfassende Statistik aus, bestehend aus der Anzahl, der Summe, dem arithmetischen Mittel, der Standardabweichung, dem Minimum und dem Maximum der Werte eines Rasterbandes oder eines Rastercoverage. Wenn kein Band angegeben ist, wird Band 1 angenommen.

Synopsis

```
summarystats ST_SummaryStats(raster rast, boolean exclude_nodata_value);
summarystats ST_SummaryStats(raster rast, integer nband, boolean exclude_nodata_value);
summarystats ST_SummaryStats(text rastertable, text rastercolumn, boolean exclude_nodata_value);
summarystats ST_SummaryStats(text rastertable, text rastercolumn, integer nband=1, boolean exclude_nodata_value=true);
```

Beschreibung

Gibt **summarystats** aus, bestehend aus der Anzahl, der Summe, dem arithmetischen Mittel, der Standardabweichung, dem Minimum und dem Maximum der Werte eines Rasterbandes oder eines Rastercoverage. Wenn kein Band angegeben ist, wird Band 1 angenommen.



Note

Standardmäßig werden nur jene Pixelwerte berücksichtigt, die nicht den Wert `NODATA` haben. Setzen Sie bitte `exclude_nodata_value` auf `FALSE` um die Anzahl sämtlicher Pixel zu erhalten.

**Note**

Standardmäßig werden alle Pixel abgetastet. Um eine schnellere Rückmeldung zu bekommen, können Sie den Parameter `sample_percent` auf einen Wert kleiner als 1 setzen.

Verfügbarkeit: 2.0.0

**Warning**

Die Varianten von `ST_SummaryStats(rastertable, rastercolumn, ...)` sind mit 2.2.0 überholt. Verwenden Sie bitte stattdessen **`ST_SummaryStatsAgg`**.

Beispiel: Einzelne Rasterkachel

```
SELECT rid, band, (stats).*
FROM (SELECT rid, band, ST_SummaryStats(rast, band) As stats
      FROM dummy_rast CROSS JOIN generate_series(1,3) As band
      WHERE rid=2) As foo;
```

rid	band	count	sum	mean	stddev	min	max
2	1	23	5821	253.086957	1.248061	250	254
2	2	25	3682	147.28	59.862188	78	254
2	3	25	3290	131.6	61.647384	62	254

Beispiel: Zusammenfassen der Pixel, die interessante Bauwerke schneiden

Dieses Beispiel benötigte 574ms in PostGIS unter Windows 64-Bit, mit allen Bauwerken und Luftbildkacheln von Boston (Kacheln jeweils 150x150 Pixel ~ 134.000 Kacheln; ~ 102.000 Datensätze mit Bauwerken)

```
WITH
-- Geoobjekte, die von Interesse sind
feat AS (SELECT gid As building_id, geom_26986 As geom FROM buildings AS b
        WHERE gid IN(100, 103,150)
        ),
-- schneidet Band 2 der Rasterkacheln an den Grenzen der Gebäuden aus
-- anschließend wird die Statistik "stats" für die ausgeschnittenen Regionen errechnet
(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
      )
-- schlussendlich wird die Statistik zusammengefasst
SELECT building_id, SUM(count) As num_pixels
      , MIN(min) As min_pval
      , MAX(max) As max_pval
      , SUM(mean*count)/SUM(count) As avg_pval
      FROM b_stats
WHERE count
> 0
      GROUP BY building_id
      ORDER BY building_id;
building_id | num_pixels | min_pval | max_pval | avg_pval
-----+-----+-----+-----+-----
```

100	1090	1	255	61.0697247706422
103	655	7	182	70.5038167938931
150	895	2	252	185.642458100559

Beispiel: Rastercoverage

```
-- die Statistik "stats" für jedes Band --
SELECT band, (stats).*
FROM (SELECT band, ST_SummaryStats('o_4_boston','rast', band) As stats
      FROM generate_series(1,3) As band) As foo;
```

band	count	sum	mean	stddev	min	max
1	8450000	725799	82.7064349112426	45.6800222638537	0	255
2	8450000	700487	81.4197705325444	44.2161184161765	0	255
3	8450000	575943	74.682739408284	44.2143885481407	0	255

```
-- Eine Tabelle erhält man in kürzerer Zeit, wenn die Stichprobe auf weniger als 100% ←
  gesetzt wird
-- Hier setzen wir die Stichprobe auf 25%, wodurch wir eine wesentlich schnellere Antwort ←
  erhalten
SELECT band, (stats).*
FROM (SELECT band, ST_SummaryStats('o_4_boston','rast', band,true,0.25) As stats
      FROM generate_series(1,3) As band) As foo;
```

band	count	sum	mean	stddev	min	max
1	2112500	180686	82.6890480473373	45.6961043857248	0	255
2	2112500	174571	81.448503668639	44.2252623171821	0	255
3	2112500	144364	74.6765884023669	44.2014869384578	0	255

Siehe auch

[summarystats](#), [ST_SummaryStatsAgg](#), [ST_Count](#), [ST_Clip](#)

9.9.6 ST_SummaryStatsAgg

ST_SummaryStatsAgg — Aggregatfunktion. Gibt eine zusammenfassende Statistik aus, die aus der Anzahl, der Summe, dem arithmetischen Mittel, dem Minimum und dem Maximum der Werte eines bestimmten Bandes eines Rastersatzes besteht. Wenn kein Band angegeben ist, wird Band 1 angenommen.

Synopsis

```
summarystats ST_SummaryStatsAgg(raster rast, integer nband, boolean exclude_nodata_value, double precision sample_percent);
summarystats ST_SummaryStatsAgg(raster rast, boolean exclude_nodata_value, double precision sample_percent);
summarystats ST_SummaryStatsAgg(raster rast, integer nband, boolean exclude_nodata_value);
```

Beschreibung

Gibt **summarystats** aus, bestehend aus der Anzahl, der Summe, dem arithmetischen Mittel, der Standardabweichung, dem Minimum und dem Maximum der Werte eines Rasterbandes oder eines Rastercoverage. Wenn kein Band angegeben ist, wird Band 1 angenommen.

**Note**

Standardmäßig werden nur jene Pixelwerte berücksichtigt, die nicht NODATA sind. Setzen Sie bitte `exclude_nodata_value` auf FALSE um die Anzahl sämtlicher Pixel zu erhalten.

**Note**

Standardmäßig werden alle Pixel abgetastet. Um eine schnellere Rückmeldung zu bekommen, können Sie den Parameter `sample_percent` auf einen Wert zwischen 0 und 1 setzen.

Verfügbarkeit: 2.2.0

Beispiele

```
WITH foo AS (
  SELECT
    rast.rast
  FROM (
    SELECT ST_SetValue(
      ST_SetValue(
        ST_SetValue(
          ST_AddBand(
            ST_MakeEmptyRaster(10, 10, 10, 10, 2, 2, 0, 0,
                                0,0),
            1, '64BF', 0, 0
          ),
          1, 1, 1, -10
        ),
        1, 5, 4, 0
      ),
      1, 5, 5, 3.14159
    ) AS rast
  ) AS rast
  FULL JOIN (
    SELECT generate_series(1, 10) AS id
  ) AS id
  ON 1 = 1
)
SELECT
  (stats).count,
  round((stats).sum::numeric, 3),
  round((stats).mean::numeric, 3),
  round((stats).stddev::numeric, 3),
  round((stats).min::numeric, 3),
  round((stats).max::numeric, 3)
FROM (
  SELECT
    ST_SummaryStatsAgg(rast, 1, TRUE, 1) AS stats
  FROM foo
) bar;
```

count	round	round	round	round	round
20	-68.584	-3.429	6.571	-10.000	3.142

(1 row)

Siehe auch

[summarystats](#), [ST_SummaryStats](#), [ST_Count](#), [ST_Clip](#)

9.9.7 ST_ValueCount

ST_ValueCount — 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.

Synopsis

```
SETOF record ST_ValueCount(raster rast, integer nband=1, boolean exclude_nodata_value=true, double precision[] searchvalues=NULL, double precision roundto=0, double precision OUT value, integer OUT count);
SETOF record ST_ValueCount(raster rast, integer nband, double precision[] searchvalues, double precision roundto=0, double precision OUT value, integer OUT count);
SETOF record ST_ValueCount(raster rast, double precision[] searchvalues, double precision roundto=0, double precision OUT value, integer OUT count);
bigint ST_ValueCount(raster rast, double precision searchvalue, double precision roundto=0);
bigint ST_ValueCount(raster rast, integer nband, boolean exclude_nodata_value, double precision searchvalue, double precision roundto=0);
bigint ST_ValueCount(raster rast, integer nband, double precision searchvalue, double precision roundto=0);
SETOF record ST_ValueCount(text rastertable, text rastercolumn, integer nband=1, boolean exclude_nodata_value=true, double precision[] searchvalues=NULL, double precision roundto=0, double precision OUT value, integer OUT count);
SETOF record ST_ValueCount(text rastertable, text rastercolumn, double precision[] searchvalues, double precision roundto=0, double precision OUT value, integer OUT count);
SETOF record ST_ValueCount(text rastertable, text rastercolumn, integer nband, double precision[] searchvalues, double precision roundto=0, double precision OUT value, integer OUT count);
bigint ST_ValueCount(text rastertable, text rastercolumn, integer nband, boolean exclude_nodata_value, double precision searchvalue, double precision roundto=0);
bigint ST_ValueCount(text rastertable, text rastercolumn, double precision searchvalue, double precision roundto=0);
bigint ST_ValueCount(text rastertable, text rastercolumn, integer nband, double precision searchvalue, double precision roundto=0);
```

Beschreibung

Gibt Datensätze mit den Spalten `value` und `count` aus, welche die Pixelwerte und die Anzahl der Pixel im angegebenen Band der Rasterkachel oder des Rastercoverage enthalten.

Wenn kein Band angegeben ist, wird `nband` standardmäßig auf 1 gesetzt. Wenn keine `searchvalues` angegeben sind, werden alle Pixelwerte des Rasters oder des Rastercoverage ausgegeben. Wenn nur ein Suchwert angegeben ist, wird eine Ganzzahl ausgegeben - anstelle von Datensätzen mit der Pixelanzahl eines jeden Pixelwertes für das Bandes.

**Note**

Wenn `exclude_nodata_value` auf FALSE gesetzt ist, werden auch die Pixel mit NODATA Werten gezählt.

Verfügbarkeit: 2.0.0

Beispiele

```
UPDATE dummy_rast SET rast = ST_SetBandNoDataValue(rast,249) WHERE rid=2;
--Dieses Beispiel zählt die Anzahl der Pixel von Band 1, die nicht den Wert 249 haben. --
```

```
SELECT (pvc).*
FROM (SELECT ST_ValueCount(rast) As pvc
      FROM dummy_rast WHERE rid=2) As foo
      ORDER BY (pvc).value;
```

value	count
250	2
251	1
252	2
253	6
254	12

```
-- Dieses Beispiel zählt alle Pixel von Band 1, inklusive 249 --
```

```
SELECT (pvc).*
FROM (SELECT ST_ValueCount(rast,1,false) As pvc
      FROM dummy_rast WHERE rid=2) As foo
      ORDER BY (pvc).value;
```

value	count
249	2
250	2
251	1
252	2
253	6
254	12

```
-- Dieses Beispiel zählt nur die Pixel mit NODATA Werten im Band 2
```

```
SELECT (pvc).*
FROM (SELECT ST_ValueCount(rast,2) As pvc
      FROM dummy_rast WHERE rid=2) As foo
      ORDER BY (pvc).value;
```

value	count
78	1
79	1
88	1
89	1
96	1
97	1
98	1
99	2
112	2

```
:
```

```
--Ein Beispiel aus der Praxis. Zählt alle Pixel eines Luftbildrasters in Band 2, die eine ↵
      Geometrie schneiden
```

```
-- und gibt nur jene Pixelwerte des Bandes aus, deren Anzahl
> 500 ist
```

```
SELECT (pvc).value, SUM((pvc).count) As total
FROM (SELECT ST_ValueCount(rast,2) As pvc
      FROM o_4_boston
```

```
      WHERE ST_Intersects(rast,
        ST_GeomFromText('POLYGON((224486 892151,224486 892200,224706 892200,224706 ↵
          892151,224486 892151))',26986)
      )
```



```

    ) As foo
  GROUP BY (pvc).value
  HAVING SUM((pvc).count)
> 500
  ORDER BY (pvc).value;

```

```

value | total
-----+-----
    51 |   502
    54 |   521

```

```

-- Die Anzahl der Pixel pro Rasterkachel, die den Wert 100 haben und deren Kacheln eine
-- bestimmte Geometrie schneidet --
SELECT rid, ST_ValueCount(rast,2,100) As count
  FROM o_4_boston
  WHERE ST_Intersects(rast,
    ST_GeomFromText('POLYGON((224486 892151,224486 892200,224706 892200,224706
    892151,224486 892151))',26986)
  ) ;

```

```

rid | count
-----+-----
   1 |     56
   2 |     95
  14 |     37
  15 |     64

```

Siehe auch

[ST_Count](#), [ST_SetBandNoDataValue](#)

9.10 Ausgabe von Rastern

9.10.1 ST_AsBinary

ST_AsBinary — Gibt die Well-known-Binary (WKB) Darstellung eines Rasters zurück.

Synopsis

bytea **ST_AsBinary**(raster rast, boolean outasin=FALSE);

Beschreibung

Gibt den Raster in binärer Darstellung zurück. Wenn `outasin` TRUE ist, werden Bänder die außerhalb der Datenbank liegen (out-db) gleich behandelt wie Bänder, die direkt in der Datenbank gespeichert (in-db) sind. Siehe "raster/doc/RFC2-WellKnownBinaryFormat" im Quellverzeichnis von PostGIS für Details zu dieser Darstellung.

Dies ist nützlich, um mit einem "binary" Cursor die Daten direkt als Binärdatei von der Datenbank abzurufen - ohne Konvertierung in eine Textdarstellung.



Note

Standardmäßig ist in der WKB-Ausgabe der Pfad der out-db Bänder enthalten. Wenn ein Client keinen Zugriff auf die Rasterdatei eines "out-db" Bandes hat, können Sie `outasin` auf TRUE setzen.

Erweiterung: 2.1.0 `outasin` hinzugefügt


```
WHERE rast && ST_MakeEnvelope(10, 10, 11, 11);

SELECT lo_export(loid, '/tmp/dummy.jpg')
  FROM tmp_out;

SELECT lo_unlink(loid)
  FROM tmp_out;
```

Beispiel für die Ausgabe von GTIFF

```
SELECT ST_AsGDALRaster(rast, 'GTiff') As rastjpg
FROM dummy_rast WHERE rid=2;

-- Out GeoTiff with jpeg compression, 90% quality
SELECT ST_AsGDALRaster(rast, 'GTiff',
  ARRAY['COMPRESS=JPEG', 'JPEG_QUALITY=90'],
  4269) As rasttiff
FROM dummy_rast WHERE rid=2;
```

Siehe auch

Section 5.3, [ST_GDALDrivers](#), [ST_SRID](#)

9.10.3 ST_AsJPEG

ST_AsJPEG — Return the raster tile selected bands as a single Joint Photographic Exports Group (JPEG) image (byte array). If no band is specified and 1 or more than 3 bands, then only the first band is used. If only 3 bands then all 3 bands are used and mapped to RGB.

Synopsis

```
bytea ST_AsJPEG(raster rast, text[] options=NULL);
bytea ST_AsJPEG(raster rast, integer nband, integer quality);
bytea ST_AsJPEG(raster rast, integer nband, text[] options=NULL);
bytea ST_AsJPEG(raster rast, integer[] nbands, text[] options=NULL);
bytea ST_AsJPEG(raster rast, integer[] nbands, integer quality);
```

Beschreibung

Returns the selected bands of the raster as a single Joint Photographic Exports Group Image (JPEG). Use [ST_AsGDALRaster](#) if you need to export as less common raster types. If no band is specified and 1 or more than 3 bands, then only the first band is used. If 3 bands then all 3 bands are used. There are many variants of the function with many options. These are itemized below:

- `nband` für den Export einzelner Bänder.
- `nbands` is an array of bands to export (note that max is 3 for JPEG) and the order of the bands is RGB. e.g `ARRAY[3,2,1]` means map band 3 to Red, band 2 to green and band 1 to blue
- `quality` number from 0 to 100. The higher the number the crisper the image.
- `options` text Array of GDAL options as defined for JPEG (look at `create_options` for JPEG [ST_GDALDrivers](#)). For JPEG valid ones are `PROGRESSIVE` ON or OFF and `QUALITY` a range from 0 to 100 and default to 75. Refer to [GDAL Raster format options](#) for more details.

Verfügbarkeit: 2.0.0 - GDAL >= 1.6.0.

Beispiele: Ausgabe

```
-- output first 3 bands 75% quality
SELECT ST_AsJPEG(rast) As rastjpg
      FROM dummy_rast WHERE rid=2;

-- output only first band as 90% quality
SELECT ST_AsJPEG(rast,1,90) As rastjpg
      FROM dummy_rast WHERE rid=2;

-- output first 3 bands (but make band 2 Red, band 1 green, and band 3 blue, progressive ↵
  and 90% quality
SELECT ST_AsJPEG(rast,ARRAY[2,1,3],ARRAY['QUALITY=90','PROGRESSIVE=ON']) As rastjpg
      FROM dummy_rast WHERE rid=2;
```

Siehe auch

Section 5.3, [ST_GDALDrivers](#), [ST_AsGDALRaster](#), [ST_AsPNG](#), [ST_AsTIFF](#)

9.10.4 ST_AsPNG

ST_AsPNG — Return the raster tile selected bands as a single portable network graphics (PNG) image (byte array). If 1, 3, or 4 bands in raster and no bands are specified, then all bands are used. If more 2 or more than 4 bands and no bands specified, then only band 1 is used. Bands are mapped to RGB or RGBA space.

Synopsis

```
bytea ST_AsPNG(raster rast, text[] options=NULL);
bytea ST_AsPNG(raster rast, integer nband, integer compression);
bytea ST_AsPNG(raster rast, integer nband, text[] options=NULL);
bytea ST_AsPNG(raster rast, integer[] nbands, integer compression);
bytea ST_AsPNG(raster rast, integer[] nbands, text[] options=NULL);
```

Beschreibung

Returns the selected bands of the raster as a single Portable Network Graphics Image (PNG). Use [ST_AsGDALRaster](#) if you need to export as less common raster types. If no band is specified, then the first 3 bands are exported. There are many variants of the function with many options. If no `srid` is specified then the `srid` of the raster is used. These are itemized below:

- `nband` für den Export einzelner Bänder.
- `nbands` is an array of bands to export (note that max is 4 for PNG) and the order of the bands is RGBA. e.g `ARRAY[3,2,1]` means map band 3 to Red, band 2 to green and band 1 to blue
- `compression` number from 1 to 9. The higher the number the greater the compression.
- `options` text Array of GDAL options as defined for PNG (look at `create_options` for PNG of [ST_GDALDrivers](#)). For PNG valid one is only `ZLEVEL` (amount of time to spend on compression -- default 6) e.g. `ARRAY['ZLEVEL=9']`. `WORLDFILE` is not allowed since the function would have to output two outputs. Refer to [GDAL Raster format options](#) for more details.

Verfügbarkeit: 2.0.0 - GDAL >= 1.6.0.

Beispiele

```
SELECT ST_AsPNG(rast) As rastpng
FROM dummy_rast WHERE rid=2;

-- die ersten 3 Bänder exportieren und Band 3 auf Rot, Band 1 auf Grün und Band 2 auf Blau ↔
  abbilden
SELECT ST_AsPNG(rast, ARRAY[3,1,2]) As rastpng
FROM dummy_rast WHERE rid=2;
```

Siehe auch

[ST_AsGDALRaster](#), [ST_ColorMap](#), [ST_GDALDrivers](#), [Section 5.3](#)

9.10.5 ST_AsTIFF

ST_AsTIFF — Return the raster selected bands as a single TIFF image (byte array). If no band is specified, then will try to use all bands.

Synopsis

```
bytea ST_AsTIFF(raster rast, text[] options="", integer srid=sameassource);
bytea ST_AsTIFF(raster rast, text compression="", integer srid=sameassource);
bytea ST_AsTIFF(raster rast, integer[] nbands, text compression="", integer srid=sameassource);
bytea ST_AsTIFF(raster rast, integer[] nbands, text[] options, integer srid=sameassource);
```

Beschreibung

Returns the selected bands of the raster as a single Tagged Image File Format (TIFF). If no band is specified, will try to use all bands. This is a wrapper around [ST_AsGDALRaster](#). Use [ST_AsGDALRaster](#) if you need to export as less common raster types. There are many variants of the function with many options. If no spatial reference SRS text is present, the spatial reference of the raster is used. These are itemized below:

- `nbands` is an array of bands to export (note that max is 3 for PNG) and the order of the bands is RGB. e.g `ARRAY[3,2,1]` means map band 3 to Red, band 2 to green and band 1 to blue
- `compression` Compression expression -- JPEG90 (or some other percent), LZW, JPEG, DEFLATE9.
- `options` text Array of GDAL create options as defined for GTiff (look at `create_options` for GTiff of [ST_GDALDrivers](#)). or refer to [GDAL Raster format options](#) for more details.
- `srid` srid of `spatial_ref_sys` of the raster. This is used to populate the georeference information

Verfügbarkeit: 2.0.0 - GDAL >= 1.6.0.

Beispiele: 90%ige JPEG Komprimierung

```
SELECT ST_AsTIFF(rast, 'JPEG90') As rasttiff
FROM dummy_rast WHERE rid=2;
```

Siehe auch

[ST_GDALDrivers](#), [ST_AsGDALRaster](#), [ST_SRID](#)

9.11 Rasterdatenverarbeitung

9.11.1 Map Algebra

9.11.1.1 ST_Clip

ST_Clip — Schneidet den Raster nach der Eingabegeometrie. Wenn die Bandnummer nicht angegeben ist, werden alle Bänder bearbeitet. Wenn `crop` nicht angegeben oder `TRUE` ist, wird der Ausgaberraster abgeschnitten.

Synopsis

```
raster ST_Clip(raster rast, integer[] nband, geometry geom, double precision[] nodataval=NULL, boolean crop=TRUE);
raster ST_Clip(raster rast, integer nband, geometry geom, double precision nodataval, boolean crop=TRUE);
raster ST_Clip(raster rast, integer nband, geometry geom, boolean crop);
raster ST_Clip(raster rast, geometry geom, double precision[] nodataval=NULL, boolean crop=TRUE);
raster ST_Clip(raster rast, geometry geom, double precision nodataval, boolean crop=TRUE);
raster ST_Clip(raster rast, geometry geom, boolean crop);
```

Beschreibung

Gibt einen Raster aus, der nach der Eingabegeometrie `geom` ausgeschnitten wird. Wird kein Band angegeben, so werden alle Bänder bearbeitet.

Raster die aus einer Operation mit `ST_Clip` resultieren, müssen in den Bereichen wo sie ausgeschnitten werden, einen NODATA-Wert für jedes Band aufweisen. Wenn keine Werte übergeben werden und für den Ausgangsraster kein NODATA-Wert festgelegt wurde, dann werden die NODATA-Werte des Zielrasters auf `ST_MinPossibleValue(ST_BandPixelType(rast, band))` gesetzt. Wenn die Anzahl der NODATA-Werte in dem übergebenen Feld kleiner als die Anzahl der Bänder ist, wird für die restlichen Bänder der letzte Wert des Feldes verwendet. Wenn die Anzahl der NODATA-Werte größer als die Anzahl der Bänder ist, so werden die zusätzlichen NODATA-Werte übergangen. Alle Varianten, die ein Feld mit NODATA-Werten akzeptieren, nehmen auch einen einzelnen Wert für alle Bänder entgegen.

Wenn `crop` nicht angegeben ist, wird `TRUE` angenommen. Dies bedeutet, dass der Ergebnisraster auf die Ausdehnung der Verschneidung von `geom` und `rast` zugeschnitten wird. Wenn `crop` auf `FALSE` gesetzt ist, hat der neue Raster dieselbe Ausdehnung wie `rast`.

Verfügbarkeit: 2.0.0

Erweiterung: 2.1.0 neu geschrieben in C

Diese Beispiele nutzen Luftbilddaten aus Massachusetts, die von [MassGIS Aerial Orthos](#) heruntergeladen werden können. Die Koordinaten liegen in "Massachusetts State Plane Meters" vor.

Beispiele: 1 Band ausschneiden

```
-- Das erste Band einer Luftbildkachel mit einem 20 Meter Puffer ausschneiden.
SELECT ST_Clip(rast, 1,
               ST_Buffer(ST_Centroid(ST_Envelope(rast)),20)
               ) from aerials.boston
WHERE rid = 4;
```

```
-- Demonstriert den Effekt von "crop" auf die endgültige Größe des Rasters
-- Wenn "crop = true",
-- dann ergibt sich der endgültige Ausschnitt durch schneiden der Geometrie
SELECT ST_XMax(ST_Envelope(ST_Clip(rast, 1, clipper, true))) As xmax_w_trim,
       ST_XMax(clipper) As xmax_clipper,
       ST_XMax(ST_Envelope(ST_Clip(rast, 1, clipper, false))) As xmax_wo_trim,
       ST_XMax(ST_Envelope(rast)) As xmax_rast_orig
FROM (SELECT rast, ST_Buffer(ST_Centroid(ST_Envelope(rast)),6) As clipper
```

```
FROM aerials.boston
WHERE rid = 6) As foo;

xmax_w_trim | xmax_clipper | xmax_wo_trim | xmax_rast_orig
-----+-----+-----+-----
230657.436173996 | 230657.436173996 | 230666.436173996 | 230666.436173996
```



Die gesamte Rasterkachel vor dem Ausschneiden



Nach dem Ausschneiden

Beispiele: 1 Band ohne "crop" ausschneiden und die anderen Bänder ungeändert erneut hinzufügen

```
-- Selbes Beispiel wie vorher, aber mit "crop" auf FALSE gesetzt, damit ST_AddBand ↔
-- verwendet werden kann
-- ST_AddBand setzt voraus, dass all Bänder dieselbe Breite und Höhe haben
SELECT ST_AddBand(ST_Clip(rast, 1,
                          ST_Buffer(ST_Centroid(ST_Envelope(rast)),20),false
                          ), ARRAY[ST_Band(rast,2),ST_Band(rast,3)] ) from aerials.boston
WHERE rid = 6;
```



Die gesamte Rasterkachel vor dem Ausschneiden



Nach dem Abschneiden - unwirklich

Beispiele: Alle Bänder ausschneiden

```
-- Alle Bänder einer Luftbildkachel mit einem 20 Meter Puffer ausschneiden.  
-- Da wir kein bestimmtes Band angeben, werden alle Bänder ausgeschnitten  
SELECT ST_Clip(rast,  
              ST_Buffer(ST_Centroid(ST_Envelope(rast)), 20),  
              false  
              ) from aerials.boston  
WHERE rid = 4;
```



Die gesamte Rasterkachel vor dem Ausschneiden



Nach dem Ausschneiden

Siehe auch

[ST_AddBand](#), [ST_MapAlgebra](#), [ST_Intersection](#)

9.11.1.2 ST_ColorMap

ST_ColorMap — Erzeugt aus einem bestimmten Band des Ausgangsrasters einen neuen Raster mit bis zu vier 8BUI-Bändern (Grauwert, RGB, RGBA). Wenn kein Band angegeben ist, wird Band 1 angenommen.

Synopsis

raster **ST_ColorMap**(raster rast, integer nband=1, text colormap=grayscale, text method=INTERPOLATE);

raster **ST_ColorMap**(raster rast, text colormap, text method=INTERPOLATE);

Beschreibung

Wendet eine `colormap` auf das Band `nband` von `rast` an, wodurch ein neuer Raster aus bis zu vier 8BUI-Bändern erstellt wird. Die Anzahl der 8BUI-Bänder des neuen Raster wird durch die Anzahl der Farbkomponenten bestimmt, die in der `color map` 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 Graustufen in einem 8BUI-Rasterband.
- `pseudocolor` für vier 8BUI-Rasterbänder (RGBA) mit Farbverläufen von Blau zu Grün zu Rot.
- `fire` für vier 8BUI-Rasterbänder (RGBA) mit Farbverläufen von Blau zu Rot zu blassem Gelb.
- `bluered` für vier 8BUI-Rasterbänder (RGBA) mit Farbverläufen von Blau zu blassem Weiß zu Rot.

Anwender können mehrere Einträge (einen pro Zeile) an `colormap` übergeben, um bestimmte Farbschemata zu spezifizieren. Üblicherweise besteht jeder Eintrag aus fünf Werten: der Pixelwert und die zugehörigen Rot-, Grün-, Blau- und Alpha-Komponenten (Farbkomponenten zwischen 0 und 255). Anstelle der Pixelwerte können auch Prozentwerte verwendet werden, wobei 0% und 100% die minimalen und maximalen Werte des Rasterbandes sind. Die Werte können durch Beistriche (','), Tabulatoren, Doppelpunkte (':') und/oder durch Leerzeichen getrennt werden. Für die NODATA-Werte kann der Pixelwert mit `nv`, `NULL` oder auf `NODATA` angegeben werden. Ein Beispiel finden Sie unterhalb.

```
5 0 0 0 255
4 100:50 55 255
1 150,100 150 255
0% 255 255 255 255
nv 0 0 0 0
```

Die Syntax von `colormap` ist ähnlich wie bei dem Modus "color-relief" bei `gdaldem` von GDAL.

Gültige Schlüsselwörter für `method`:

- `INTERPOLATE` verwendet eine lineare Interpolation um einen kontinuierlichen Übergang der Farben zwischen den Pixelwerten zu erhalten
- `EXACT` genaue Entsprechung der Pixelwerte mit der "colormap". Pixel, deren Werte keinen Eintrag in "colormap" haben, werden mit "0 0 0 0" (RGBA) eingefärbt
- `NEAREST` verwendet die Einträge der "colormap", deren Wert dem Pixelwert am nächsten kommt



Note

Eine großartige Hilfestellung für "colormaps" bietet [ColorBrewer](#)



Warning

Bei den resultierenden Bänder des neuen Rasters sind keine NODATA-Werte gesetzt. Verwenden Sie bitte `ST_SetBandNoDataValue` um einen NODATA-Wert zu setzen, falls dieser benötigt wird.

Verfügbarkeit: 2.1.0

Beispiele

Eine "Junk"-Tabelle zum Herumspielen

```
-- Erstellung einer Raster-Testtabelle --
DROP TABLE IF EXISTS funky_shapes;
CREATE TABLE funky_shapes(rast raster);

INSERT INTO funky_shapes(rast)
WITH ref AS (
    SELECT ST_MakeEmptyRaster( 200, 200, 0, 200, 1, -1, 0, 0) AS rast
```

```

)
SELECT
    ST_Union(rast)
FROM (
    SELECT
        ST_AsRaster(
            ST_Rotate(
                ST_Buffer(
                    ST_GeomFromText('LINESTRING(0 2,50 50,150 150,125 50)'),
                    i*2
                ),
                pi() * i * 0.125, ST_Point(50,50)
            ),
            ref.rast, '8BUI'::text, i * 5
        ) AS rast
    FROM ref
    CROSS JOIN generate_series(1, 10, 3) AS i
) AS shapes;

```

```

SELECT
    ST_NumBands(rast) As n_orig,
    ST_NumBands(ST_ColorMap(rast,1, 'greyscale')) As ngrey,
    ST_NumBands(ST_ColorMap(rast,1, 'pseudocolor')) As npseudo,
    ST_NumBands(ST_ColorMap(rast,1, 'fire')) As nfire,
    ST_NumBands(ST_ColorMap(rast,1, 'bluered')) As nbluered,
    ST_NumBands(ST_ColorMap(rast,1, '
100% 255 0 0
80% 160 0 0
50% 130 0 0
30% 30 0 0
20% 60 0 0
0% 0 0 0
nv 255 255 255
')) As nred
FROM funky_shapes;

```

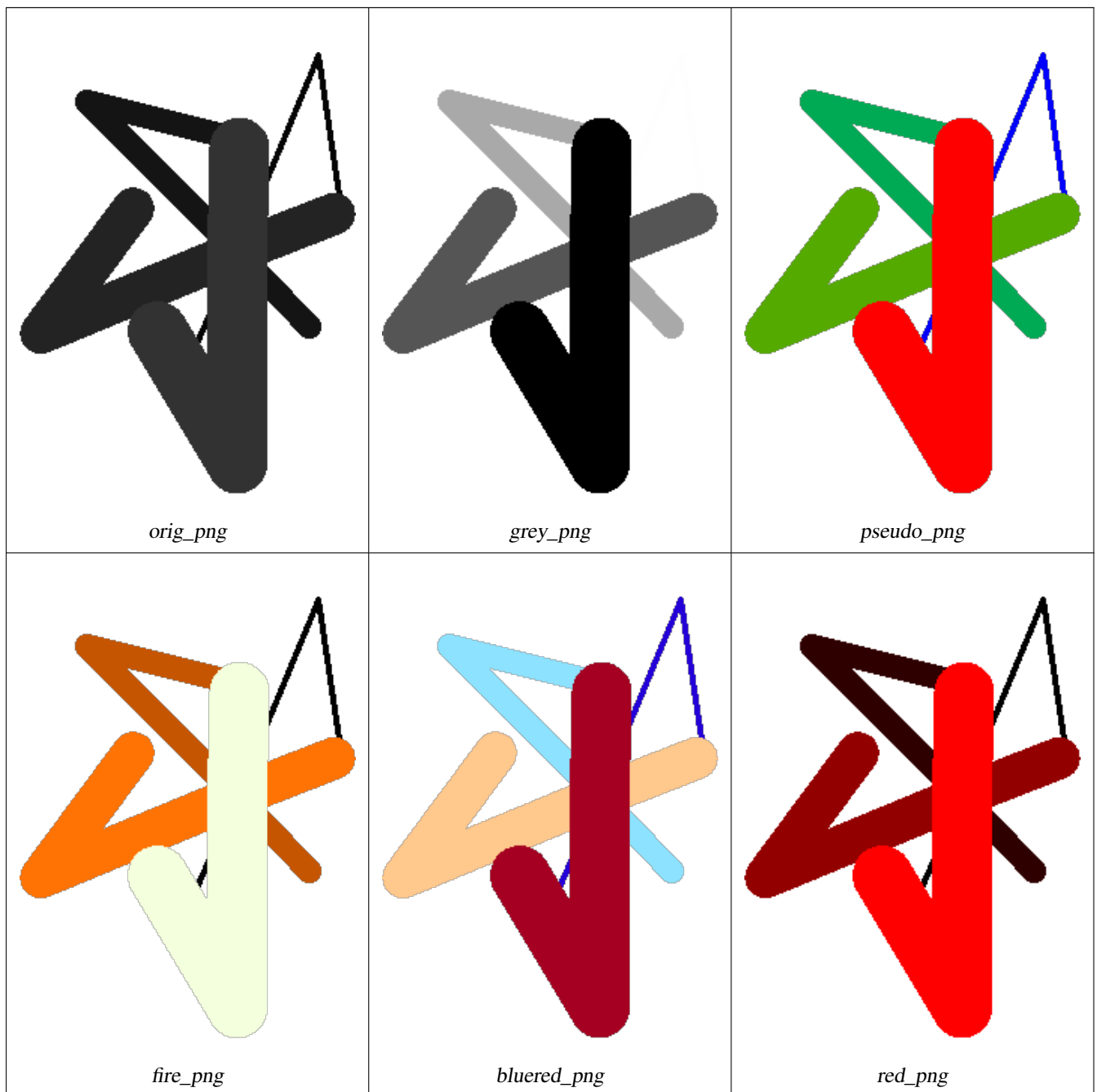
n_orig	ngrey	npseudo	nfire	nbluered	nred
1	1	4	4	4	3

Beispiele: Vergleich verschiedener Farbtafeln mit ST_AsPNG

```

SELECT
    ST_AsPNG(rast) As orig_png,
    ST_AsPNG(ST_ColorMap(rast,1, 'greyscale')) As grey_png,
    ST_AsPNG(ST_ColorMap(rast,1, 'pseudocolor')) As pseudo_png,
    ST_AsPNG(ST_ColorMap(rast,1, 'nfire')) As fire_png,
    ST_AsPNG(ST_ColorMap(rast,1, 'bluered')) As bluered_png,
    ST_AsPNG(ST_ColorMap(rast,1, '
100% 255 0 0
80% 160 0 0
50% 130 0 0
30% 30 0 0
20% 60 0 0
0% 0 0 0
nv 255 255 255
')) As red_png
FROM funky_shapes;

```

**Siehe auch**

[ST_AsPNG](#), [ST_AsRaster](#), [ST_MapAlgebra](#), [ST_NumBands](#), [ST_Reclass](#), [ST_SetBandNoDataValue](#), [ST_Union](#)

9.11.1.3 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 geom);
raster ST_Intersection(raster rast1, raster rast2, double precision[] nodataval);
raster ST_Intersection(raster rast1, raster rast2, text returnband, double precision[] nodataval);
raster ST_Intersection(raster rast1, integer band1, raster rast2, integer band2, double precision[] nodataval);
raster ST_Intersection(raster rast1, integer band1, raster rast2, integer band2, text returnband, double precision[] nodataval);
```

Beschreibung

Gibt Geometry-PixelValue Paare, oder einen Raster aus, der durch die Schnittmenge der beiden Raster bestimmt wird, oder durch die geometrische Verschneidung einer Vektorisierung des Rasters mit einem geometrischen Datentyp.

Die ersten drei Varianten, die ein "setof geomval" zurückgeben, führen die Berechnungen im Vektorraum aus. Der Raster wird zuerst in eine Menge von "geomval"-Zeilen vektorisiert (mit `ST_DumpAsPolygon`) und anschließend mit der Geometrie über die PostGIS Funktion `St_Intersection(geometry, geometry)` verschnitten. Wenn die verschnittene Geometrie nur aus NO-DATA Werten besteht, wird eine leere Geometrie zurückgegeben. Diese wird üblicherweise durch die richtige Verwendung von `ST_Intersect` 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 Rasteraum 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 ausgedrückt, jedes Pixel, das ein Pixel mit NODATA-Wert schneidet wird im Ergebnis selbst zu einem Pixel mit NODATA-Wert.

Rasters resulting from `ST_Intersection` must have a nodata value assigned for areas not intersecting. You can define or replace the nodata value for any resulting band by providing a `nodataval[]` array of one or two nodata values depending if you request 'BAND1', 'BAND2' or 'BOTH' bands. The first value in the array replace the nodata value in the first band and the second value replace the nodata value in the second band. If one input band do not have a nodata value defined and none are provided as an array, one is chosen using the `ST_MinPossibleValue` function. All variant accepting an array of nodata value can also accept a single value which will be assigned to each requested band.

Bei sämtlichen Varianten wird Band 1 angenommen, wenn keine Bandnummer angegeben ist. Wenn Sie die Verschneidung zwischen einem Raster und einer Geometrie als Raster ausgegeben haben wollen, sehen Sie bitte [ST_Clip](#).



Note

Um über die resultierende Ausdehnung, oder über das was für einen NODATA-Wert zurückgeben werden soll, eine bessere Kontrolle zu haben, können Sie die Variante von [ST_MapAlgebraExpr](#) mit den zwei Raster verwenden.



Note

To compute the intersection of a raster band with a geometry in raster space, use [ST_Clip](#). `ST_Clip` works on multiple bands rasters and does not return a band corresponding to the rasterized geometry.



Note

`ST_Intersection` sollte in Verbindung mit `ST_Intersects` und einem Index auf die Rasterspalte und/oder auf die Geometriespalte angewendet werden.

Enhanced: 2.0.0 - Verschneidungsoperation im Rasterraum eingeführt. In Vorgängerversionen von 2.0.0 wurde lediglich die Verschneidung im Vektorraum unterstützt.

Beispiele: Geometrie, Raster -- das Ergebnis sind "geomvals"

```
SELECT
    foo.rid,
    foo.gid,
    ST_AsText((foo.geomval).geom) As geomwkt,
    (foo.geomval).val
FROM (
    SELECT
        A.rid,
        g.gid,
        ST_Intersection(A.rast, g.geom) As geomval
    FROM dummy_rast AS A
    CROSS JOIN (
        VALUES
            (1, ST_Point(3427928, 5793243.85) ),
            (2, ST_GeomFromText('LINESTRING(3427927.85 5793243.75,3427927.8  ←
            5793243.75,3427927.8 5793243.8)')),
            (3, ST_GeomFromText('LINESTRING(1 2, 3 4)'))
        ) As g(gid,geom)
    WHERE A.rid = 2
) As foo;
```

rid	gid	geomwkt	↔	val
2	1	POINT(3427928 5793243.85)	↔	249
2	1	POINT(3427928 5793243.85)	↔	253
2	2	POINT(3427927.85 5793243.75)	↔	254
2	2	POINT(3427927.8 5793243.8)	↔	251
2	2	POINT(3427927.8 5793243.8)	↔	253
2	2	LINESTRING(3427927.8 5793243.75,3427927.8 5793243.8)		252
2	2	MULTILINESTRING((3427927.8 5793243.8,3427927.8 5793243.75),...)		250
2	3	GEOMETRYCOLLECTION EMPTY		

Siehe auch

[geomval](#), [ST_Intersects](#), [ST_MapAlgebraExpr](#), [ST_Clip](#), [ST_AsText](#)

9.11.1.4 ST_MapAlgebra

ST_MapAlgebra — 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=INTERSECTION, raster customextent=NULL, integer distancex=0, integer distancey=0, text[] VARIADIC userargs=NULL);

```

raster ST_MapAlgebra(raster rast, integer[] nband, regprocedure callbackfunc, text pixeltype=NULL, text extenttype=FIRST,
raster customextent=NULL, integer distancex=0, integer distancey=0, text[] VARIADIC userargs=NULL);
raster ST_MapAlgebra(raster rast, integer nband, regprocedure callbackfunc, text pixeltype=NULL, text extenttype=FIRST,
raster customextent=NULL, integer distancex=0, integer distancey=0, text[] VARIADIC userargs=NULL);
raster ST_MapAlgebra(raster rast1, integer nband1, raster rast2, integer nband2, regprocedure callbackfunc, text pixeltype=NULL,
text extenttype=INTERSECTION, raster customextent=NULL, integer distancex=0, integer distancey=0, text[] VARIADIC user-
args=NULL);
raster ST_MapAlgebra(nband integer, regprocedure callbackfunc, float8[] mask, boolean weighted, text pixeltype=NULL, text
extenttype=INTERSECTION, raster customextent=NULL, text[] VARIADIC userargs=NULL);

```

Beschreibung

Gibt für einen oder mehrere Eingaberaster einen Raster mit einem Band, den Bandindizes und einer vom Anwender vorgegebenen Rückruffunktion zurück.

rast,rast1,rast2, rastbandargset Raster die mit der Map Algebra Operation ausgewertet werden.

rastbandargset ermöglicht die Anwendung einer Map Algebra Operation auf viele Raster und/oder viele Bänder. Siehe das Beispiel zu Variante 1.

nband, nband1, nband2 Die Nummern der Rasterbänder, die ausgewertet werden sollen. Die Bänder können über "nband" als Integer oder Integer[] angegeben werden. Bei der Variante mit 2 Raster steht "nband1" für die Bänder von "rast1" und nband2 für die Bänder von rast2.

callbackfunc The `callbackfunc` parameter must be the name and signature of an SQL or PL/pgSQL function, cast to a regprocedure. An example PL/pgSQL function example is:

```

CREATE OR REPLACE FUNCTION sample_callbackfunc(value double precision[][][], position integer[][], VARIADIC userargs text[])
    RETURNS double precision
    AS $$
    BEGIN
        RETURN 0;
    END;
    $$ LANGUAGE 'plpgsql' IMMUTABLE;

```

The `callbackfunc` must have three arguments: a 3-dimension double precision array, a 2-dimension integer array and a variadic 1-dimension text array. The first argument `value` is the set of values (as double precision) from all input rasters. The three dimensions (where indexes are 1-based) are: raster #, row y, column x. The second argument `position` is the set of pixel positions from the output raster and input rasters. The outer dimension (where indexes are 0-based) is the raster #. The position at outer dimension index 0 is the output raster's pixel position. For each outer dimension, there are two elements in the inner dimension for X and Y. The third argument `userargs` is for passing through any user-specified arguments.

Passing a regprocedure argument to a SQL function requires the full function signature to be passed, then cast to a regprocedure type. To pass the above example PL/pgSQL function as an argument, the SQL for the argument is:

```
'sample_callbackfunc(double precision[], integer[], text[])::regprocedure
```

Note that the argument contains the name of the function, the types of the function arguments, quotes around the name and argument types, and a cast to a regprocedure.

mask An n-dimensional array (matrix) of numbers used to filter what cells get passed to map algebra call-back function. 0 means a neighbor cell value should be treated as no-data and 1 means value should be treated as data. If weight is set to true, then the values, are used as multipliers to multiple the pixel value of that value in the neighborhood position.

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 If `pixeltype` is passed in, the one band of the new raster will be of that `pixeltype`. If `pixeltype` is passed NULL or left out, the new raster band will have the same `pixeltype` as the specified band of the first raster (for extent types: INTERSECTION, UNION, FIRST, CUSTOM) or the specified band of the appropriate raster (for extent types: SECOND, LAST). If in doubt, always specify `pixeltype`.

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 `extenttype` CUSTOM ist, dann muss ein Raster für `customextent` übergeben werden. Siehe Beispiel 4 von Variante 1.

distancex The distance in pixels from the reference cell in x direction. So width of resulting matrix would be $2 * \text{distancex} + 1$. If not specified only the reference cell is considered (neighborhood of 0).

distancey Die Entfernung der Pixel zur Referenzzelle in Y-Richtung. Die Höhe der resultierenden Matrix ergibt sich aus $2 * \text{distancey} + 1$. Wenn nicht angegeben, dann wird nur die Referenzzelle berücksichtigt (keine Nachbarschaft/"neighborhood of 0").

userargs Der dritte Übergabewert an die Rückruffunktion `callbackfunc` ist ein Feld mit dem Datentyp `variadic text`. Die an diesen Datentyp angehängten Parameter werden an die `callbackfunc` durchgereicht und sind in dem Übergabewert `userargs` enthalten.



Note

Für nähere Information zu dem Schlüsselwort VARIADIC, sehen Sie bitte die PostgreSQL Dokumentation und den Abschnitt "SQL Functions with Variable Numbers of Arguments" unter [Query Language \(SQL\) Functions](#).



Note

Der Übergabewert `text[]` an die Rückruffunktion `callbackfunc` ist auch dann erforderlich, wenn Sie sonst keine Argumente für die Auswertung übergeben.

Variante 1 nimmt ein Feld an `rastbandarg` entgegen, wodurch die Anwendung einer Map Algebra Operation auf mehrere Raster und/oder mehrere Bänder ermöglicht wird. Siehe das Beispiel zu Variante 1.

Die Varianten 2 und 3 führen die Operation auf einem oder mehreren Bändern eines Raster aus. Siehe die Beispiele mit Variante 2 und 3.

Die Variante 4 führt die Operationen an zwei Raster mit einem Band pro Raster aus. Siehe das Beispiel mit Variante 4.

Verfügbarkeit: 2.2.0: Möglichkeit eine Maske hinzuzufügen

Verfügbarkeit: 2.1.0

Beispiele: Variante 1

Ein Raster, ein Band

```
WITH foo AS (
    SELECT 1 AS rid, ST_AddBand(ST_MakeEmptyRaster(2, 2, 0, 0, 1, -1, 0, 0, 0), 1, '16 ←
        BUI', 1, 0) AS rast
)
SELECT
    ST_MapAlgebra(
        ARRAY[ROW(rast, 1)]::rastbandarg[],
        'sample_callbackfunc(double precision[], int[], text[])::regprocedure
    ) AS rast
FROM foo
```

Ein Raster, mehrere Bänder

```
WITH foo AS (
    SELECT 1 AS rid, ST_AddBand(ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(2, 2, 0, 0, 1, ←
        -1, 0, 0, 0), 1, '16BUI', 1, 0), 2, '8BUI', 10, 0), 3, '32BUI', 100, 0) AS rast
)
SELECT
    ST_MapAlgebra(
        ARRAY[ROW(rast, 3), ROW(rast, 1), ROW(rast, 3), ROW(rast, 2)]::rastbandarg ←
        [],
        'sample_callbackfunc(double precision[], int[], text[])::regprocedure
    ) AS rast
FROM foo
```

Mehrere Raster, mehrere Bänder

```
WITH foo AS (
    SELECT 1 AS rid, ST_AddBand(ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(2, 2, 0, 0, 1, ←
        -1, 0, 0, 0), 1, '16BUI', 1, 0), 2, '8BUI', 10, 0), 3, '32BUI', 100, 0) AS rast ←
    UNION ALL
    SELECT 2 AS rid, ST_AddBand(ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(2, 2, 0, 1, 1, ←
        -1, 0, 0, 0), 1, '16BUI', 2, 0), 2, '8BUI', 20, 0), 3, '32BUI', 300, 0) AS rast
)
SELECT
    ST_MapAlgebra(
        ARRAY[ROW(t1.rast, 3), ROW(t2.rast, 1), ROW(t2.rast, 3), ROW(t1.rast, 2)]:: ←
        rastbandarg[],
        'sample_callbackfunc(double precision[], int[], text[])::regprocedure
    ) AS rast
FROM foo t1
CROSS JOIN foo t2
WHERE t1.rid = 1
    AND t2.rid = 2
```

Ein vollständiges Beispiel mit Coveragekacheln und Nachbarschaft. Diese Abfrage funktioniert nur mit PostgreSQL 9.1 oder höher.

```
WITH foo AS (
    SELECT 0 AS rid, ST_AddBand(ST_MakeEmptyRaster(2, 2, 0, 0, 1, -1, 0, 0, 0), 1, '16 ←
        BUI', 1, 0) AS rast UNION ALL
    SELECT 1, ST_AddBand(ST_MakeEmptyRaster(2, 2, 2, 0, 1, -1, 0, 0, 0), 1, '16BUI', 2, ←
        0) AS rast UNION ALL
    SELECT 2, ST_AddBand(ST_MakeEmptyRaster(2, 2, 4, 0, 1, -1, 0, 0, 0), 1, '16BUI', 3, ←
        0) AS rast UNION ALL

    SELECT 3, ST_AddBand(ST_MakeEmptyRaster(2, 2, 0, -2, 1, -1, 0, 0, 0), 1, '16BUI', ←
        10, 0) AS rast UNION ALL
    SELECT 4, ST_AddBand(ST_MakeEmptyRaster(2, 2, 2, -2, 1, -1, 0, 0, 0), 1, '16BUI', ←
        20, 0) AS rast UNION ALL
    SELECT 5, ST_AddBand(ST_MakeEmptyRaster(2, 2, 4, -2, 1, -1, 0, 0, 0), 1, '16BUI', ←
        30, 0) AS rast UNION ALL

    SELECT 6, ST_AddBand(ST_MakeEmptyRaster(2, 2, 0, -4, 1, -1, 0, 0, 0), 1, '16BUI', ←
        100, 0) AS rast UNION ALL
    SELECT 7, ST_AddBand(ST_MakeEmptyRaster(2, 2, 2, -4, 1, -1, 0, 0, 0), 1, '16BUI', ←
        200, 0) AS rast UNION ALL
    SELECT 8, ST_AddBand(ST_MakeEmptyRaster(2, 2, 4, -4, 1, -1, 0, 0, 0), 1, '16BUI', ←
        300, 0) AS rast
)
SELECT
    t1.rid,
    ST_MapAlgebra(
```



```

        ARRAY[ROW(ST_Union(t2.rast), 1)]::rastbandarg[],
        'sample_callbackfunc(double precision[], int[], text[])::regprocedure,
        '32BUI',
        'CUSTOM', t1.rast,
        1, 1
    ) AS rast
FROM foo t1
CROSS JOIN foo t2
WHERE t1.rid = 4
      AND t2.rid BETWEEN 0 AND 8
      AND ST_Intersects(t1.rast, t2.rast)
GROUP BY t1.rid, t1.rast

```

Das selbe Beispiel wie vorher, mit Coveragekacheln und Nachbarschaft, das aber auch mit PostgreSQL 9.0 funktioniert.

```

WITH src AS (
    SELECT 0 AS rid, ST_AddBand(ST_MakeEmptyRaster(2, 2, 0, 0, 1, -1, 0, 0, 0), 1, '16 ←
        BUI', 1, 0) AS rast UNION ALL
    SELECT 1, ST_AddBand(ST_MakeEmptyRaster(2, 2, 2, 0, 1, -1, 0, 0, 0), 1, '16BUI', 2, ←
        0) AS rast UNION ALL
    SELECT 2, ST_AddBand(ST_MakeEmptyRaster(2, 2, 4, 0, 1, -1, 0, 0, 0), 1, '16BUI', 3, ←
        0) AS rast UNION ALL

    SELECT 3, ST_AddBand(ST_MakeEmptyRaster(2, 2, 0, -2, 1, -1, 0, 0, 0), 1, '16BUI', ←
        10, 0) AS rast UNION ALL
    SELECT 4, ST_AddBand(ST_MakeEmptyRaster(2, 2, 2, -2, 1, -1, 0, 0, 0), 1, '16BUI', ←
        20, 0) AS rast UNION ALL
    SELECT 5, ST_AddBand(ST_MakeEmptyRaster(2, 2, 4, -2, 1, -1, 0, 0, 0), 1, '16BUI', ←
        30, 0) AS rast UNION ALL

    SELECT 6, ST_AddBand(ST_MakeEmptyRaster(2, 2, 0, -4, 1, -1, 0, 0, 0), 1, '16BUI', ←
        100, 0) AS rast UNION ALL
    SELECT 7, ST_AddBand(ST_MakeEmptyRaster(2, 2, 2, -4, 1, -1, 0, 0, 0), 1, '16BUI', ←
        200, 0) AS rast UNION ALL
    SELECT 8, ST_AddBand(ST_MakeEmptyRaster(2, 2, 4, -4, 1, -1, 0, 0, 0), 1, '16BUI', ←
        300, 0) AS rast
)
WITH foo AS (
    SELECT
        t1.rid,
        ST_Union(t2.rast) AS rast
    FROM src t1
    JOIN src t2
        ON ST_Intersects(t1.rast, t2.rast)
        AND t2.rid BETWEEN 0 AND 8
    WHERE t1.rid = 4
    GROUP BY t1.rid
), bar AS (
    SELECT
        t1.rid,
        ST_MapAlgebra(
            ARRAY[ROW(t2.rast, 1)]::rastbandarg[],
            'raster_nmapalgebra_test(double precision[], int[], text[]):: ←
                regprocedure,
            '32BUI',
            'CUSTOM', t1.rast,
            1, 1
        ) AS rast
    FROM src t1
    JOIN foo t2
        ON t1.rid = t2.rid
)

```

```
SELECT
    rid,
    (ST_Metadatas(rast)),
    (ST_BandMetadatas(rast, 1)),
    ST_Value(rast, 1, 1, 1)
FROM bar;
```

Beispiele: Varianten 2 und 3

Ein Raster, mehrere Bänder

```
WITH foo AS (
    SELECT 1 AS rid, ST_AddBand(ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(2, 2, 0, 0, 1, ←
        -1, 0, 0, 0), 1, '16BUI', 1, 0), 2, '8BUI', 10, 0), 3, '32BUI', 100, 0) AS rast
)
SELECT
    ST_MapAlgebra(
        rast, ARRAY[3, 1, 3, 2]::integer[],
        'sample_callbackfunc(double precision[], int[], text[])::regprocedure
    ) AS rast
FROM foo
```

Ein Raster, ein Band

```
WITH foo AS (
    SELECT 1 AS rid, ST_AddBand(ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(2, 2, 0, 0, 1, ←
        -1, 0, 0, 0), 1, '16BUI', 1, 0), 2, '8BUI', 10, 0), 3, '32BUI', 100, 0) AS rast
)
SELECT
    ST_MapAlgebra(
        rast, 2,
        'sample_callbackfunc(double precision[], int[], text[])::regprocedure
    ) AS rast
FROM foo
```

Beispiele: Variante 4

Zwei Raster, zwei Bänder

```
WITH foo AS (
    SELECT 1 AS rid, ST_AddBand(ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(2, 2, 0, 0, 1, ←
        -1, 0, 0, 0), 1, '16BUI', 1, 0), 2, '8BUI', 10, 0), 3, '32BUI', 100, 0) AS rast ←
    UNION ALL
    SELECT 2 AS rid, ST_AddBand(ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(2, 2, 0, 0, 1, ←
        -1, 0, 0, 0), 1, '16BUI', 2, 0), 2, '8BUI', 20, 0), 3, '32BUI', 300, 0) AS rast
)
SELECT
    ST_MapAlgebra(
        t1.rast, 2,
        t2.rast, 1,
        'sample_callbackfunc(double precision[], int[], text[])::regprocedure
    ) AS rast
FROM foo t1
CROSS JOIN foo t2
WHERE t1.rid = 1
    AND t2.rid = 2
```

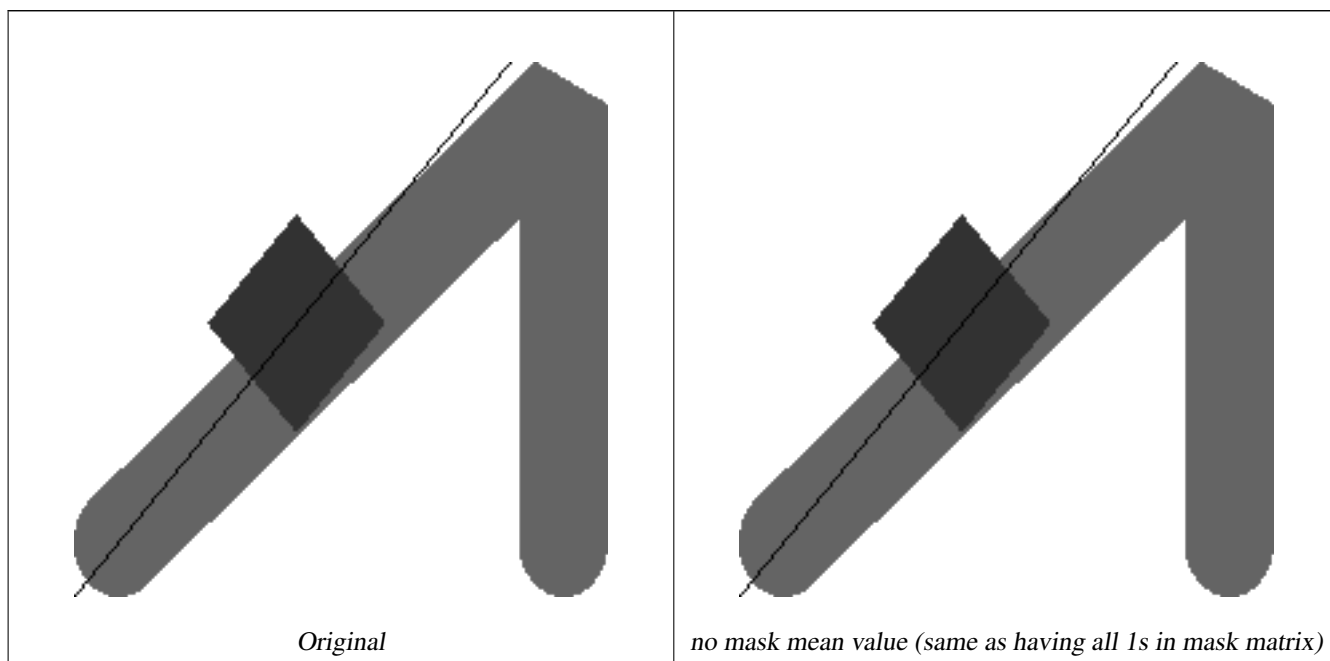
Beispiele: Verwendung von Masken

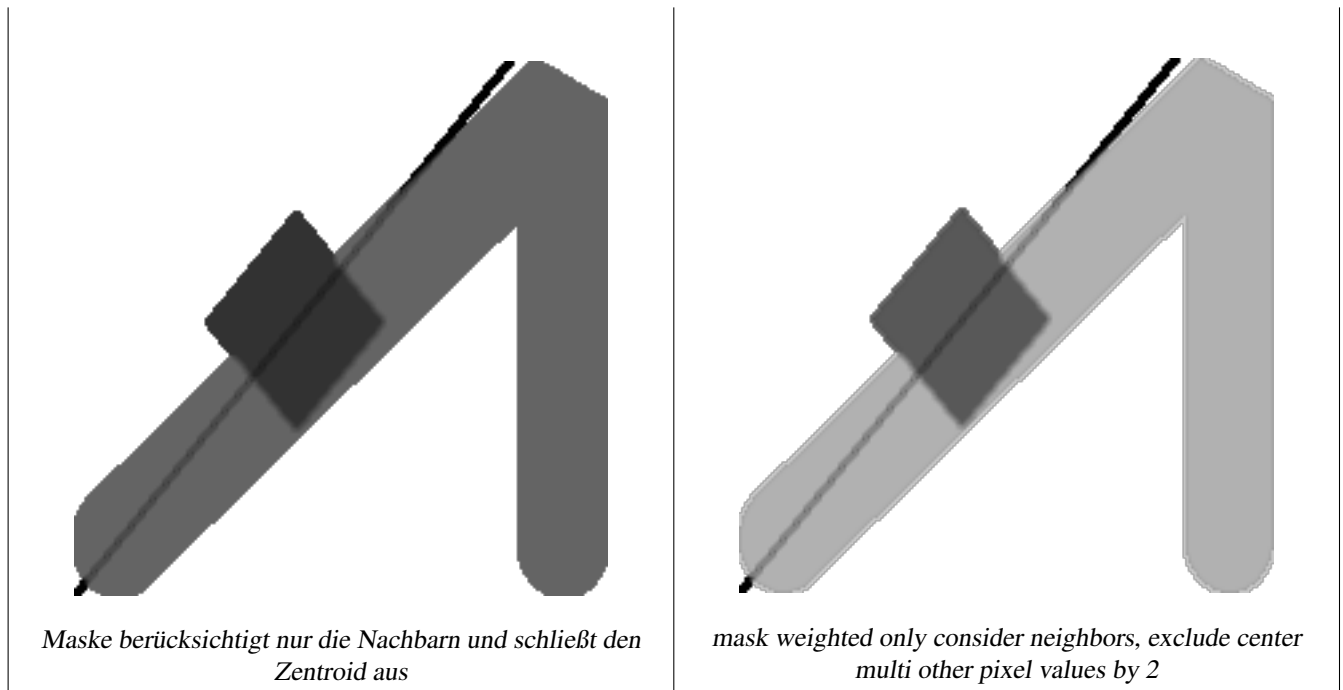
```

WITH foo AS (SELECT
    ST_SetBandNoDataValue(
ST_SetValue(ST_SetValue(ST_AsRaster(
    ST_Buffer(
        ST_GeomFromText('LINESTRING(50 50,100 90,100 50)'), 5,'join=bevel') ←
        ,
        200,200,ARRAY['8BUI'], ARRAY[100], ARRAY[0]), ST_Buffer('POINT(70 ←
        70) '::geometry,10,'quad_segs=1') ,50),
    'LINESTRING(20 20, 100 100, 150 98) '::geometry,1),0) AS rast )
SELECT 'original' AS title, rast
FROM foo
UNION ALL
SELECT 'no mask mean value' AS title, ST_MapAlgebra(rast,1,'ST_mean4ma(double precision[], ←
    int[], text[]) '::regprocedure) AS rast
FROM foo
UNION ALL
SELECT 'mask only consider neighbors, exclude center' AS title, ST_MapAlgebra(rast,1,' ←
    ST_mean4ma(double precision[], int[], text[]) '::regprocedure,
    '{{1,1,1}, {1,0,1}, {1,1,1}}' ::double precision[], false) As rast
FROM foo

UNION ALL
SELECT 'mask weighted only consider neighbors, exclude center multi otehr pixel values by ←
    2' AS title, ST_MapAlgebra(rast,1,'ST_mean4ma(double precision[], int[], text[]) ':: ←
    regprocedure,
    '{{2,2,2}, {2,0,2}, {2,2,2}}' ::double precision[], true) As rast
FROM foo;

```





Siehe auch

[rastbandarg](#), [ST_Union](#), [ST_MapAlgebra](#)

9.11.1.5 ST_MapAlgebra

ST_MapAlgebra — Expression version - Returns a one-band raster given one or two input rasters, band indexes and one or more user-specified SQL expressions.

Synopsis

```
raster ST_MapAlgebra(raster rast, integer nband, text pixeltype, text expression, double precision nodataval=NULL);
raster ST_MapAlgebra(raster rast, text pixeltype, text expression, double precision nodataval=NULL);
raster ST_MapAlgebra(raster rast1, integer nband1, raster rast2, integer nband2, text expression, text pixeltype=NULL, text
extenttype=INTERSECTION, text nodata1expr=NULL, text nodata2expr=NULL, double precision nodatanodataval=NULL);
raster ST_MapAlgebra(raster rast1, raster rast2, text expression, text pixeltype=NULL, text extenttype=INTERSECTION, text
nodata1expr=NULL, text nodata2expr=NULL, double precision nodatanodataval=NULL);
```

Beschreibung

Expression version - Returns a one-band raster given one or two input rasters, band indexes and one or more user-specified SQL expressions.

Verfügbarkeit: 2.1.0

Beschreibung: Variante 1 und 2 (ein Raster)

Creates a new one band raster formed by applying a valid PostgreSQL algebraic operation defined by the `expression` on the input raster (`rast`). If `nband` is not provided, band 1 is assumed. The new raster will have the same georeference, width, and height as the original raster but will only have one band.

If `pixeltype` is passed in, then the new raster will have a band of that pixeltype. If `pixeltype` is passed NULL, then the new raster band will have the same pixeltype as the input `rast` band.

- 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]` - Rasterzeile (von 1 wegzählend) der Pixel von Interesse

Beschreibung: Variante 3 und 4 (zwei Raster)

Creates a new one band raster formed by applying a valid PostgreSQL algebraic operation to the two bands defined by the `expression` on the two input raster bands `rast1`, `(rast2)`. If no `band1`, `band2` is specified band 1 is assumed. The resulting raster will be aligned (scale, skew and pixel corners) on the grid defined by the first raster. The resulting raster will have the extent defined by the `extenttype` parameter.

expression A PostgreSQL algebraic expression involving the two rasters and PostgreSQL defined functions/operators that will define the pixel value when pixels intersect. e.g. `(([rast1] + [rast2])/2.0)::integer`

pixeltype The resulting pixel type of the output raster. Must be one listed in [ST_BandPixelType](#), left out or set to NULL. If not passed in or set to NULL, will default to the pixeltype of the first raster.

extenttype Bestimmt die Ausdehnung des resultierenden Raster

1. `INTERSECTION` - Die Ausdehnung des neuen Raster entspricht der Schnittmenge der beiden Raster. Die Standard-einstellung.
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 An algebraic expression involving only `rast2` or a constant that defines what to return when pixels of `rast1` are nodata values and spatially corresponding `rast2` pixels have values.

nodata2expr An algebraic expression involving only `rast1` or a constant that defines what to return when pixels of `rast2` are nodata values and spatially corresponding `rast1` pixels have values.

nodatanodataval A numeric constant to return when spatially corresponding `rast1` and `rast2` pixels are both nodata values.

- Zugelassene Schlüsselwörter in `expression`, `nodata1expr` und `nodata2expr`

1. `[rast1]` - Zellwert der Pixel von Interesse von `rast1`
2. `[rast1.val]` - Zellwert der Pixel von Interesse von `rast1`
3. `[rast1.x]` - Rasterspalte (von 1 wegzählend) der Pixel von Interesse von `rast1`
4. `[rast1.y]` - Rasterzeile (von 1 wegzählend) der Pixel von Interesse von `rast1`
5. `[rast2]` - Zellwert der Pixel von Interesse von `rast2`
6. `[rast2.val]` - Zellwert der Pixel von Interesse von `rast2`
7. `[rast2.x]` - Rasterspalte (von 1 wegzählend) der Pixel von Interesse von `rast2`
8. `[rast2.y]` - Rasterzeile (von 1 wegzählend) der Pixel von Interesse von `rast2`

Beispiele: Varianten 1 und 2

```
WITH foo AS (
    SELECT ST_AddBand(ST_MakeEmptyRaster(10, 10, 0, 0, 1, 1, 0, 0, 0), '32BF'::text, 1, ↵
        -1) AS rast
)
SELECT
    ST_MapAlgebra(rast, 1, NULL, 'ceil([rast]*[rast.x]/[rast.y]+[rast.val])')
FROM foo;
```

Beispiele: Varianten 3 und 4

```

WITH foo AS (
    SELECT 1 AS rid, ST_AddBand(ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(2, 2, 0, 0, 1, ←
        -1, 0, 0, 0), 1, '16BUI', 1, 0), 2, '8BUI', 10, 0), 3, '32BUI'::text, 100, 0) ←
        AS rast UNION ALL
    SELECT 2 AS rid, ST_AddBand(ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(2, 2, 0, 1, 1, ←
        -1, 0, 0, 0), 1, '16BUI', 2, 0), 2, '8BUI', 20, 0), 3, '32BUI'::text, 300, 0) ←
        AS rast
)
SELECT
    ST_MapAlgebra(
        t1.rast, 2,
        t2.rast, 1,
        '([rast2] + [rast1.val]) / 2'
    ) AS rast
FROM foo t1
CROSS JOIN foo t2
WHERE t1.rid = 1
      AND t2.rid = 2;

```

Siehe auch

[rastbandarg](#), [ST_Union](#), [ST_MapAlgebra](#)

9.11.1.6 ST_MapAlgebraExpr

ST_MapAlgebraExpr — 1 raster band version: Creates a new one band raster formed by applying a valid PostgreSQL algebraic operation on the input raster band and of pixeltype provided. Band 1 is assumed if no band is specified.

Synopsis

raster **ST_MapAlgebraExpr**(raster rast, integer band, text pixeltype, text expression, double precision nodataval=NULL);
 raster **ST_MapAlgebraExpr**(raster rast, text pixeltype, text expression, double precision nodataval=NULL);

Beschreibung**Warning**

ST_MapAlgebraExpr Seit der Version 2.1.0 überholt. Benutzen Sie stattdessen bitte [ST_MapAlgebra](#) .

Creates a new one band raster formed by applying a valid PostgreSQL algebraic operation defined by the *expression* on the input raster (*rast*). If no *band* is specified band 1 is assumed. The new raster will have the same georeference, width, and height as the original raster but will only have one band.

If *pixeltype* is passed in, then the new raster will have a band of that pixeltype. If *pixeltype* is passed NULL, then the new raster band will have the same pixeltype as the input *rast* band.

In the expression you can use the term `[rast]` to refer to the pixel value of the original band, `[rast.x]` to refer to the 1-based pixel column index, `[rast.y]` to refer to the 1-based pixel row index.

Verfügbarkeit: 2.0.0

Beispiele

Create a new 1 band raster from our original that is a function of modulo 2 of the original raster band.

```
ALTER TABLE dummy_rast ADD COLUMN map_rast raster;
UPDATE dummy_rast SET map_rast = ST_MapAlgebraExpr(rast,NULL,'mod([rast]::numeric,2)') ←
WHERE rid = 2;
```

```
SELECT
    ST_Value(rast,1,i,j) As origval,
    ST_Value(map_rast, 1, i, j) As mapval
FROM dummy_rast
CROSS JOIN generate_series(1, 3) AS i
CROSS JOIN generate_series(1,3) AS j
WHERE rid = 2;
```

origval	mapval
253	1
254	0
253	1
253	1
254	0
254	0
250	0
254	0
254	0

Create a new 1 band raster of pixel-type 2BUI from our original that is reclassified and set the nodata value to be 0.

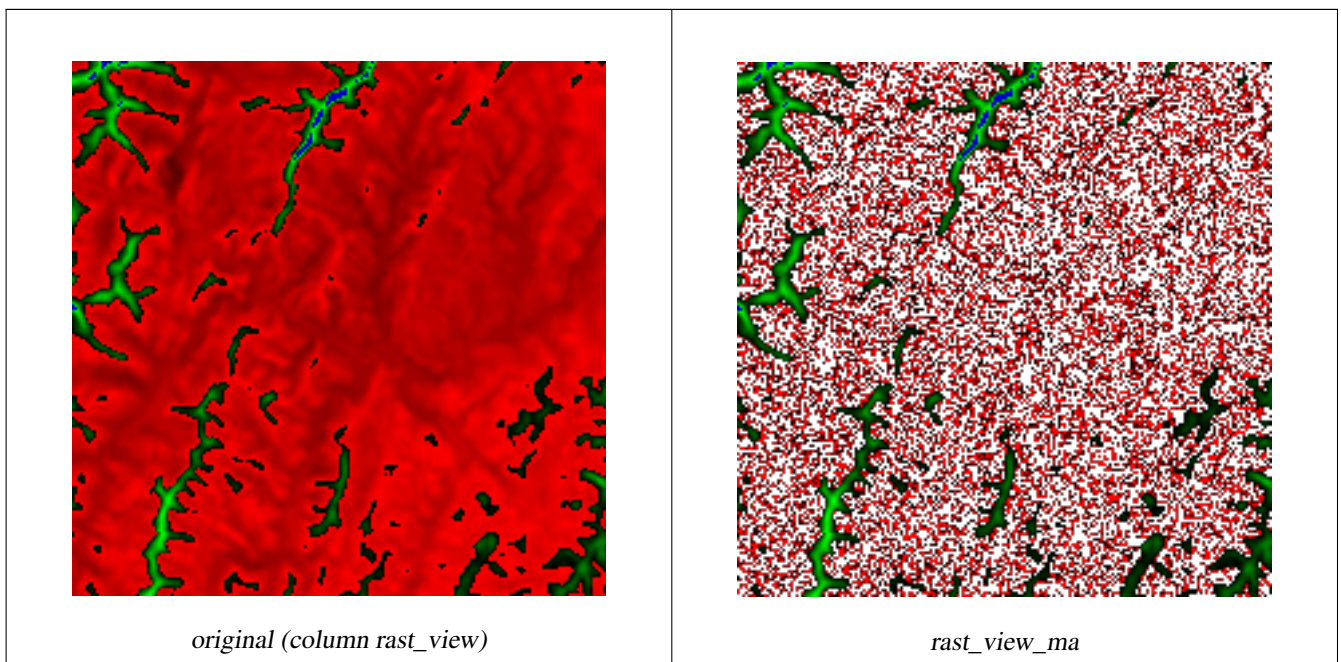
```
ALTER TABLE dummy_rast ADD COLUMN map_rast2 raster;
UPDATE dummy_rast SET
    map_rast2 = ST_MapAlgebraExpr(rast,'2BUI'::text,'CASE WHEN [rast] BETWEEN 100 and ←
    250 THEN 1 WHEN [rast] = 252 THEN 2 WHEN [rast] BETWEEN 253 and 254 THEN 3 ELSE ←
    0 END'::text, '0')
WHERE rid = 2;
```

```
SELECT DISTINCT
    ST_Value(rast,1,i,j) As origval,
    ST_Value(map_rast2, 1, i, j) As mapval
FROM dummy_rast
CROSS JOIN generate_series(1, 5) AS i
CROSS JOIN generate_series(1,5) AS j
WHERE rid = 2;
```

origval	mapval
249	1
250	1
251	
252	2
253	3
254	3

```
SELECT
    ST_BandPixelType(map_rast2) As b1pixtyp
FROM dummy_rast
WHERE rid = 2;
```

b1pixtyp
2BUI



Create a new 3 band raster same pixel type from our original 3 band raster with first band altered by map algebra and remaining 2 bands unaltered.

```
SELECT
    ST_AddBand(
        ST_AddBand(
            ST_AddBand(
                ST_MakeEmptyRaster(rast_view),
                ST_MapAlgebraExpr(rast_view,1,NULL,'tan([rast])*[rast]')
            ),
            ST_Band(rast_view,2)
        ),
        ST_Band(rast_view, 3)
    ) As rast_view_ma
FROM wind
WHERE rid=167;
```

Siehe auch

[ST_MapAlgebraExpr](#), [ST_MapAlgebraFct](#), [ST_BandPixelType](#), [ST_GeoReference](#), [ST_Value](#)

9.11.1.7 ST_MapAlgebraExpr

ST_MapAlgebraExpr — 2 raster band version: Creates a new one band raster formed by applying a valid PostgreSQL algebraic operation on the two input raster bands and of pixeltype provided. band 1 of each raster is assumed if no band numbers are specified. The resulting raster will be aligned (scale, skew and pixel corners) on the grid defined by the first raster and have its extent defined by the "extenttype" parameter. Values for "extenttype" can be: INTERSECTION, UNION, FIRST, SECOND.

Synopsis

raster **ST_MapAlgebraExpr**(raster rast1, raster rast2, text expression, text pixeltype=same_as_rast1_band, text extenttype=INTERSECTION, text nodata1expr=NULL, text nodata2expr=NULL, double precision nodatanodataval=NULL);
raster **ST_MapAlgebraExpr**(raster rast1, integer band1, raster rast2, integer band2, text expression, text pixeltype=same_as_rast1_band, text extenttype=INTERSECTION, text nodata1expr=NULL, text nodata2expr=NULL, double precision nodatanodataval=NULL);

Beschreibung



Warning

ST_MapAlgebraExpr Seit der Version 2.1.0 überholt. Benutzen Sie stattdessen bitte **ST_MapAlgebra** .

Creates a new one band raster formed by applying a valid PostgreSQL algebraic operation to the two bands defined by the expression on the two input raster bands `rast1`, (`rast2`). If no `band1`, `band2` is specified band 1 is assumed. The resulting raster will be aligned (scale, skew and pixel corners) on the grid defined by the first raster. The resulting raster will have the extent defined by the `extenttype` parameter.

expression A PostgreSQL algebraic expression involving the two rasters and PostgreSQL defined functions/operators that will define the pixel value when pixels intersect. e.g. `(([rast1] + [rast2])/2.0)::integer`

pixeltype The resulting pixel type of the output raster. Must be one listed in **ST_BandPixelType**, left out or set to NULL. If not passed in or set to NULL, will default to the pixeltype of the first raster.

extenttype Bestimmt die Ausdehnung des resultierenden Raster

1. **INTERSECTION** - Die Ausdehnung des neuen Raster entspricht der Schnittmenge der beiden Raster. Die Standard-einstellung.
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 An algebraic expression involving only `rast2` or a constant that defines what to return when pixels of `rast1` are nodata values and spatially corresponding `rast2` pixels have values.

nodata2expr An algebraic expression involving only `rast1` or a constant that defines what to return when pixels of `rast2` are nodata values and spatially corresponding `rast1` pixels have values.

nodatanodataval A numeric constant to return when spatially corresponding `rast1` and `rast2` pixels are both nodata values.

If `pixeltype` is passed in, then the new raster will have a band of that pixeltype. If `pixeltype` is passed NULL or no pixel type specified, then the new raster band will have the same pixeltype as the input `rast1` band.

Use the term `[rast1.val]` `[rast2.val]` to refer to the pixel value of the original raster bands and `[rast1.x]`, `[rast1.y]` etc. to refer to the column / row positions of the pixels.

Verfügbarkeit: 2.0.0

Example: 2 Band Intersection and Union

Create a new 1 band raster from our original that is a function of modulo 2 of the original raster band.

```
--Create a cool set of rasters --
DROP TABLE IF EXISTS fun_shapes;
CREATE TABLE fun_shapes(rid serial PRIMARY KEY, fun_name text, rast raster);

-- Insert some cool shapes around Boston in Massachusetts state plane meters --
INSERT INTO fun_shapes(fun_name, rast)
VALUES ('ref', ST_AsRaster(ST_MakeEnvelope(235229, 899970, 237229, 901930,26986),200,200,'8 ←
    BUI',0,0));

INSERT INTO fun_shapes(fun_name,rast)
WITH ref(rast) AS (SELECT rast FROM fun_shapes WHERE fun_name = 'ref' )
SELECT 'area' AS fun_name, ST_AsRaster(ST_Buffer(ST_SetSRID(ST_Point(236229, 900930),26986) ←
    , 1000),
```

```

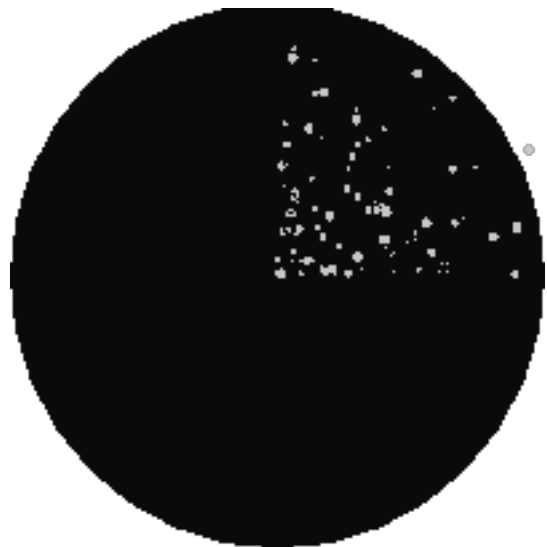
                                ref.rast,'8BUI', 10, 0) As rast
FROM ref
UNION ALL
SELECT 'rand bubbles',
       ST_AsRaster(
         (SELECT ST_Collect(geom)
          FROM (SELECT ST_Buffer(ST_SetSRID(ST_Point(236229 + i*random()*100, 900930 + j*
            random()*100),26986), random()*20) As geom
               FROM generate_series(1,10) As i, generate_series(1,10) As j
               ) As foo ), ref.rast,'8BUI', 200, 0)
FROM ref;

--map them -
SELECT ST_MapAlgebraExpr(
  area.rast, bub.rast, '[rast2.val]', '8BUI', 'INTERSECTION', '[rast2.val]', ←
  '[rast1.val]') As interrast,
  ST_MapAlgebraExpr(
    area.rast, bub.rast, '[rast2.val]', '8BUI', 'UNION', '[rast2.val]', ←
    '[rast1.val]') As unionrast
FROM
  (SELECT rast FROM fun_shapes WHERE
   fun_name = 'area') As area
CROSS JOIN (SELECT rast
FROM fun_shapes WHERE
fun_name = 'rand bubbles') As bub

```



Map Algebra Verschneidung



Map Algebra Vereinigung

Example: Overlaying rasters on a canvas as separate bands

```

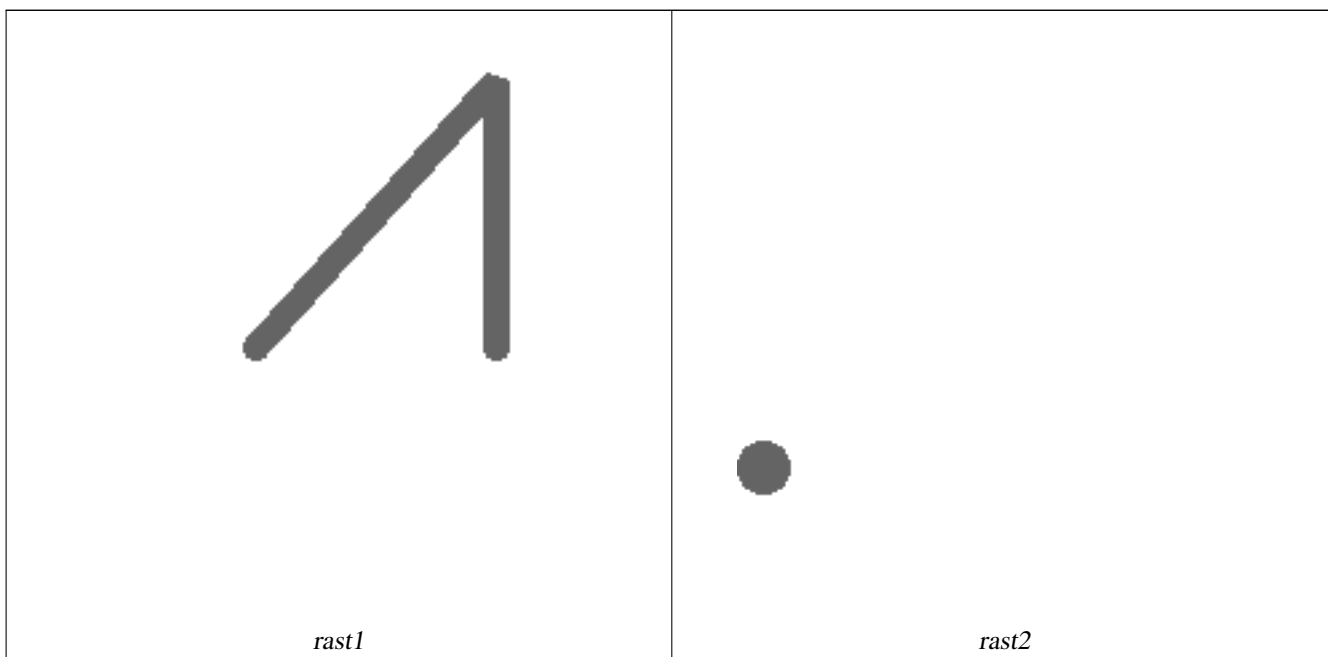
-- we use ST_AsPNG to render the image so all single band ones look grey --
WITH mygeoms
  AS ( SELECT 2 As bnum, ST_Buffer(ST_Point(1,5),10) As geom
       UNION ALL
       SELECT 3 AS bnum,
         ST_Buffer(ST_GeomFromText('LINESTRING(50 50,150 150,150 50)'), 10,'join= ←
         bevel') As geom
       UNION ALL

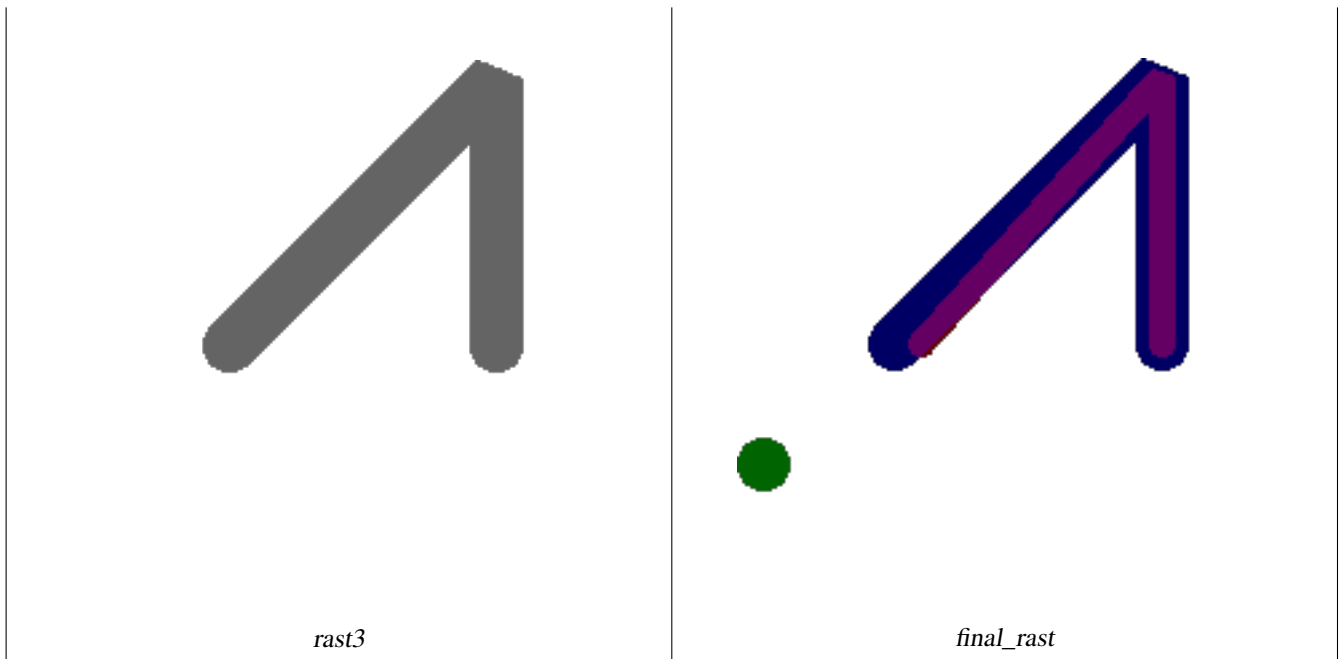
```

```

SELECT 1 As bnum,
       ST_Buffer(ST_GeomFromText('LINESTRING(60 50,150 150,150 50)'), 5,'join= ↵
       bevel') As geom
),
-- define our canvas to be 1 to 1 pixel to geometry
canvas
AS (SELECT ST_AddBand(ST_MakeEmptyRaster(200,
    200,
    ST_XMin(e)::integer, ST_YMax(e)::integer, 1, -1, 0, 0) , '8BUI'::text,0) As rast
FROM (SELECT ST_Extent(geom) As e,
        Max(ST_SRID(geom)) As srid
      from mygeoms
      ) As foo
),
rbands AS (SELECT ARRAY(SELECT ST_MapAlgebraExpr(canvas.rast, ST_AsRaster(m.geom, canvas ↵
.rast, '8BUI', 100),
        '[rast2.val]', '8BUI', 'FIRST', '[rast2.val]', '[rast1.val]') As rast
      FROM mygeoms AS m CROSS JOIN canvas
      ORDER BY m.bnum) As rasts
)
SELECT rasts[1] As rast1 , rasts[2] As rast2, rasts[3] As rast3, ST_AddBand(
        ST_AddBand(rasts[1],rasts[2]), rasts[3]) As final_rast
FROM rbands;

```





Example: Overlay 2 meter boundary of select parcels over an aerial imagery

```
-- Create new 3 band raster composed of first 2 clipped bands, and overlay of 3rd band with ←
our geometry
-- This query took 3.6 seconds on PostGIS windows 64-bit install
WITH pr AS
-- Note the order of operation: we clip all the rasters to dimensions of our region
(SELECT ST_Clip(rast,ST_Expand(geom,50) ) As rast, g.geom
FROM aerals.o_2_boston AS r INNER JOIN
-- union our parcels of interest so they form a single geometry we can later intersect with
(SELECT ST_Union(ST_Transform(the_geom,26986)) AS geom
FROM landparcels WHERE pid IN('0303890000', '0303900000')) As g
ON ST_Intersects(rast::geometry, ST_Expand(g.geom,50))
),
-- we then union the raster shards together
-- ST_Union on raster is kinda of slow but much faster the smaller you can get the rasters
-- therefore we want to clip first and then union
prunion AS
(SELECT ST_AddBand(NULL, ARRAY[ST_Union(rast,1),ST_Union(rast,2),ST_Union(rast,3)] ) As ←
clipped,geom
FROM pr
GROUP BY geom)
-- return our final raster which is the unioned shard with
-- with the overlay of our parcel boundaries
-- add first 2 bands, then mapalgebra of 3rd band + geometry
SELECT ST_AddBand(ST_Band(clipped,ARRAY[1,2])
, ST_MapAlgebraExpr(ST_Band(clipped,3), ST_AsRaster(ST_Buffer(ST_Boundary(geom),2), ←
clipped, '8BUI',250),
'[rast2.val]', '8BUI', 'FIRST', '[rast2.val]', '[rast1.val]') ) As rast
FROM prunion;
```



Die blauen Linien sind die Ränder der ausgewählten Grundstücke.

Siehe auch

[ST_MapAlgebraExpr](#), [ST_AddBand](#), [ST_AsPNG](#), [ST_AsRaster](#), [ST_MapAlgebraFct](#), [ST_BandPixelType](#), [ST_GeoReference](#), [ST_Value](#), [ST_Union](#), [ST_Union](#)

9.11.1.8 ST_MapAlgebraFct

ST_MapAlgebraFct — 1 band version - Creates a new one band raster formed by applying a valid PostgreSQL function on the input raster band and of pixeltype provided. Band 1 is assumed if no band is specified.

Synopsis

```
raster ST_MapAlgebraFct(raster rast, regprocedure onerasteruserfunc);  
raster ST_MapAlgebraFct(raster rast, regprocedure onerasteruserfunc, text[] VARIADIC args);  
raster ST_MapAlgebraFct(raster rast, text pixeltype, regprocedure onerasteruserfunc);  
raster ST_MapAlgebraFct(raster rast, text pixeltype, regprocedure onerasteruserfunc, text[] VARIADIC args);  
raster ST_MapAlgebraFct(raster rast, integer band, regprocedure onerasteruserfunc);  
raster ST_MapAlgebraFct(raster rast, integer band, regprocedure onerasteruserfunc, text[] VARIADIC args);  
raster ST_MapAlgebraFct(raster rast, integer band, text pixeltype, regprocedure onerasteruserfunc);  
raster ST_MapAlgebraFct(raster rast, integer band, text pixeltype, regprocedure onerasteruserfunc, text[] VARIADIC args);
```

Beschreibung



Warning

ST_MapAlgebraFct Seit der Version 2.1.0 überholt. Benutzen Sie stattdessen bitte **ST_MapAlgebra**

Creates a new one band raster formed by applying a valid PostgreSQL function specified by the `onerasteruserfunc` on the input raster (`rast`). If no band is specified, band 1 is assumed. The new raster will have the same georeference, width, and height as the original raster but will only have one band.

If `pixeltype` is passed in, then the new raster will have a band of that pixeltype. If `pixeltype` is passed NULL, then the new raster band will have the same pixeltype as the input `rast` band.

The `onerasteruserfunc` parameter must be the name and signature of a SQL or PL/pgSQL function, cast to a regprocedure. A very simple and quite useless PL/pgSQL function example is:

```
CREATE OR REPLACE FUNCTION simple_function(pixel FLOAT, pos INTEGER[], VARIADIC args TEXT ←
[])
RETURNS FLOAT
AS $$ BEGIN
    RETURN 0.0;
END; $$
LANGUAGE 'plpgsql' IMMUTABLE;
```

The `userfunction` may accept two or three arguments: a float value, an optional integer array, and a variadic text array. The first argument is the value of an individual raster cell (regardless of the raster datatype). The second argument is the position of the current processing cell in the form `'{x,y}'`. The third argument indicates that all remaining parameters to `ST_MapAlgebraFct` shall be passed through to the `userfunction`.

Passing a regprocedure argument to a SQL function requires the full function signature to be passed, then cast to a regprocedure type. To pass the above example PL/pgSQL function as an argument, the SQL for the argument is:

```
'simple_function(float,integer[],text[]) '::regprocedure
```

Note that the argument contains the name of the function, the types of the function arguments, quotes around the name and argument types, and a cast to a regprocedure.

The third argument to the `userfunction` is a variadic text array. All trailing text arguments to any `ST_MapAlgebraFct` call are passed through to the specified `userfunction`, and are contained in the `args` argument.



Note

Für nähere Information zu dem Schlüsselwort VARIADIC, sehen Sie bitte die PostgreSQL Dokumentation und den Abschnitt "SQL Functions with Variable Numbers of Arguments" unter [Query Language \(SQL\) Functions](#).



Note

The `text[]` argument to the `userfunction` is required, regardless of whether you choose to pass any arguments to your user function for processing or not.

Verfügbarkeit: 2.0.0

Beispiele

Create a new 1 band raster from our original that is a function of modulo 2 of the original raster band.

```
ALTER TABLE dummy_rast ADD COLUMN map_rast raster;
CREATE FUNCTION mod_fct(pixel float, pos integer[], variadic args text[])
RETURNS float
AS $$
BEGIN
    RETURN pixel::integer % 2;
END;
$$
```

```
LANGUAGE 'plpgsql' IMMUTABLE;
```

```
UPDATE dummy_rast SET map_rast = ST_MapAlgebraFct(rast,NULL,'mod_fct(float,integer[],text ←  
[])'::regprocedure) WHERE rid = 2;
```

```
SELECT ST_Value(rast,1,i,j) As origval, ST_Value(map_rast, 1, i, j) As mapval  
FROM dummy_rast CROSS JOIN generate_series(1, 3) AS i CROSS JOIN generate_series(1,3) AS j  
WHERE rid = 2;
```

origval	mapval
253	1
254	0
253	1
253	1
254	0
254	0
250	0
254	0
254	0

Create a new 1 band raster of pixel-type 2BUI from our original that is reclassified and set the nodata value to a passed parameter to the user function (0).

```
ALTER TABLE dummy_rast ADD COLUMN map_rast2 raster;
```

```
CREATE FUNCTION classify_fct(pixel float, pos integer[], variadic args text[])  
RETURNS float
```

```
AS
```

```
$$
```

```
DECLARE
```

```
    nodata float := 0;
```

```
BEGIN
```

```
    IF NOT args[1] IS NULL THEN  
        nodata := args[1];
```

```
    END IF;
```

```
    IF pixel < 251 THEN
```

```
        RETURN 1;
```

```
    ELSIF pixel = 252 THEN
```

```
        RETURN 2;
```

```
    ELSIF pixel > 252 THEN
```

```
        RETURN 3;
```

```
    ELSE
```

```
        RETURN nodata;
```

```
    END IF;
```

```
END;
```

```
$$
```

```
LANGUAGE 'plpgsql';
```

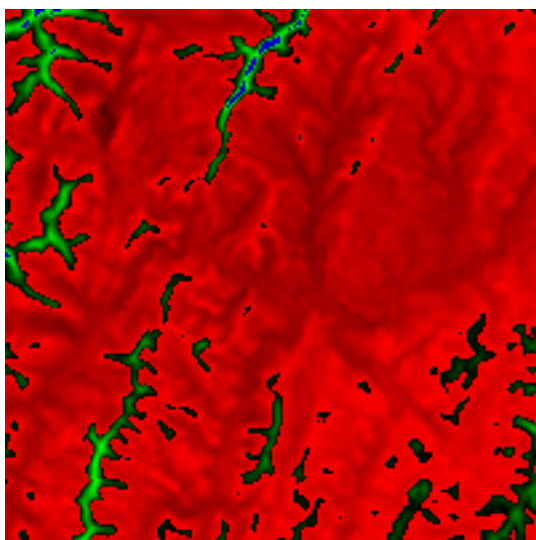
```
UPDATE dummy_rast SET map_rast2 = ST_MapAlgebraFct(rast,'2BUI','classify_fct(float,integer ←  
[],text[])'::regprocedure, '0') WHERE rid = 2;
```

```
SELECT DISTINCT ST_Value(rast,1,i,j) As origval, ST_Value(map_rast2, 1, i, j) As mapval  
FROM dummy_rast CROSS JOIN generate_series(1, 5) AS i CROSS JOIN generate_series(1,5) AS j  
WHERE rid = 2;
```

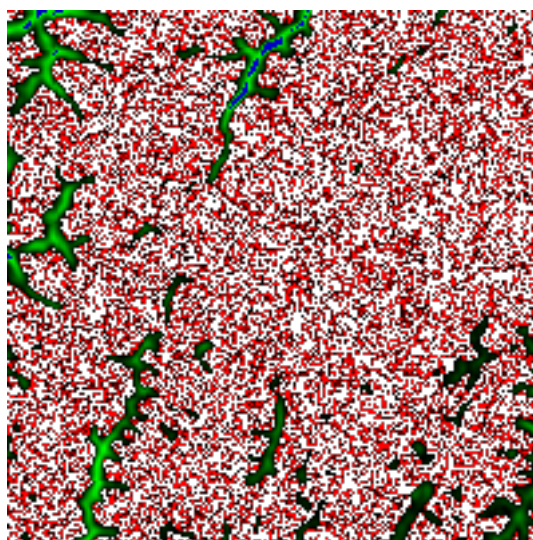
origval	mapval
249	1
250	1
251	
252	2
253	3
254	3

```
SELECT ST_BandPixelType(map_rast2) As b1pixtyp
FROM dummy_rast WHERE rid = 2;
```

```
b1pixtyp
-----
2BUI
```



original (column rast-view)



rast_view_ma

Create a new 3 band raster same pixel type from our original 3 band raster with first band altered by map algebra and remaining 2 bands unaltered.

```
CREATE FUNCTION rast_plus_tan(pixel float, pos integer[], variadic args text[])
RETURNS float
AS
$$
BEGIN
    RETURN tan(pixel) * pixel;
END;
$$
LANGUAGE 'plpgsql';

SELECT ST_AddBand(
    ST_AddBand(
        ST_AddBand(
            ST_MakeEmptyRaster(rast_view),
            ST_MapAlgebraFct(rast_view,1,NULL,'rast_plus_tan(float,integer[], ↵
                text[])'::regprocedure)
        ),
        ST_Band(rast_view,2)
    ),
    ST_Band(rast_view, 3) As rast_view_ma
)
FROM wind
WHERE rid=167;
```


Siehe auch

[ST_MapAlgebraExpr](#), [ST_BandPixelType](#), [ST_GeoReference](#), [ST_SetValue](#)

9.11.1.9 ST_MapAlgebraFct

ST_MapAlgebraFct — 2 band version - Creates a new one band raster formed by applying a valid PostgreSQL function on the 2 input raster bands and of pixeltype provided. Band 1 is assumed if no band is specified. Extent type defaults to INTERSECTION if not specified.

Synopsis

raster **ST_MapAlgebraFct**(raster rast1, raster rast2, regprocedure tworastuserfunc, text pixeltype=same_as_rast1, text extenttype=INTERSECTION, text[] VARIADIC userargs);

raster **ST_MapAlgebraFct**(raster rast1, integer band1, raster rast2, integer band2, regprocedure tworastuserfunc, text pixeltype=same_as_rast1, text extenttype=INTERSECTION, text[] VARIADIC userargs);

Beschreibung**Warning**

ST_MapAlgebraFct Seit der Version 2.1.0 überholt. Benutzen Sie stattdessen bitte [ST_MapAlgebra](#)

Creates a new one band raster formed by applying a valid PostgreSQL function specified by the `tworastuserfunc` on the input raster `rast1`, `rast2`. If no `band1` or `band2` is specified, band 1 is assumed. The new raster will have the same georeference, width, and height as the original rasters but will only have one band.

If `pixeltype` is passed in, then the new raster will have a band of that pixeltype. If `pixeltype` is passed NULL or left out, then the new raster band will have the same pixeltype as the input `rast1` band.

The `tworastuserfunc` parameter must be the name and signature of an SQL or PL/pgSQL function, cast to a regprocedure. An example PL/pgSQL function example is:

```
CREATE OR REPLACE FUNCTION simple_function_for_two_rasters(pixel1 FLOAT, pixel2 FLOAT, pos ←
    INTEGER[], VARIADIC args TEXT[])
    RETURNS FLOAT
    AS $$ BEGIN
        RETURN 0.0;
    END; $$
LANGUAGE 'plpgsql' IMMUTABLE;
```

The `tworastuserfunc` may accept three or four arguments: a double precision value, a double precision value, an optional integer array, and a variadic text array. The first argument is the value of an individual raster cell in `rast1` (regardless of the raster datatype). The second argument is an individual raster cell value in `rast2`. The third argument is the position of the current processing cell in the form `'{x,y}'`. The fourth argument indicates that all remaining parameters to **ST_MapAlgebraFct** shall be passed through to the `tworastuserfunc`.

Passing a regprocedure argument to a SQL function requires the full function signature to be passed, then cast to a regprocedure type. To pass the above example PL/pgSQL function as an argument, the SQL for the argument is:

```
'simple_function(double precision, double precision, integer[], text[])::regprocedure
```

Note that the argument contains the name of the function, the types of the function arguments, quotes around the name and argument types, and a cast to a regprocedure.

The fourth argument to the `tworastuserfunc` is a variadic text array. All trailing text arguments to any **ST_MapAlgebraFct** call are passed through to the specified `tworastuserfunc`, and are contained in the `userargs` argument.

**Note**

Für nähere Information zu dem Schlüsselwort VARIADIC, sehen Sie bitte die PostgreSQL Dokumentation und den Abschnitt "SQL Functions with Variable Numbers of Arguments" unter [Query Language \(SQL\) Functions](#).

**Note**

The `text[]` argument to the `tworastuserfunc` is required, regardless of whether you choose to pass any arguments to your user function for processing or not.

Verfügbarkeit: 2.0.0

Example: Overlaying rasters on a canvas as separate bands

```
-- define our user defined function --
CREATE OR REPLACE FUNCTION raster_mapalgebra_union(
    rast1 double precision,
    rast2 double precision,
    pos integer[],
    VARIADIC userargs text[]
)
    RETURNS double precision
    AS $$
    DECLARE
    BEGIN
        CASE
            WHEN rast1 IS NOT NULL AND rast2 IS NOT NULL THEN
                RETURN ((rast1 + rast2)/2.);
            WHEN rast1 IS NULL AND rast2 IS NULL THEN
                RETURN NULL;
            WHEN rast1 IS NULL THEN
                RETURN rast2;
            ELSE
                RETURN rast1;
        END CASE;

        RETURN NULL;
    END;
    $$ LANGUAGE 'plpgsql' IMMUTABLE COST 1000;

-- prep our test table of rasters
DROP TABLE IF EXISTS map_shapes;
CREATE TABLE map_shapes(rid serial PRIMARY KEY, rast raster, bnum integer, descrip text);
INSERT INTO map_shapes(rast,bnum, descrip)
WITH mygeoms
    AS ( SELECT 2 As bnum, ST_Buffer(ST_Point(90,90),30) As geom, 'circle' As descrip
        UNION ALL
        SELECT 3 AS bnum,
            ST_Buffer(ST_GeomFromText('LINESTRING(50 50,150 150,150 50)'), 15) As geom, ←
            'big road' As descrip
        UNION ALL
        SELECT 1 As bnum,
            ST_Translate(ST_Buffer(ST_GeomFromText('LINESTRING(60 50,150 150,150 50)'), ←
            8,'join=bevel'), 10,-6) As geom, 'small road' As descrip
    ),
-- define our canvas to be 1 to 1 pixel to geometry
canvas
    AS ( SELECT ST_AddBand(ST_MakeEmptyRaster(250,
```

```

250,
ST_XMin(e)::integer, ST_YMax(e)::integer, 1, -1, 0, 0 ) , '8BUI'::text,0) As rast
FROM (SELECT ST_Extent(geom) As e,
        Max(ST_SRID(geom)) As srid
      from mygeoms
      ) As foo
    )
-- return our rasters aligned with our canvas
SELECT ST_AsRaster(m.geom, canvas.rast, '8BUI', 240) As rast, bnum, descrip
      FROM mygeoms AS m CROSS JOIN canvas
UNION ALL
SELECT canvas.rast, 4, 'canvas'
FROM canvas;

-- Map algebra on single band rasters and then collect with ST_AddBand
INSERT INTO map_shapes(rast,bnum,descrip)
SELECT ST_AddBand(ST_AddBand(rasts[1], rasts[2]),rasts[3]), 4, 'map bands overlay fct union ←
      (canvas)'
      FROM (SELECT ARRAY(SELECT ST_MapAlgebraFct(m1.rast, m2.rast,
        'raster_mapalgebra_union(double precision, double precision, ←
        integer[], text[])'::regprocedure, '8BUI', 'FIRST')
        FROM map_shapes As m1 CROSS JOIN map_shapes As m2
        WHERE m1.descrip = 'canvas' AND m2.descrip <> 'canvas' ORDER BY m2.bnum) As rasts) ←
      As foo;

```



map bands overlay (canvas) (R: small road, G: circle, B: big road)

Eine benutzerdefinierte Funktion, die zusätzliche Übergabewerte entgegennimmt

```

CREATE OR REPLACE FUNCTION raster_mapalgebra_userargs(
    rast1 double precision,
    rast2 double precision,
    pos integer[],
    VARIADIC userargs text[]

```

```

)
    RETURNS double precision
    AS $$
    DECLARE
    BEGIN
        CASE
            WHEN rast1 IS NOT NULL AND rast2 IS NOT NULL THEN
                RETURN least(userargs[1]::integer, (rast1 + rast2)/2.);
            WHEN rast1 IS NULL AND rast2 IS NULL THEN
                RETURN userargs[2]::integer;
            WHEN rast1 IS NULL THEN
                RETURN greatest(rast2, random()*userargs[3]::integer)::integer;
            ELSE
                RETURN greatest(rast1, random()*userargs[4]::integer)::integer;
        END CASE;

        RETURN NULL;
    END;
    $$ LANGUAGE 'plpgsql' VOLATILE COST 1000;

SELECT ST_MapAlgebraFct(m1.rast, 1, m1.rast, 3,
    'raster_mapalgebra_userargs(double precision, double precision, integer[], text[])'::regprocedure,
    '8BUI', 'INTERSECT', '100','200','200','0')
    FROM map_shapes As m1
    WHERE m1.descrip = 'map bands overlay fct union (canvas)';

```



Eine benutzerdefinierte Funktion mit zusätzlichen Übergabewerten und unterschiedlichen Bändern des gleichen Raster

Siehe auch

[ST_MapAlgebraExpr](#), [ST_BandPixelType](#), [ST_GeoReference](#), [ST_SetValue](#)

9.11.1.10 ST_MapAlgebraFctNgb

ST_MapAlgebraFctNgb — 1-band version: Map Algebra Nearest Neighbor using user-defined PostgreSQL function. Return a raster which values are the result of a PLPGSQL user function involving a neighborhood of values from the input raster band.

Synopsis

raster **ST_MapAlgebraFctNgb**(raster rast, integer band, text pixeltype, integer ngbwidth, integer ngbheight, regprocedure onerastngbuserfunc, text nodatamode, text[] VARIADIC args);

Beschreibung



Warning

ST_MapAlgebraFctNgb Seit der Version 2.1.0 überholt. Benutzen Sie stattdessen bitte **ST_MapAlgebra**

(one raster version) Return a raster which values are the result of a PLPGSQL user function involving a neighborhood of values from the input raster band. The user function takes the neighborhood of pixel values as an array of numbers, for each pixel, returns the result from the user function, replacing pixel value of currently inspected pixel with the function result.

rast Raster der durch die benutzerdefinierte Funktion ausgewertet werden soll.

band Die Bandnummer des Raster, die ausgewertet werden soll. Die Standardeinstellung ist 1.

pixeltype The resulting pixel type of the output raster. Must be one listed in **ST_BandPixelType** or left out or set to NULL. If not passed in or set to NULL, will default to the pixeltype of the `rast`. Results are truncated if they are larger than what is allowed for the pixeltype.

ngbwidth The width of the neighborhood, in cells.

ngbheight The height of the neighborhood, in cells.

onerastngbuserfunc PLPGSQL/psql user function to apply to neighborhood pixels of a single band of a raster. The first element is a 2-dimensional array of numbers representing the rectangular pixel neighborhood

nodatamode Defines what value to pass to the function for a neighborhood pixel that is nodata or NULL

'ignore': any NODATA values encountered in the neighborhood are ignored by the computation -- this flag must be sent to the user callback function, and the user function decides how to ignore it.

'NULL': any NODATA values encountered in the neighborhood will cause the resulting pixel to be NULL -- the user callback function is skipped in this case.

'value': any NODATA values encountered in the neighborhood are replaced by the reference pixel (the one in the center of the neighborhood). Note that if this value is NODATA, the behavior is the same as 'NULL' (for the affected neighborhood)

args Arguments to pass into the user function.

Verfügbarkeit: 2.0.0

Beispiele

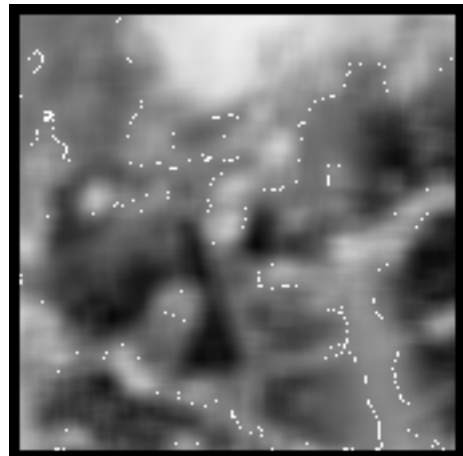
Examples utilize the katrina raster loaded as a single tile described in http://trac.osgeo.org/gdal/wiki/frmts_wtkraster.html and then prepared in the **ST_Rescale** examples

```
--
-- A simple 'callback' user function that averages up all the values in a neighborhood.
--
CREATE OR REPLACE FUNCTION rast_avg(matrix float[][], nodatamode text, variadic args text ←
[])
RETURNS float AS
$$
DECLARE
    _matrix float[][];
    x1 integer;
    x2 integer;
    y1 integer;
    y2 integer;
    sum float;
BEGIN
    _matrix := matrix;
    sum := 0;
    FOR x in array_lower(matrix, 1)..array_upper(matrix, 1) LOOP
        FOR y in array_lower(matrix, 2)..array_upper(matrix, 2) LOOP
            sum := sum + _matrix[x][y];
        END LOOP;
    END LOOP;
    RETURN (sum*1.0/(array_upper(matrix,1)*array_upper(matrix,2) ))::integer ;
END;
$$
LANGUAGE 'plpgsql' IMMUTABLE COST 1000;

-- now we apply to our raster averaging pixels within 2 pixels of each other in X and Y ←
direction --
SELECT ST_MapAlgebraFctNgb(rast, 1, '8BUI', 4,4,
    'rast_avg(float[][], text, text[])::regprocedure, 'NULL', NULL) As ←
    nn_with_border
FROM katrinas_rescaled
limit 1;
```



Das erste Band Unseres Rasters



new raster after averaging pixels withing 4x4 pixels of each other

Siehe auch

[ST_MapAlgebraFct](#), [ST_MapAlgebraExpr](#), [ST_Rescale](#)

9.11.1.11 ST_Reclass

ST_Reclass — Creates a new raster composed of band types reclassified from original. The nband is the band to be changed. If nband is not specified assumed to be 1. All other bands are returned unchanged. Use case: convert a 16BUI band to a 8BUI and so forth for simpler rendering as viewable formats.

Synopsis

```
raster ST_Reclass(raster rast, integer nband, text reclassexpr, text pixeltype, double precision nodataval=NULL);
raster ST_Reclass(raster rast, reclassarg[] VARIADIC reclassargset);
raster ST_Reclass(raster rast, text reclassexpr, text pixeltype);
```

Beschreibung

Creates a new raster formed by applying a valid PostgreSQL algebraic operation defined by the `reclassexpr` on the input raster (`rast`). If no band is specified band 1 is assumed. The new raster will have the same georeference, width, and height as the original raster. Bands not designated will come back unchanged. Refer to [reclassarg](#) for description of valid reclassification expressions.

The bands of the new raster will have pixel type of `pixeltype`. If `reclassargset` is passed in then each `reclassarg` defines behavior of each band generated.

Verfügbarkeit: 2.0.0

Grundlegende Beispiele

Erzeugt einen neuen Raster aus dem Original, indem Band 2 von 8BUI auf 4BUI und alle Werte von 101-254 auf NODATA gesetzt werden.

```
ALTER TABLE dummy_rast ADD COLUMN reclass_rast raster;
UPDATE dummy_rast SET reclass_rast = ST_Reclass(rast,2,'0-87:1-10, 88-100:11-15, ↵
    101-254:0-0', '4BUI',0) WHERE rid = 2;

SELECT i as col, j as row, ST_Value(rast,2,i,j) As origval,
       ST_Value(reclass_rast, 2, i, j) As reclassval,
       ST_Value(reclass_rast, 2, i, j, false) As reclassval_include_nodata
FROM dummy_rast CROSS JOIN generate_series(1, 3) AS i CROSS JOIN generate_series(1,3) AS j
WHERE rid = 2;
```

col	row	origval	reclassval	reclassval_include_nodata
1	1	78	9	9
2	1	98	14	14
3	1	122		0
1	2	96	14	14
2	2	118		0
3	2	180		0
1	3	99	15	15
2	3	112		0
3	3	169		0

Example: Advanced using multiple reclassargs

Create a new raster from the original where band 1,2,3 is converted to 1BB,4BUI, 4BUI respectively and reclassified. Note this uses the variadic `reclassarg` argument which can take as input an indefinite number of reclassargs (theoretically as many bands as you have)

```

UPDATE dummy_rast SET reclass_rast =
    ST_Reclass(rast,
        ROW(2,'0-87]:1-10, (87-100]:11-15, (101-254]:0-0', '4BUI',NULL)::reclassarg,
        ROW(1,'0-253]:1, 254:0', '1BB', NULL)::reclassarg,
        ROW(3,'0-70]:1, (70-86:2, [86-150]:3, [150-255:4', '4BUI', NULL)::reclassarg
    ) WHERE rid = 2;

SELECT i as col, j as row, ST_Value(rast,1,i,j) As ov1, ST_Value(reclass_rast, 1, i, j) As ←
    rv1,
    ST_Value(rast,2,i,j) As ov2, ST_Value(reclass_rast, 2, i, j) As rv2,
    ST_Value(rast,3,i,j) As ov3, ST_Value(reclass_rast, 3, i, j) As rv3
FROM dummy_rast CROSS JOIN generate_series(1, 3) AS i CROSS JOIN generate_series(1,3) AS j
WHERE rid = 2;

```

col	row	ov1	rv1	ov2	rv2	ov3	rv3
1	1	253	1	78	9	70	1
2	1	254	0	98	14	86	3
3	1	253	1	122	0	100	3
1	2	253	1	96	14	80	2
2	2	254	0	118	0	108	3
3	2	254	0	180	0	162	4
1	3	250	1	99	15	90	3
2	3	254	0	112	0	108	3
3	3	254	0	169	0	175	4

Example: Advanced Map a single band 32BF raster to multiple viewable bands

Create a new 3 band (8BUI,8BUI,8BUI viewable raster) from a raster that has only one 32bf band

```

ALTER TABLE wind ADD COLUMN rast_view raster;
UPDATE wind
    set rast_view = ST_AddBand( NULL,
        ARRAY[
            ST_Reclass(rast, 1,'0.1-10]:1-10,9-10]:11, (11-33:0'::text, '8BUI'::text,0),
            ST_Reclass(rast,1, '11-33):0-255,[0-32:0, (34-1000:0'::text, '8BUI'::text,0),
            ST_Reclass(rast,1,'0-32]:0, (32-100:100-255'::text, '8BUI'::text,0)
        ]
    );

```

Siehe auch

[ST_AddBand](#), [ST_Band](#), [ST_BandPixelType](#), [ST_MakeEmptyRaster](#), [reclassarg](#), [ST_Value](#)

9.11.1.12 ST_Union

ST_Union — Gibt die Vereinigung mehrerer Rasterkacheln in einem einzelnen Raster mit mehreren Bändern zurück.

Synopsis

```

raster ST_Union(setof raster rast);
raster ST_Union(setof raster rast, unionarg[] unionargset);
raster ST_Union(setof raster rast, integer nband);
raster ST_Union(setof raster rast, text uniontype);
raster ST_Union(setof raster rast, integer nband, text uniontype);

```


Beschreibung

Returns the union of a set of raster tiles into a single raster composed of at least one band. The resulting raster's extent is the extent of the whole set. In the case of intersection, the resulting value is defined by `uniontype` which is one of the following: LAST (default), FIRST, MIN, MAX, COUNT, SUM, MEAN, RANGE.



Note

In order for rasters to be unioned, they must all have the same alignment. Use [ST_SameAlignment](#) and [ST_NotSameAlignmentReason](#) for more details and help. One way to fix alignment issues is to use [ST_Resample](#) and use the same reference raster for alignment.

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.

Examples: Reconstitute a single band chunked raster tile

```
-- this creates a single band from first band of raster tiles
-- that form the original file system tile
SELECT filename, ST_Union(rast,1) As file_rast
FROM sometable WHERE filename IN('dem01', 'dem02') GROUP BY filename;
```

Examples: Return a multi-band raster that is the union of tiles intersecting geometry

```
-- this creates a multi band raster collecting all the tiles that intersect a line
-- Note: In 2.0, this would have just returned a single band raster
-- , new union works on all bands by default
-- this is equivalent to unionarg: ARRAY[ROW(1, 'LAST'), ROW(2, 'LAST'), ROW(3, 'LAST')]::unionarg[]
SELECT ST_Union(rast)
FROM aerials.boston
WHERE ST_Intersects(rast, ST_GeomFromText('LINESTRING(230486 887771, 230500 88772)',26986) ←
);
```

Examples: Return a multi-band raster that is the union of tiles intersecting geometry

Here we use the longer syntax if we only wanted a subset of bands or we want to change order of bands

```
-- this creates a multi band raster collecting all the tiles that intersect a line
SELECT ST_Union(rast,ARRAY[ROW(2, 'LAST'), ROW(1, 'LAST'), ROW(3, 'LAST')]::unionarg[])
FROM aerials.boston
WHERE ST_Intersects(rast, ST_GeomFromText('LINESTRING(230486 887771, 230500 88772)',26986) ←
);
```

Siehe auch

[unionarg](#), [ST_Envelope](#), [ST_ConvexHull](#), [ST_Clip](#), [ST_Union](#)

9.11.2 Integrierte Map Algebra Callback Funktionen

9.11.2.1 ST_Distinct4ma

ST_Distinct4ma — Raster processing function that calculates the number of unique pixel values in a neighborhood.

Synopsis

```
float8 ST_Distinct4ma(float8[][] matrix, text nodatamode, text[] VARIADIC args);
double precision ST_Distinct4ma(double precision[][] value, integer[][] pos, text[] VARIADIC userargs);
```

Beschreibung

Berechnet die Anzahl der einzelnen Pixelwerte in der Nachbarschaft von Pixel.



Note

Variant 1 is a specialized callback function for use as a callback parameter to [ST_MapAlgebraFctNgb](#).



Note

Variant 2 is a specialized callback function for use as a callback parameter to [ST_MapAlgebra](#).



Warning

Von der Verwendung der Variante 1 wird abgeraten, da [ST_MapAlgebraFctNgb](#) seit 2.1.0 überholt ist.

Verfügbarkeit: 2.0.0

Erweiterung: 2.1.0 Variante 2 wurde hinzugefügt

Beispiele

```
SELECT
  rid,
  st_value(
    st_mapalgebrafctngb(rast, 1, NULL, 1, 1, 'st_distinct4ma(float[][],text,text[])':: ↵
    regprocedure, 'ignore', NULL), 2, 2
  )
FROM dummy_rast
WHERE rid = 2;
  rid | st_value
-----+-----
    2 |        3
(1 row)
```

Siehe auch

[ST_MapAlgebraFctNgb](#), [ST_MapAlgebra](#), [ST_Min4ma](#), [ST_Max4ma](#), [ST_Sum4ma](#), [ST_Mean4ma](#), [ST_Distinct4ma](#), [ST_StdDev4ma](#)

9.11.2.2 ST_InvDistWeight4ma

ST_InvDistWeight4ma — Funktion zur Rasterdatenverarbeitung, die den Wert eines Pixel aus den Pixel der Nachbarschaft interpoliert.

Synopsis

```
double precision ST_InvDistWeight4ma(double precision[][][] value, integer[][] pos, text[] VARIADIC userargs);
```

Beschreibung

Calculate an interpolated value for a pixel using the Inverse Distance Weighted method.

There are two optional parameters that can be passed through `userargs`. The first parameter is the power factor (variable k in the equation below) between 0 and 1 used in the Inverse Distance Weighted equation. If not specified, default value is 1. The second parameter is the weight percentage applied only when the value of the pixel of interest is included with the interpolated value from the neighborhood. If not specified and the pixel of interest has a value, that value is returned.

The basic inverse distance weight equation is:

$$\hat{z}(x_o) = \frac{\sum_{j=1}^m z(x_j) d_{ij}^{-k}}{\sum_{j=1}^m d_{ij}^{-k}}$$

k = power factor, a real number between 0 and 1



Note

This function is a specialized callback function for use as a callback parameter to [ST_MapAlgebra](#).

Verfügbarkeit: 2.1.0

Beispiele

```
-- BEISPIEL GEFRAGT
```

Siehe auch

[ST_MapAlgebra](#), [ST_MinDist4ma](#)

9.11.2.3 ST_Max4ma

ST_Max4ma — Funktion zur Rasterdatenverarbeitung, die den maximalen Zellwert in der Nachbarschaft eines Pixel errechnet.

Synopsis

```
float8 ST_Max4ma(float8[][] matrix, text nodatamode, text[] VARIADIC args);
double precision ST_Max4ma(double precision[][][] value, integer[][] pos, text[] VARIADIC userargs);
```

Beschreibung

Berechnet den maximalen Zellwert in der Nachbarschaft von Pixel.

For Variant 2, a substitution value for NODATA pixels can be specified by passing that value to userargs.



Note

Variant 1 is a specialized callback function for use as a callback parameter to [ST_MapAlgebraFctNgb](#).



Note

Variant 2 is a specialized callback function for use as a callback parameter to [ST_MapAlgebra](#).



Warning

Von der Verwendung der Variante 1 wird abgeraten, da [ST_MapAlgebraFctNgb](#) seit 2.1.0 überholt ist.

Verfügbarkeit: 2.0.0

Erweiterung: 2.1.0 Variante 2 wurde hinzugefügt

Beispiele

```
SELECT
  rid,
  st_value(
    st_mapalgebrafctngb(rast, 1, NULL, 1, 1, 'st_max4ma(float[][],text,text[])':: ↵
    regprocedure, 'ignore', NULL), 2, 2
  )
FROM dummy_rast
WHERE rid = 2;
 rid | st_value
-----+-----
   2 |    254
(1 row)
```

Siehe auch

[ST_MapAlgebraFctNgb](#), [ST_MapAlgebra](#), [ST_Min4ma](#), [ST_Sum4ma](#), [ST_Mean4ma](#), [ST_Range4ma](#), [ST_Distinct4ma](#), [ST_StdDev4ma](#)

9.11.2.4 ST_Mean4ma

ST_Mean4ma — Funktion zur Rasterdatenverarbeitung, die den mittleren Zellwert in der Nachbarschaft von Pixel errechnet.

Synopsis

float8 **ST_Mean4ma**(float8[][] matrix, text nodatamode, text[] VARIADIC args);

double precision **ST_Mean4ma**(double precision[][][] value, integer[][] pos, text[] VARIADIC userargs);

Beschreibung

Berechnet den mittleren Zellwert in der Nachbarschaft von Pixel.

For Variant 2, a substitution value for NODATA pixels can be specified by passing that value to userargs.



Note

Variant 1 is a specialized callback function for use as a callback parameter to [ST_MapAlgebraFctNgb](#).



Note

Variant 2 is a specialized callback function for use as a callback parameter to [ST_MapAlgebra](#).



Warning

Von der Verwendung der Variante 1 wird abgeraten, da [ST_MapAlgebraFctNgb](#) seit 2.1.0 überholt ist.

Verfügbarkeit: 2.0.0

Erweiterung: 2.1.0 Variante 2 wurde hinzugefügt

Beispiele: Variante 1

```
SELECT
  rid,
  st_value(
    st_mapalgebrafctngb(rast, 1, '32BF', 1, 1, 'st_mean4ma(float[][],text,text[])':: ↵
      regprocedure, 'ignore', NULL), 2, 2
  )
FROM dummy_rast
WHERE rid = 2;
  rid |      st_value
-----+-----
    2 | 253.222229003906
(1 row)
```

Beispiele: Variante 2

```
SELECT
  rid,
  st_value(
    ST_MapAlgebra(rast, 1, 'st_mean4ma(double precision[][][], integer[][], text ↵
      [])'::regprocedure,'32BF', 'FIRST', NULL, 1, 1)
    , 2, 2)
FROM dummy_rast
WHERE rid = 2;
  rid |      st_value
-----+-----
    2 | 253.222229003906
(1 row)
```

Siehe auch

[ST_MapAlgebraFctNgb](#), [ST_MapAlgebra](#), [ST_Min4ma](#), [ST_Max4ma](#), [ST_Sum4ma](#), [ST_Range4ma](#), [ST_StdDev4ma](#)

9.11.2.5 ST_Min4ma

ST_Min4ma — Funktion zur Rasterdatenverarbeitung, die den minimalen Zellwert in der Nachbarschaft von Pixel errechnet.

Synopsis

```
float8 ST_Min4ma(float8[][] matrix, text nodatamode, text[] VARIADIC args);
double precision ST_Min4ma(double precision[][][] value, integer[][] pos, text[] VARIADIC userargs);
```

Beschreibung

Berechnet den minimalen Zellwert in der Nachbarschaft von Pixel.

For Variant 2, a substitution value for NODATA pixels can be specified by passing that value to userargs.

**Note**

Variant 1 is a specialized callback function for use as a callback parameter to [ST_MapAlgebraFctNgb](#).

**Note**

Variant 2 is a specialized callback function for use as a callback parameter to [ST_MapAlgebra](#).

**Warning**

Von der Verwendung der Variante 1 wird abgeraten, da [ST_MapAlgebraFctNgb](#) seit 2.1.0 überholt ist.

Verfügbarkeit: 2.0.0

Erweiterung: 2.1.0 Variante 2 wurde hinzugefügt

Beispiele

```
SELECT
  rid,
  st_value(
    st_mapalgebrafctngb(rast, 1, NULL, 1, 1, 'st_min4ma(float[][],text,text[])':: ↵
    regprocedure, 'ignore', NULL), 2, 2
  )
FROM dummy_rast
WHERE rid = 2;
 rid | st_value
-----+-----
   2 |      250
(1 row)
```

Siehe auch

[ST_MapAlgebraFctNgb](#), [ST_MapAlgebra](#), [ST_Max4ma](#), [ST_Sum4ma](#), [ST_Mean4ma](#), [ST_Range4ma](#), [ST_Distinct4ma](#), [ST_StdDev4ma](#)

9.11.2.6 ST_MinDist4ma

ST_MinDist4ma — Raster processing function that returns the minimum distance (in number of pixels) between the pixel of interest and a neighboring pixel with value.

Synopsis

```
double precision ST_MinDist4ma(double precision[][][] value, integer[][] pos, text[] VARIADIC userargs);
```

Beschreibung

Return the shortest distance (in number of pixels) between the pixel of interest and the closest pixel with value in the neighborhood.

**Note**

The intent of this function is to provide an informative data point that helps infer the usefulness of the pixel of interest's interpolated value from [ST_InvDistWeight4ma](#). This function is particularly useful when the neighborhood is sparsely populated.

**Note**

This function is a specialized callback function for use as a callback parameter to [ST_MapAlgebra](#).

Verfügbarkeit: 2.1.0

Beispiele

```
-- BEISPIEL GEFRAGT
```

Siehe auch

[ST_MapAlgebra](#), [ST_InvDistWeight4ma](#)

9.11.2.7 ST_Range4ma

ST_Range4ma — Raster processing function that calculates the range of pixel values in a neighborhood.

Synopsis

```
float8 ST_Range4ma(float8[][] matrix, text nodatamode, text[] VARIADIC args);  
double precision ST_Range4ma(double precision[][][] value, integer[][] pos, text[] VARIADIC userargs);
```

Beschreibung

Calculate the range of pixel values in a neighborhood of pixels.

For Variant 2, a substitution value for NODATA pixels can be specified by passing that value to userargs.



Note

Variant 1 is a specialized callback function for use as a callback parameter to [ST_MapAlgebraFctNgb](#).



Note

Variant 2 is a specialized callback function for use as a callback parameter to [ST_MapAlgebra](#).



Warning

Von der Verwendung der Variante 1 wird abgeraten, da [ST_MapAlgebraFctNgb](#) seit 2.1.0 überholt ist.

Verfügbarkeit: 2.0.0

Erweiterung: 2.1.0 Variante 2 wurde hinzugefügt

Beispiele

```
SELECT
  rid,
  st_value(
    st_mapalgebrafctngb(rast, 1, NULL, 1, 1, 'st_range4ma(float[][],text,text[])'::: ↵
    regprocedure, 'ignore', NULL), 2, 2
  )
FROM dummy_rast
WHERE rid = 2;
  rid | st_value
-----+-----
    2 |         4
(1 row)
```

Siehe auch

[ST_MapAlgebraFctNgb](#), [ST_MapAlgebra](#), [ST_Min4ma](#), [ST_Max4ma](#), [ST_Sum4ma](#), [ST_Mean4ma](#), [ST_Distinct4ma](#), [ST_StdDev4ma](#)

9.11.2.8 ST_StdDev4ma

ST_StdDev4ma — Funktion zur Rasterdatenverarbeitung, welche die Standardabweichung der Zellwerte in der Nachbarschaft von Pixel errechnet.

Synopsis

```
float8 ST_StdDev4ma(float8[][] matrix, text nodatamode, text[] VARIADIC args);
double precision ST_StdDev4ma(double precision[][] value, integer[][] pos, text[] VARIADIC userargs);
```


Beschreibung

Berechnet die Standardabweichung der Zellwerte in der Nachbarschaft von Pixel.



Note

Variant 1 is a specialized callback function for use as a callback parameter to [ST_MapAlgebraFctNgb](#).



Note

Variant 2 is a specialized callback function for use as a callback parameter to [ST_MapAlgebra](#).



Warning

Von der Verwendung der Variante 1 wird abgeraten, da [ST_MapAlgebraFctNgb](#) seit 2.1.0 überholt ist.

Verfügbarkeit: 2.0.0

Erweiterung: 2.1.0 Variante 2 wurde hinzugefügt

Beispiele

```
SELECT
  rid,
  st_value(
    st_mapalgebrafctngb(rast, 1, '32BF', 1, 1, 'st_stddev4ma(float[][],text,text[])':: ↵
      regprocedure, 'ignore', NULL), 2, 2
  )
FROM dummy_rast
WHERE rid = 2;
rid |      st_value
-----+-----
  2 | 1.30170822143555
(1 row)
```

Siehe auch

[ST_MapAlgebraFctNgb](#), [ST_MapAlgebra](#), [ST_Min4ma](#), [ST_Max4ma](#), [ST_Sum4ma](#), [ST_Mean4ma](#), [ST_Distinct4ma](#), [ST_StdDev4ma](#)

9.11.2.9 ST_Sum4ma

ST_Sum4ma — Funktion zur Rasterdatenverarbeitung, die die Summe aller Zellwerte in der Nachbarschaft von Pixel errechnet.

Synopsis

`float8` **ST_Sum4ma**(`float8[][]` matrix, `text` nodatamode, `text[]` VARIADIC args);
`double precision` **ST_Sum4ma**(`double precision[][][]` value, `integer[][]` pos, `text[]` VARIADIC userargs);

Beschreibung

Berechnet die Summe aller Zellwerte in der Nachbarschaft von Pixel.

For Variant 2, a substitution value for NODATA pixels can be specified by passing that value to userargs.



Note

Variant 1 is a specialized callback function for use as a callback parameter to [ST_MapAlgebraFctNgb](#).



Note

Variant 2 is a specialized callback function for use as a callback parameter to [ST_MapAlgebra](#).



Warning

Von der Verwendung der Variante 1 wird abgeraten, da [ST_MapAlgebraFctNgb](#) seit 2.1.0 überholt ist.

Verfügbarkeit: 2.0.0

Erweiterung: 2.1.0 Variante 2 wurde hinzugefügt

Beispiele

```
SELECT
  rid,
  st_value(
    st_mapalgebrafctngb(rast, 1, '32BF', 1, 1, 'st_sum4ma(float[][],text,text[])':: ↵
      regprocedure, 'ignore', NULL), 2, 2
  )
FROM dummy_rast
WHERE rid = 2;
 rid | st_value
-----+-----
   2 |    2279
(1 row)
```

Siehe auch

[ST_MapAlgebraFctNgb](#), [ST_MapAlgebra](#), [ST_Min4ma](#), [ST_Max4ma](#), [ST_Mean4ma](#), [ST_Range4ma](#), [ST_Distinct4ma](#), [ST_StdDev4ma](#)

9.11.3 DHM (Digitales Höhenmodell)

9.11.3.1 ST_Aspect

ST_Aspect — Returns the aspect (in degrees by default) of an elevation raster band. Useful for analyzing terrain.

Synopsis

raster **ST_Aspect**(raster rast, integer band=1, text pixeltype=32BF, text units=DEGREES, boolean interpolate_nodata=FALSE);
 raster **ST_Aspect**(raster rast, integer band, raster customextent, text pixeltype=32BF, text units=DEGREES, boolean interpolate_nodata=FALSE);


```

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](#), [ST_TRI](#), [ST_TPI](#), [ST_Roughness](#), [ST_HillShade](#), [ST_Slope](#)

9.11.3.2 ST_HillShade

ST_HillShade — Returns the hypothetical illumination of an elevation raster band using provided azimuth, altitude, brightness and scale inputs.

Synopsis

raster **ST_HillShade**(raster rast, integer band=1, text pixeltype=32BF, double precision azimuth=315, double precision altitude=45, double precision max_bright=255, double precision scale=1.0, boolean interpolate_nodata=FALSE);
 raster **ST_HillShade**(raster rast, integer band, raster customextent, text pixeltype=32BF, double precision azimuth=315, double precision altitude=45, double precision max_bright=255, double precision scale=1.0, boolean interpolate_nodata=FALSE);

Beschreibung

Returns the hypothetical illumination of an elevation raster band using the azimuth, altitude, brightness, and scale inputs. Utilizes map algebra and applies the hill shade equation to neighboring pixels. Return pixel values are between 0 and 255.

`azimuth` is a value between 0 and 360 degrees measured clockwise from North.

`altitude` is a value between 0 and 90 degrees where 0 degrees is at the horizon and 90 degrees is directly overhead.

`max_bright` is a value between 0 and 255 with 0 as no brightness and 255 as max brightness.

`scale` is the ratio of vertical units to horizontal. For Feet:LatLon use `scale=370400`, for Meters:LatLon use `scale=111120`.

If `interpolate_nodata` is TRUE, values for NODATA pixels from the input raster will be interpolated using [ST_InvDistWeight4ma](#) before computing the hillshade illumination.

**Note**

For more information about Hillshade, please refer to [How hillshade works](#).

Verfügbarkeit: 2.0.0

Enhanced: 2.1.0 Uses `ST_MapAlgebra()` and added optional `interpolate_nodata` function parameter

Changed: 2.1.0 In prior versions, azimuth and altitude were expressed in radians. Now, azimuth and altitude are expressed in degrees

Beispiele: Variante 1

```
WITH foo AS (
    SELECT ST_SetValues(
        ST_AddBand(ST_MakeEmptyRaster(5, 5, 0, 0, 1, -1, 0, 0, 0), 1, '32BF', 0, ↵
        -9999),
        1, 1, 1, ARRAY[
            [1, 1, 1, 1, 1],
            [1, 2, 2, 2, 1],
            [1, 2, 3, 2, 1],
            [1, 2, 2, 2, 1],
            [1, 1, 1, 1, 1]
        ]::double precision[][])
    ) AS rast
)
SELECT
    ST_DumpValues(ST_Hillshade(rast, 1, '32BF'))
FROM foo
```

```
-----
(1, "{ {NULL,NULL,NULL,NULL,NULL}, {NULL,251.32763671875,220.749786376953,147.224319458008, ↵
NULL}, {NULL,220.749786376953,180.312225341797,67.7497863769531,NULL}, {NULL ↵
,147.224319458008
,67.7497863769531,43.1210060119629,NULL}, {NULL,NULL,NULL,NULL,NULL}}")
(1 row)
```

Beispiele: Variante 2

Complete example of tiles of a coverage. This query only works with PostgreSQL 9.1 or higher.

```
WITH foo AS (
    SELECT ST_Tile(
        ST_SetValues(
            ST_AddBand(
                ST_MakeEmptyRaster(6, 6, 0, 0, 1, -1, 0, 0, 0),
                1, '32BF', 0, -9999
            ),
            1, 1, 1, ARRAY[
                [1, 1, 1, 1, 1, 1],
                [1, 1, 1, 1, 2, 1],
                [1, 2, 2, 3, 3, 1],
                [1, 1, 3, 2, 1, 1],
            ]
        )
    )
```

```

                [1, 2, 2, 1, 2, 1],
                [1, 1, 1, 1, 1, 1]
            ]::double precision[]
        ),
        2, 2
    ) AS rast
)
SELECT
    t1.rast,
    ST_Hillshade(ST_Union(t2.rast), 1, t1.rast)
FROM foo t1
CROSS JOIN foo t2
WHERE ST_Intersects(t1.rast, t2.rast)
GROUP BY t1.rast;

```

Siehe auch

[ST_MapAlgebra](#), [ST_TRI](#), [ST_TPI](#), [ST_Roughness](#), [ST_Aspect](#), [ST_Slope](#)

9.11.3.3 ST_Roughness

ST_Roughness — Returns a raster with the calculated "roughness" of a DEM.

Synopsis

raster **ST_Roughness**(raster rast, integer nband, raster customextent, text pixeltype="32BF" , boolean interpolate_nodata=FALSE);

Beschreibung

Calculates the "roughness" of a DEM, by subtracting the maximum from the minimum for a given area.

Verfügbarkeit: 2.1.0

Beispiele

```
-- Beispiel benötigt
```

Siehe auch

[ST_MapAlgebra](#), [ST_TRI](#), [ST_TPI](#), [ST_Slope](#), [ST_HillShade](#), [ST_Aspect](#)

9.11.3.4 ST_Slope

ST_Slope — Returns the slope (in degrees by default) of an elevation raster band. Useful for analyzing terrain.

Synopsis

raster **ST_Slope**(raster rast, integer nband=1, text pixeltype=32BF, text units=DEGREES, double precision scale=1.0, boolean interpolate_nodata=FALSE);
raster **ST_Slope**(raster rast, integer nband, raster customextent, text pixeltype=32BF, text units=DEGREES, double precision scale=1.0, boolean interpolate_nodata=FALSE);

Beschreibung

Returns the slope (in degrees by default) of an elevation raster band. Utilizes map algebra and applies the slope equation to neighboring pixels.

`units` indicates the units of the slope. Possible values are: RADIANS, DEGREES (default), PERCENT.

`scale` is the ratio of vertical units to horizontal. For Feet:LatLon use `scale=370400`, for Meters:LatLon use `scale=111120`.

If `interpolate_nodata` is TRUE, values for NODATA pixels from the input raster will be interpolated using [ST_InvDistWeight4ma](#) before computing the surface slope.



Note

For more information about Slope, Aspect and Hillshade, please refer to [ESRI - How hillshade works](#) and [ERDAS Field Guide - Slope Images](#).

Verfügbarkeit: 2.0.0

Enhanced: 2.1.0 Uses `ST_MapAlgebra()` and added optional `units`, `scale`, `interpolate_nodata` function parameters

Changed: 2.1.0 In prior versions, return values were in radians. Now, return values default to degrees

Beispiele: Variante 1

```
WITH foo AS (
    SELECT ST_SetValues(
        ST_AddBand(ST_MakeEmptyRaster(5, 5, 0, 0, 1, -1, 0, 0, 0), 1, '32BF', 0, -9999),
        1, 1, 1, ARRAY[
            [1, 1, 1, 1, 1],
            [1, 2, 2, 2, 1],
            [1, 2, 3, 2, 1],
            [1, 2, 2, 2, 1],
            [1, 1, 1, 1, 1]
        ]::double precision[][]) AS rast
)
SELECT
    ST_DumpValues(ST_Slope(rast, 1, '32BF'))
FROM foo
```

st_dumpvalues
(1, "{10.0249881744385,21.5681285858154,26.5650520324707,21.5681285858154,10.0249881744385},{21.5681285858154,26.5650520324707,36.8698959350586,0,36.8698959350586,26.5650520324707},{21.5681285858154,35.26438905681285858154,26.5650520324707,21.5681285858154,10.0249881744385}}")

(1 row)

Beispiele: Variante 2

Complete example of tiles of a coverage. This query only works with PostgreSQL 9.1 or higher.

```

WITH foo AS (
    SELECT ST_Tile(
        ST_SetValues(
            ST_AddBand(
                ST_MakeEmptyRaster(6, 6, 0, 0, 1, -1, 0, 0, 0),
                1, '32BF', 0, -9999
            ),
            1, 1, 1, ARRAY[
                [1, 1, 1, 1, 1, 1],
                [1, 1, 1, 1, 2, 1],
                [1, 2, 2, 3, 3, 1],
                [1, 1, 3, 2, 1, 1],
                [1, 2, 2, 1, 2, 1],
                [1, 1, 1, 1, 1, 1]
            ]::double precision[]
        ),
        2, 2
    ) AS rast
)
SELECT
    t1.rast,
    ST_Slope(ST_Union(t2.rast), 1, t1.rast)
FROM foo t1
CROSS JOIN foo t2
WHERE ST_Intersects(t1.rast, t2.rast)
GROUP BY t1.rast;

```

Siehe auch

[ST_MapAlgebra](#), [ST_TRI](#), [ST_TPI](#), [ST_Roughness](#), [ST_HillShade](#), [ST_Aspect](#)

9.11.3.5 ST_TPI

ST_TPI — Returns a raster with the calculated Topographic Position Index.

Synopsis

raster **ST_TPI**(raster rast, integer nband, raster customextent, text pixeltype="32BF" , boolean interpolate_nodata=FALSE);

Beschreibung

Calculates the Topographic Position Index, which is defined as the focal mean with radius of one minus the center cell.



Note

This function only supports a focalmean radius of one.

Verfügbarkeit: 2.1.0

Beispiele

```
-- Beispiel benötigt
```


Siehe auch

[ST_MapAlgebra](#), [ST_TRI](#), [ST_Roughness](#), [ST_Slope](#), [ST_HillShade](#), [ST_Aspect](#)

9.11.3.6 ST_TRI

ST_TRI — Returns a raster with the calculated Terrain Ruggedness Index.

Synopsis

```
raster ST_TRI(raster rast, integer nband, raster customextent, text pixeltype="32BF" , boolean interpolate_nodata=FALSE );
```

Beschreibung

Terrain Ruggedness Index is calculated by comparing a central pixel with its neighbors, taking the absolute values of the differences, and averaging the result.

**Note**

This function only supports a focalmean radius of one.

Verfügbarkeit: 2.1.0

Beispiele

```
-- Beispiel benötigt
```

Siehe auch

[ST_MapAlgebra](#), [ST_Roughness](#), [ST_TPI](#), [ST_Slope](#), [ST_HillShade](#), [ST_Aspect](#)

9.11.4 Raster to Geometry**9.11.4.1 Box3D**

Box3D — Returns the box 3d representation of the enclosing box of the raster.

Synopsis

```
box3d Box3D(raster rast);
```

Beschreibung

Returns the box representing the extent of the raster.

The polygon is defined by the corner points of the bounding box ((MINX, MINY), (MAXX, MAXY))

Changed: 2.0.0 In pre-2.0 versions, there used to be a box2d instead of box3d. Since box2d is a deprecated type, this was changed to box3d.

Beispiele

```
SELECT
    rid,
    Box3D(rast) AS rastbox
FROM dummy_rast;
```

```
rid |          rastbox
----+-----
1   | BOX3D(0.5 0.5 0,20.5 60.5 0)
2   | BOX3D(3427927.75 5793243.5 0,3427928 5793244 0)
```

Siehe auch

[ST_Envelope](#)

9.11.4.2 ST_ConvexHull

ST_ConvexHull — Return the convex hull geometry of the raster including pixel values equal to `BandNoDataValue`. For regular shaped and non-skewed rasters, this gives the same result as `ST_Envelope` so only useful for irregularly shaped or skewed rasters.

Synopsis

geometry **ST_ConvexHull**(raster rast);

Beschreibung

Return the convex hull geometry of the raster including the `NoDataBandValue` band pixels. For regular shaped and non-skewed rasters, this gives more or less the same result as `ST_Envelope` so only useful for irregularly shaped or skewed rasters.



Note

`ST_Envelope` floors the coordinates and hence add a little buffer around the raster so the answer is subtly different from `ST_ConvexHull` which does not floor.

Beispiele

Siehe [PostGIS Raster Specification](#) für eine entsprechende Darstellung.

```
-- Note envelope and convexhull are more or less the same
SELECT ST_AsText(ST_ConvexHull(rast)) As convhull,
       ST_AsText(ST_Envelope(rast)) As env
FROM dummy_rast WHERE rid=1;
```

```
convhull | env
-----+-----
POLYGON((0.5 0.5,20.5 0.5,20.5 60.5,0.5 60.5,0.5 0.5)) | POLYGON((0 0,20 0,20 60,0 60,0 0) ←
)
```

```
-- now we skew the raster
-- note how the convex hull and envelope are now different
SELECT ST_AsText(ST_ConvexHull(rast)) As convhull,
       ST_AsText(ST_Envelope(rast)) As env
FROM (SELECT ST_SetRotation(rast, 0.1, 0.1) As rast
```

```

FROM dummy_rast WHERE rid=1) As foo;

convhull | env
-----+-----
POLYGON((0.5 0.5,20.5 1.5,22.5 61.5,2.5 60.5,0.5 0.5)) | POLYGON((0 0,22 0,22 61,0 61,0 0)
)

```

Siehe auch

[ST_Envelope](#), [ST_MinConvexHull](#), [ST_ConvexHull](#), [ST_AsText](#)

9.11.4.3 ST_DumpAsPolygons

ST_DumpAsPolygons — Returns a set of geomval (geom,val) rows, from a given raster band. If no band number is specified, band num defaults to 1.

Synopsis

setof geomval **ST_DumpAsPolygons**(raster rast, integer band_num=1, boolean exclude_nodata_value=TRUE);

Beschreibung

This is a set-returning function (SRF). It returns a set of geomval rows, formed by a geometry (geom) and a pixel band value (val). Each polygon is the union of all pixels for that band that have the same pixel value denoted by val.

ST_DumpAsPolygon is useful for polygonizing rasters. It is the reverse of a GROUP BY in that it creates new rows. For example it can be used to expand a single raster into multiple POLYGONS/MULTIPOLYGONS.

Verfügbarkeit: Benötigt GDAL 1.7+



Note

If there is a no data value set for a band, pixels with that value will not be returned except in the case of exclude_nodata_value=false.



Note

If you only care about count of pixels with a given value in a raster, it is faster to use [ST_ValueCount](#).



Note

This is different than ST_PixelAsPolygons where one geometry is returned for each pixel regardless of pixel value.

Beispiele

```

-- this syntax requires PostgreSQL 9.3+
SELECT val, ST_AsText(geom) As geomwkt
FROM (
SELECT dp.*
FROM dummy_rast, LATERAL ST_DumpAsPolygons(rast) AS dp

```

```
WHERE rid = 2
) As foo
WHERE val BETWEEN 249 and 251
ORDER BY val;
```

val	geomwkt
249	POLYGON((3427927.95 5793243.95,3427927.95 5793243.85,3427928 5793243.85,3427928 5793243.95,3427927.95 5793243.95))
250	POLYGON((3427927.75 5793243.9,3427927.75 5793243.85,3427927.8 5793243.85,3427927.8 5793243.9,3427927.75 5793243.9))
250	POLYGON((3427927.8 5793243.8,3427927.8 5793243.75,3427927.85 5793243.75,3427927.85 5793243.8, 3427927.8 5793243.8))
251	POLYGON((3427927.75 5793243.85,3427927.75 5793243.8,3427927.8 5793243.8,3427927.8 5793243.85,3427927.75 5793243.85))

Siehe auch

[geomval](#), [ST_Value](#), [ST_Polygon](#), [ST_ValueCount](#)

9.11.4.4 ST_Envelope

ST_Envelope — Returns the polygon representation of the extent of the raster.

Synopsis

geometry **ST_Envelope**(raster rast);

Beschreibung

Returns the polygon representation of the extent of the raster in spatial coordinate units defined by srid. It is a float8 minimum bounding box represented as a polygon.

The polygon is defined by the corner points of the bounding box ((MINX, MINY), (MINX, MAXY), (MAXX, MAXY), (MAXX, MINY), (MINX, MINY))

Beispiele

```
SELECT rid, ST_AsText(ST_Envelope(rast)) As envgeomwkt
FROM dummy_rast;
```

rid	envgeomwkt
1	POLYGON((0 0,20 0,20 60,0 60,0 0))
2	POLYGON((3427927 5793243,3427928 5793243,3427928 5793244,3427927 5793244, 3427927 5793243))

Siehe auch

[ST_Envelope](#), [ST_AsText](#), [ST_SRID](#)

9.11.4.5 ST_MinConvexHull

ST_MinConvexHull — Return the convex hull geometry of the raster excluding NODATA pixels.

Synopsis

geometry **ST_MinConvexHull**(raster rast, integer nband=NULL);

Beschreibung

Return the convex hull geometry of the raster excluding NODATA pixels. If nband is NULL, all bands of the raster are considered.

Verfügbarkeit: 2.1.0

Beispiele

```
WITH foo AS (
    SELECT
        ST_SetValues(
            ST_SetValues(
                ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(9, 9, 0, 0, 1, -1, ↵
                    0, 0, 0), 1, '8BUI', 0, 0), 2, '8BUI', 1, 0),
                1, 1, 1,
                ARRAY[
                    [0, 0, 0, 0, 0, 0, 0, 0, 0],
                    [0, 0, 0, 0, 0, 0, 0, 0, 0],
                    [0, 0, 0, 0, 0, 0, 0, 0, 0],
                    [0, 0, 0, 1, 0, 0, 0, 0, 1],
                    [0, 0, 0, 1, 1, 0, 0, 0, 0],
                    [0, 0, 0, 1, 0, 0, 0, 0, 0],
                    [0, 0, 0, 0, 0, 0, 0, 0, 0],
                    [0, 0, 0, 0, 0, 0, 0, 0, 0],
                    [0, 0, 0, 0, 0, 0, 0, 0, 0]
                ]::double precision[][])
            ),
            2, 1, 1,
            ARRAY[
                [0, 0, 0, 0, 0, 0, 0, 0, 0],
                [0, 0, 0, 0, 0, 0, 0, 0, 0],
                [0, 0, 0, 0, 0, 0, 0, 0, 0],
                [1, 0, 0, 0, 0, 1, 0, 0, 0],
                [0, 0, 0, 0, 1, 1, 0, 0, 0],
                [0, 0, 0, 0, 0, 1, 0, 0, 0],
                [0, 0, 0, 0, 0, 0, 0, 0, 0],
                [0, 0, 0, 0, 0, 0, 0, 0, 0],
                [0, 0, 1, 0, 0, 0, 0, 0, 0]
            ]::double precision[][])
        ) AS rast
    )
SELECT
    ST_AsText(ST_ConvexHull(rast)) AS hull,
    ST_AsText(ST_MinConvexHull(rast)) AS mhull,
    ST_AsText(ST_MinConvexHull(rast, 1)) AS mhull_1,
    ST_AsText(ST_MinConvexHull(rast, 2)) AS mhull_2
FROM foo
```

hull	mhull_1	mhull	mhull_2
POLYGON((0 0,9 0,9 -9,0 -9,0 0))	POLYGON((0 -3,9 -3,9 -9,0 -9,0 -3))	POLYGON((3 -3,9 -3,9 -6,3 -6,3 -3))	POLYGON((0 -3,6 -3,6 -9,0 -9,0 -3))

Siehe auch

[ST_Envelope](#), [ST_ConvexHull](#), [ST_ConvexHull](#), [ST_AsText](#)

9.11.4.6 ST_Polygon

ST_Polygon — Returns a multipolygon geometry formed by the union of pixels that have a pixel value that is not no data value. If no band number is specified, band num defaults to 1.

Synopsis

geometry **ST_Polygon**(raster rast, integer band_num=1);

Beschreibung

Verfügbarkeit: 0.1.6 - benötigt GDAL 1.7+

Enhanced: 2.1.0 Improved Speed (fully C-Based) and the returning multipolygon is ensured to be valid.

Changed: 2.1.0 In prior versions would sometimes return a polygon, changed to always return multipolygon.

Beispiele

```
-- by default no data band value is 0 or not set, so polygon will return a square polygon
SELECT ST_AsText(ST_Polygon(rast)) As geomwkt
FROM dummy_rast
WHERE rid = 2;

geomwkt
-----
MULTIPOLYGON(((3427927.75 5793244,3427928 5793244,3427928 5793243.75,3427927.75 5793243.75,3427927.75 5793244)))

-- now we change the no data value of first band
UPDATE dummy_rast SET rast = ST_SetBandNoDataValue(rast,1,254)
WHERE rid = 2;
SELECT rid, ST_BandNoDataValue(rast)
from dummy_rast where rid = 2;

-- ST_Polygon excludes the pixel value 254 and returns a multipolygon
SELECT ST_AsText(ST_Polygon(rast)) As geomwkt
FROM dummy_rast
WHERE rid = 2;

geomwkt
-----
MULTIPOLYGON(((3427927.9 5793243.95,3427927.85 5793243.95,3427927.85 5793244,3427927.9 5793244,3427927.9 5793243.95)),((3427928 5793243.85,3427928 5793243.8,3427927.95 5793243.8,3427927.95 5793243.85,3427927.9 5793243.9,3427927.9 5793243.95,3427927.95 5793243.95,3427928 5793243.95,3427928 5793243.85)),((3427927.8 5793243.75,3427927.75 5793243.75,3427927.75 5793243.8,3427927.75 5793243.85,3427927.75 5793243.9,3427927.75 5793244,3427927.8 5793244,3427927.8 5793243.9,3427927.8 5793243.85,3427927.85 5793243.85,3427927.85 5793243.8,3427927.85 5793243.75,3427927.8 5793243.75)))

-- Or if you want the no data value different for just one time
SELECT ST_AsText (
```

```

        ST_Polygon(
            ST_SetBandNoDataValue(rast,1,252)
        ) As geomwkt
FROM dummy_rast
WHERE rid =2;

geomwkt
-----
MULTIPOLYGON(((3427928 5793243.85,3427928 5793243.8,3427928 5793243.75,3427927.85 ↵
5793243.75,3427927.8 5793243.75,3427927.8 5793243.8,3427927.75 5793243.8,3427927.75 ↵
5793243.85,3427927.75 5793243.9,3427927.75 5793244,3427927.8 5793244,3427927.85 ↵
5793244,3427927.9 5793244,3427928 5793244,3427928 5793243.95,3427928 5793243.85) ↵
,(3427927.9 5793243.9,3427927.9 5793243.85,3427927.95 5793243.85,3427927.95 ↵
5793243.9,3427927.9 5793243.9)))

```

Siehe auch

[ST_Value](#), [ST_DumpAsPolygons](#)

9.12 Rasteroperatoren

9.12.1 &&

&& — Returns TRUE if A's bounding box intersects B's bounding box.

Synopsis

```

boolean &&( raster A , raster B );
boolean &&( raster A , geometry B );
boolean &&( geometry B , raster A );

```

Beschreibung

The **&&** operator returns TRUE if the bounding box of raster/geometr A intersects the bounding box of raster/geometr B.



Note

This operand will make use of any indexes that may be available on the rasters.

Verfügbarkeit: 2.0.0

Beispiele

```

SELECT A.rid As a_rid, B.rid As b_rid, A.rast && B.rast As intersect
FROM dummy_rast AS A CROSS JOIN dummy_rast AS B LIMIT 3;

```

```

a_rid | b_rid | intersect
-----+-----+-----
2 | 2 | t
2 | 3 | f
2 | 1 | f

```

9.12.2 &<

&< — Returns `TRUE` if A's bounding box is to the left of B's.

Synopsis

boolean **&<**(raster A , raster B);

Beschreibung

The **&<** operator returns `TRUE` if the bounding box of raster A overlaps or is to the left of the bounding box of raster B, or more accurately, overlaps or is **NOT** to the right of the bounding box of raster B.



Note

This operand will make use of any indexes that may be available on the rasters.

Beispiele

```
SELECT A.rid As a_rid, B.rid As b_rid, A.rast &< B.rast As overleft
FROM dummy_rast AS A CROSS JOIN dummy_rast AS B;
```

a_rid	b_rid	overleft
2	2	t
2	3	f
2	1	f
3	2	t
3	3	t
3	1	f
1	2	t
1	3	t
1	1	t

9.12.3 &>

&> — Returns `TRUE` if A's bounding box is to the right of B's.

Synopsis

boolean **&>**(raster A , raster B);

Beschreibung

The **&>** operator returns `TRUE` if the bounding box of raster A overlaps or is to the right of the bounding box of raster B, or more accurately, overlaps or is **NOT** to the left of the bounding box of raster B.



Note

This operand will make use of any indexes that may be available on the geometries.

Beispiele

```
SELECT A.rid As a_rid, B.rid As b_rid, A.rast &> B.rast As overright
FROM dummy_rast AS A CROSS JOIN dummy_rast AS B;
```

a_rid	b_rid	overright
2	2	t
2	3	t
2	1	t
3	2	f
3	3	t
3	1	f
1	2	f
1	3	t
1	1	t

9.12.4 =

= — Returns TRUE if A's bounding box is the same as B's. Uses double precision bounding box.

Synopsis

```
boolean =( raster A , raster B );
```

Beschreibung

The = operator returns TRUE if the bounding box of raster A is the same as the bounding box of raster B. PostgreSQL uses the =, <, and > operators defined for rasters to perform internal orderings and comparison of rasters (ie. in a GROUP BY or ORDER BY clause).



Caution

This operand will NOT make use of any indexes that may be available on the rasters. Use ~= instead. This operator exists mostly so one can group by the raster column.

Verfügbarkeit: 2.1.0

Siehe auch

~=

9.12.5 @

@ — Returns TRUE if A's bounding box is contained by B's. Uses double precision bounding box.

Synopsis

```
boolean @( raster A , raster B );
boolean @( geometry A , raster B );
boolean @( raster B , geometry A );
```

Beschreibung

The @ operator returns TRUE if the bounding box of raster/geometry A is contained by bounding box of raster/geometry B.



Note

Dieser Operand verwendet die räumlichen Indizes der Raster.

Verfügbarkeit: 2.0.0 raster @ raster, und raster @ geometry eingeführt

Availability: 2.0.5 geometry @ raster introduced

Siehe auch

~

9.12.6 ~=

~= — Gibt TRUE zurück wenn die Umgebungsrechtecke von "A" und "B" ident sind.

Synopsis

boolean ~= (raster A , raster B);

Beschreibung

Der Operator ~= gibt TRUE zurück, wenn die Umgebungsrechtecke der Raster "A" und "B" ident sind.



Note

This operand will make use of any indexes that may be available on the rasters.

Verfügbarkeit: 2.0.0

Beispiele

Very useful usecase is for taking two sets of single band rasters that are of the same chunk but represent different themes and creating a multi-band raster

```
SELECT ST_AddBand(prec.rast, alt.rast) As new_rast
FROM prec INNER JOIN alt ON (prec.rast ~= alt.rast);
```

Siehe auch

ST_AddBand, =

9.12.7 ~

~ — Returns TRUE if A's bounding box is contains B's. Uses double precision bounding box.

Synopsis

```
boolean ~( raster A , raster B );
boolean ~( geometry A , raster B );
boolean ~( raster B , geometry A );
```

Beschreibung

The ~ operator returns `TRUE` if the bounding box of raster/geometry A contains bounding box of raster/geometry B.

**Note**

Dieser Operand verwendet die räumlichen Indizes der Raster.

Verfügbarkeit: 2.0.0

Siehe auch

@

9.13 Räumliche Beziehungen von Rastern und Rasterbändern

9.13.1 ST_Contains

ST_Contains — Return true if no points of raster `rastB` lie in the exterior of raster `rastA` and at least one point of the interior of `rastB` lies in the interior of `rastA`.

Synopsis

```
boolean ST_Contains( raster rastA , integer nbandA , raster rastB , integer nbandB );
boolean ST_Contains( raster rastA , raster rastB );
```

Beschreibung

Raster `rastA` contains `rastB` if and only if no points of `rastB` lie in the exterior of `rastA` and at least one point of the interior of `rastB` lies in the interior of `rastA`. If the band number is not provided (or set to `NULL`), only the convex hull of the raster is considered in the test. If the band number is provided, only those pixels with value (not `NODATA`) are considered in the test.

**Note**

Diese Funktion verwendet die für Raster verfügbaren Indizes.

**Note**

Um die räumliche Beziehung zwischen einem Raster und einem geometrischen Datentyp zu überprüfen, verwenden Sie bitte `ST_Polygon` für den Raster, z.B. `ST_Contains(ST_Polygon(raster), geometry)` or `ST_Contains(geometry, ST_Polygon(raster))`.

**Note**

ST_Contains() ist das Gegenstück zu ST_Within(). Daher impliziert ST_Contains(rastA, rastB) ST_Within(rastB, rastA).

Verfügbarkeit: 2.1.0

Beispiele

```
-- mit Nummern für die Bänder
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;
```

-- ANMERKUNG: Der erste Raster der geliefert wird hat keine Bänder

```
rid | rid | st_contains
-----+-----+-----
  1 |  1 |
  1 |  2 | f
```

-- ohne Angabe von Nummern für die Bänder

```
SELECT r1.rid, r2.rid, ST_Contains(r1.rast, r2.rast) FROM dummy_rast r1 CROSS JOIN ↔
    dummy_rast r2 WHERE r1.rid = 1;
rid | rid | st_contains
-----+-----+-----
  1 |  1 | t
  1 |  2 | f
```

Siehe auch

[ST_Intersects](#), [ST_Within](#)

9.13.2 ST_ContainsProperly

ST_ContainsProperly — Gibt TRUE zurück, wenn "rastB" das Innere von "rastA" schneidet, aber nicht die Begrenzung oder das Äußere von "rastA".

Synopsis

```
boolean ST_ContainsProperly( raster rastA , integer nbandA , raster rastB , integer nbandB );
boolean ST_ContainsProperly( raster rastA , raster rastB );
```

Beschreibung

Raster rastA contains properly rastB if rastB intersects the interior of rastA but not the boundary or exterior of rastA. If the band number is not provided (or set to NULL), only the convex hull of the raster is considered in the test. If the band number is provided, only those pixels with value (not NODATA) are considered in the test.

Raster "rastA" enthält sich selbst, aber enthält sich nicht zur Gänze selbst.

**Note**

Diese Funktion verwendet die für Raster verfügbaren Indizes.

**Note**

Um die räumliche Beziehung zwischen einem Raster und einem geometrischen Datentyp zu überprüfen, verwenden Sie bitte `ST_Polygon` für den Raster, z.B. `ST_ContainsProperly(ST_Polygon(raster), geometry)` or `ST_ContainsProperly(geometry, ST_Polygon(raster))`.

Verfügbarkeit: 2.1.0

Beispiele

```
SELECT r1.rid, r2.rid, ST_ContainsProperly(r1.rast, 1, r2.rast, 1) FROM dummy_rast r1 CROSS JOIN
dummy_rast r2 WHERE r1.rid = 2;
```

rid	rid	st_containsproperly
2	1	f
2	2	f

Siehe auch

[ST_Intersects](#), [ST_Contains](#)

9.13.3 ST_Covers

`ST_Covers` — Gibt TRUE zurück, wenn kein Punkt des Rasters "rastB" außerhalb des Rasters "rastA" liegt.

Synopsis

```
boolean ST_Covers( raster rastA , integer nbandA , raster rastB , integer nbandB );
boolean ST_Covers( raster rastA , raster rastB );
```

Beschreibung

Raster "rastA" deckt "rastB" dann und nur dann ab, wenn kein Punkt von "rastB" im Äußeren von "rastA" liegt. Wenn die Bandnummer nicht angegeben ist (oder auf NULL gesetzt), dann wird nur die konvexe Hülle des Rasters zur Überprüfung herangezogen. Wenn die Bandnummer angegeben ist, werden nur die Pixel mit einem Wert (nicht NODATA) bei der Überprüfung herangezogen.

**Note**

Diese Funktion verwendet die für Raster verfügbaren Indizes.

**Note**

Um die räumliche Beziehung zwischen einem Raster und einem geometrischen Datentyp zu überprüfen, verwenden Sie bitte `ST_Polygon` für den Raster, z.B. `ST_Covers(ST_Polygon(raster), geometry)` or `ST_Covers(geometry, ST_Polygon(raster))`.

Verfügbarkeit: 2.1.0

Beispiele

```
SELECT r1.rid, r2.rid, ST_Covers(r1.rast, 1, r2.rast, 1) FROM dummy_rast r1 CROSS JOIN ↵
dummy_rast r2 WHERE r1.rid = 2;
```

rid	rid	st_covers
2	1	f
2	2	t

Siehe auch

[ST_Intersects](#), [ST_CoveredBy](#)

9.13.4 ST_CoveredBy

ST_CoveredBy — Gibt TRUE zurück, wenn kein Punkt des Rasters "rastA" außerhalb des Rasters "rastB" liegt.

Synopsis

boolean **ST_CoveredBy**(raster rastA , integer nbandA , raster rastB , integer nbandB);
 boolean **ST_CoveredBy**(raster rastA , raster rastB);

Beschreibung

Raster rastA is covered by rastB if and only if no points of rastA lie in the exterior of rastB. If the band number is not provided (or set to NULL), only the convex hull of the raster is considered in the test. If the band number is provided, only those pixels with value (not NODATA) are considered in the test.



Note

Diese Funktion verwendet die für Raster verfügbaren Indizes.



Note

Um die räumliche Beziehung zwischen einem Raster und einem geometrischen Datentyp zu überprüfen, verwenden Sie bitte ST_Polygon für den Raster, z.B. ST_CoveredBy(ST_Polygon(raster), geometry) or ST_CoveredBy(geometry, ST_Polygon(raster)).

Verfügbarkeit: 2.1.0

Beispiele

```
SELECT r1.rid, r2.rid, ST_CoveredBy(r1.rast, 1, r2.rast, 1) FROM dummy_rast r1 CROSS JOIN ↵
dummy_rast r2 WHERE r1.rid = 2;
```

rid	rid	st_coveredby
2	1	f
2	2	t

Siehe auch[ST_Intersects](#), [ST_Covers](#)**9.13.5 ST_Disjoint**

ST_Disjoint — Gibt TRUE zurück, wenn sich die Raster "rastA" und "rastB" räumlich nicht überschneiden.

Synopsis

boolean **ST_Disjoint**(raster rastA , integer nbandA , raster rastB , integer nbandB);

boolean **ST_Disjoint**(raster rastA , raster rastB);

Beschreibung

Raster rastA and rastB are disjoint if they do not share any space together. If the band number is not provided (or set to NULL), only the convex hull of the raster is considered in the test. If the band number is provided, only those pixels with value (not NODATA) are considered in the test.

**Note**

Diese Funktion verwendet keine Indizes.

**Note**

Um die räumliche Beziehung zwischen einem Raster und einem geometrischen Datentyp zu überprüfen, verwenden Sie bitte ST_Polygon für den Raster, z.B. ST_Disjoint(ST_Polygon(raster), geometry).

Verfügbarkeit: 2.1.0

Beispiele

```
-- rid = 1 hat keine Bänder, daher die ANMERKUNG und die NULL Werte für "st_disjoint"
SELECT r1.rid, r2.rid, ST_Disjoint(r1.rast, 1, r2.rast, 1) FROM dummy_rast r1 CROSS JOIN ↵
    dummy_rast r2 WHERE r1.rid = 2;
```

-- ANMERKUNG: Der zweite Raster hat keine Bänder

rid	rid	st_disjoint
2	1	
2	2	f

-- diesmal ohne Nummern für die Bänder

```
SELECT r1.rid, r2.rid, ST_Disjoint(r1.rast, r2.rast) FROM dummy_rast r1 CROSS JOIN ↵
    dummy_rast r2 WHERE r1.rid = 2;
```

rid	rid	st_disjoint
2	1	t
2	2	f

Siehe auch[ST_Intersects](#)**9.13.6 ST_Intersects**

ST_Intersects — Gibt TRUE zurück, wenn sich die Raster "rastA" und "rastB" nicht räumlich überschneiden.

Synopsis

```
boolean ST_Intersects( raster rastA , integer nbandA , raster rastB , integer nbandB );
boolean ST_Intersects( raster rastA , raster rastB );
boolean ST_Intersects( raster rast , integer nband , geometry geommin );
boolean ST_Intersects( raster rast , geometry geommin , integer nband=NULL );
boolean ST_Intersects( geometry geommin , raster rast , integer nband=NULL );
```

Beschreibung

Return true if raster rastA spatially intersects raster rastB. If the band number is not provided (or set to NULL), only the convex hull of the raster is considered in the test. If the band number is provided, only those pixels with value (not NODATA) are considered in the test.

**Note**

Diese Funktion verwendet die für Raster verfügbaren Indizes.

Erweiterung: 2.0.0 Unterstützung für Raster/Raster Verschneidungen eingeführt.

**Warning**

Änderung: 2.1.0 Die Verhaltensweise der Varianten von **ST_Intersects**(raster, geometry) wurde geändert um mit dem von **ST_Intersects**(geometry, raster) übereinzustimmen.

Beispiele

```
-- unterschiedliche Bänder desselben Rasters
SELECT ST_Intersects(rast, 2, rast, 3) FROM dummy_rast WHERE rid = 2;

st_intersects
-----
t
```

Siehe auch[ST_Intersection](#), [ST_Disjoint](#)**9.13.7 ST_Overlaps**

ST_Overlaps — Gibt TRUE zurück, wenn sich die Raster "rastA" und "rastB" schneiden, aber ein Raster den anderen nicht zur Gänze enthält.

Synopsis

boolean **ST_Overlaps**(raster rastA , integer nbandA , raster rastB , integer nbandB);
boolean **ST_Overlaps**(raster rastA , raster rastB);

Beschreibung

Return true if raster rastA spatially overlaps raster rastB. This means that rastA and rastB intersect but one does not completely contain the other. If the band number is not provided (or set to NULL), only the convex hull of the raster is considered in the test. If the band number is provided, only those pixels with value (not NODATA) are considered in the test.



Note

Diese Funktion verwendet die für Raster verfügbaren Indizes.



Note

Um die räumliche Beziehung zwischen einem Raster und einem geometrischen Datentyp zu überprüfen, verwenden Sie bitte ST_Polygon für den Raster, z.B. ST_Overlaps(ST_Polygon(raster), geometry).

Verfügbarkeit: 2.1.0

Beispiele

```
-- verschiedene Bänder des gleichen Rasters vergleichen
SELECT ST_Overlaps(rast, 1, rast, 2) FROM dummy_rast WHERE rid = 2;

st_overlaps
-----
f
```

Siehe auch

[ST_Intersects](#)

9.13.8 ST_Touches

ST_Touches — Gibt TRUE zurück, wenn rastA und rastB zumindest einen Punkt gemeinsam haben sich aber nicht überschneiden.

Synopsis

boolean **ST_Touches**(raster rastA , integer nbandA , raster rastB , integer nbandB);
boolean **ST_Touches**(raster rastA , raster rastB);

Beschreibung

Return true if raster `rastA` spatially touches raster `rastB`. This means that `rastA` and `rastB` have at least one point in common but their interiors do not intersect. If the band number is not provided (or set to NULL), only the convex hull of the raster is considered in the test. If the band number is provided, only those pixels with value (not NODATA) are considered in the test.



Note

Diese Funktion verwendet die für Raster verfügbaren Indizes.



Note

Um die räumliche Beziehung zwischen einem Raster und einem geometrischen Datentyp zu überprüfen, verwenden Sie bitte `ST_Polygon` für den Raster, z.B. `ST_Touches(ST_Polygon(raster), geometry)`.

Verfügbarkeit: 2.1.0

Beispiele

```
SELECT r1.rid, r2.rid, ST_Touches(r1.rast, 1, r2.rast, 1) FROM dummy_rast r1 CROSS JOIN ↵
    dummy_rast r2 WHERE r1.rid = 2;
```

rid	rid	st_touches
2	1	f
2	2	f

Siehe auch

[ST_Intersects](#)

9.13.9 ST_SameAlignment

`ST_SameAlignment` — Returns true if rasters have same skew, scale, spatial ref, and offset (pixels can be put on same grid without cutting into pixels) and false if they don't with notice detailing issue.

Synopsis

```
boolean ST_SameAlignment( raster rastA , raster rastB );
boolean ST_SameAlignment( double precision ulx1 , double precision uly1 , double precision scalex1 , double precision scaley1
, double precision skewx1 , double precision skewy1 , double precision ulx2 , double precision uly2 , double precision scalex2 ,
double precision scaley2 , double precision skewx2 , double precision skewy2 );
boolean ST_SameAlignment( raster set rastfield );
```

Beschreibung

Non-Aggregate version (Variants 1 and 2): Returns true if the two rasters (either provided directly or made using the values for upperleft, scale, skew and srid) have the same scale, skew, srid and at least one of any of the four corners of any pixel of one raster falls on any corner of the grid of the other raster. Returns false if they don't and a NOTICE detailing the alignment issue.

Aggregate version (Variant 3): From a set of rasters, returns true if all rasters in the set are aligned. The `ST_SameAlignment()` function is an "aggregate" function in the terminology of PostgreSQL. That means that it operates on rows of data, in the same way the `SUM()` and `AVG()` functions do.

Verfügbarkeit: 2.0.0

Erweiterung: 2.1.0 die Variante mit der Aggregatfunktion hinzugefügt

Beispiele: Raster

```
SELECT ST_SameAlignment (
    ST_MakeEmptyRaster(1, 1, 0, 0, 1, 1, 0, 0),
    ST_MakeEmptyRaster(1, 1, 0, 0, 1, 1, 0, 0)
) as sm;
```

```
sm
----
t
```

```
SELECT ST_SameAlignment(A.rast,b.rast)
FROM dummy_rast AS A CROSS JOIN dummy_rast AS B;
```

```
NOTICE: The two rasters provided have different SRIDs
NOTICE: The two rasters provided have different SRIDs
st_samealignment
-----
t
f
f
f
```

Siehe auch

Section 5.1, [ST_NotSameAlignmentReason](#), [ST_MakeEmptyRaster](#)

9.13.10 ST_NotSameAlignmentReason

`ST_NotSameAlignmentReason` — Gibt eine Meldung aus, die angibt ob die Raster untereinander ausgerichtet sind oder nicht und warum wenn nicht.

Synopsis

text `ST_NotSameAlignmentReason`(raster rastA, raster rastB);

Beschreibung

Gibt eine Meldung aus, die angibt ob die Raster untereinander ausgerichtet sind oder nicht und warum wenn nicht.



Note

Falls es mehrere Gründe gibt, warum die Raster nicht untereinander ausgerichtet sind, so wird nur der erste Grund (die erste fehlgeschlagene Überprüfung) ausgegeben.

Verfügbarkeit: 2.1.0

Beispiele

```
SELECT
    ST_SameAlignment (
        ST_MakeEmptyRaster(1, 1, 0, 0, 1, 1, 0, 0),
        ST_MakeEmptyRaster(1, 1, 0, 0, 1.1, 1.1, 0, 0)
    ),
    ST_NotSameAlignmentReason(
        ST_MakeEmptyRaster(1, 1, 0, 0, 1, 1, 0, 0),
        ST_MakeEmptyRaster(1, 1, 0, 0, 1.1, 1.1, 0, 0)
    )
;

st_samealignment | st_otsamealignmentreason
-----+-----
f                | The rasters have different scales on the X axis
(1 row)
```

Siehe auch

Section [5.1](#), [ST_SameAlignment](#)

9.13.11 ST_Within

ST_Within — Gibt TRUE zurück, wenn kein Punkt des Rasters "rastA" außerhalb des Rasters "rastB" liegt und zumindest ein Punkt im Inneren von "rastA" auch im Inneren von "rastB" liegt.

Synopsis

```
boolean ST_Within( raster rastA , integer nbandA , raster rastB , integer nbandB );
boolean ST_Within( raster rastA , raster rastB );
```

Beschreibung

Raster "rastA" liegt dann und nur dann "within"/innerhalb von "rastB", wenn sich kein Punkt von "rastA" im Äußeren des Rasters "rastB" befindet und zumindest ein Punkt im Inneren von "rastA" auch im Inneren von "rastB" liegt. Wenn die Bandnummer nicht angegeben ist (oder auf NULL gesetzt), dann wird nur die konvexe Hülle des Rasters zur Überprüfung herangezogen. Wenn die Bandnummer angegeben ist, werden nur die Pixel mit einem Wert (nicht NODATA) bei der Überprüfung herangezogen.



Note

This operand will make use of any indexes that may be available on the rasters.



Note

Um die räumliche Beziehung zwischen einem Raster und einer Geometrie zu überprüfen, verwenden Sie bitte `ST_Polygon` für den Raster, z.B. `ST_Within(ST_Polygon(raster), geometry)` or `ST_Within(geometry, ST_Polygon(raster))`.



Note

`ST_Within()` ist die Umkehrfunktion von `ST_Contains()`. Es gilt daher: `ST_Within(rastA, rastB)` impliziert `ST_Contains(rastB, rastA)`.

Verfügbarkeit: 2.1.0

Beispiele

```
SELECT r1.rid, r2.rid, ST_Within(r1.rast, 1, r2.rast, 1) FROM dummy_rast r1 CROSS JOIN ↔
    dummy_rast r2 WHERE r1.rid = 2;
```

rid	rid	st_within
2	1	f
2	2	t

Siehe auch

[ST_Intersects](#), [ST_Contains](#), [ST_DWithin](#), [ST_DFullyWithin](#)

9.13.12 ST_DWithin

ST_DWithin — Gibt TRUE zurück, wenn die Raster "rastA" und "rastB" innerhalb der angegebenen Entfernung voneinander liegen.

Synopsis

```
boolean ST_DWithin( raster rastA , integer nbandA , raster rastB , integer nbandB , double precision distance_of_srid );
boolean ST_DWithin( raster rastA , raster rastB , double precision distance_of_srid );
```

Beschreibung

Gibt TRUE zurück, wenn die Raster "rastA" und "rastB" innerhalb der angegebenen Distanz zueinander liegen. Wenn die Bandnummer nicht angegeben ist (oder auf NULL gesetzt), dann wird nur die konvexe Hülle des Rasters zur Überprüfung herangezogen. Wenn die Bandnummer angegeben ist, werden nur die Pixel mit einem Wert (nicht NODATA) bei der Überprüfung herangezogen.

Die Distanz wird in den Einheiten des Koordinatenreferenzsystems des Rasters angegeben. Damit diese Funktion Sinn hat, müssen die Quellraster dieselbe Projektion und die gleiche SRID haben.



Note

This operand will make use of any indexes that may be available on the rasters.



Note

Um die räumliche Beziehung zwischen einem Raster und einer Geometrie zu untersuchen, wenden Sie bitte **ST_Polygon** auf den Raster an, z.B.: **ST_DWithin(ST_Polygon(raster), geometry)**.

Verfügbarkeit: 2.1.0

Beispiele

```
SELECT r1.rid, r2.rid, ST_DWithin(r1.rast, 1, r2.rast, 1, 3.14) FROM dummy_rast r1 CROSS
JOIN dummy_rast r2 WHERE r1.rid = 2;
```

rid	rid	st_dwithin
2	1	f
2	2	t

Siehe auch

[ST_Within](#), [ST_DFullyWithin](#)

9.13.13 ST_DFullyWithin

ST_DFullyWithin — Gibt TRUE zurück, wenn die Raster "rastA" und "rastB" zur Gänze innerhalb der angegebenen Distanz zueinander liegen.

Synopsis

boolean **ST_DFullyWithin**(raster rastA , integer nbandA , raster rastB , integer nbandB , double precision distance_of_srid);
 boolean **ST_DFullyWithin**(raster rastA , raster rastB , double precision distance_of_srid);

Beschreibung

Gibt TRUE zurück, wenn die Raster "rastA" und "rastB" zur Gänze innerhalb der angegebenen Distanz zueinander liegen. Wenn die Bandnummer nicht angegeben ist (oder auf NULL gesetzt), dann wird nur die konvexe Hülle des Rasters zur Überprüfung herangezogen. Wenn die Bandnummer angegeben ist, werden nur die Pixel mit einem Wert (nicht NODATA) bei der Überprüfung herangezogen.

Die Distanz wird in den Einheiten des Koordinatenreferenzsystems des Rasters angegeben. Damit diese Funktion Sinn hat, müssen die Quellraster dieselbe Projektion und die gleiche SRID haben.



Note

This operand will make use of any indexes that may be available on the rasters.



Note

Um die räumliche Relation zwischen einem Raster und einer Geometrie zu untersuchen, wenden Sie bitte **ST_Polygon** auf den Raster an, z.B.: **ST_DFullyWithin(ST_Polygon(raster), geometry)**.

Verfügbarkeit: 2.1.0

Beispiele

```
SELECT r1.rid, r2.rid, ST_DFullyWithin(r1.rast, 1, r2.rast, 1, 3.14) FROM dummy_rast r1
CROSS JOIN dummy_rast r2 WHERE r1.rid = 2;
```

rid	rid	st_dfullywithin
2	1	f
2	2	t

Siehe auch

[ST_Within](#), [ST_DWithin](#)

Chapter 10

Häufige Fragen zu PostGIS Raster

1. *Ich erhalte die Fehlermeldung `ERROR: RASTER_fromGDALRaster: Could not open bytea with GDAL. Check that the bytea is of a GDAL supported format. wenn ich ST_FromGDALRaster verwende, oder ERROR: rt_raster_to_gdal: Could not load the output GDAL driver wenn Ich ST_AsPNG oder andere Rastereingabefunktionen verwende.`*

As of PostGIS 2.1.3 and 2.0.5, a security change was made to by default disable all GDAL drivers and out of db rasters. The release notes are at [PostGIS 2.0.6, 2.1.3 security release](#). In order to reenale specific drivers or all drivers and reenale out of database support, refer to Section [2.1](#).

2. *Wo kann ich detaillierte Information über das Raster-Projekt von PostGIS finden?*

Siehe [PostGIS Raster Homepage](#).

3. *Gibt es irgendwelche Bücher oder Übungen, die mir erklären wie Ich mit dieser wunderbare Erfindung loslegen kann?*

There is a full length beginner tutorial [Intersecting vector buffers with large raster coverage using PostGIS Raster](#). Jorge has a series of blog articles on PostGIS Raster that demonstrate how to load raster data as well as cross compare to same tasks in Oracle GeoRaster. Check out [Jorge's PostGIS Raster / Oracle GeoRaster Series](#). There is a whole chapter (more than 35 pages of content) dedicated to PostGIS Raster with free code and data downloads at [PostGIS in Action - Raster chapter](#). You can [buy PostGIS in Action](#) now from Manning in hard-copy (significant discounts for bulk purchases) or just the E-book format. You can also buy from Amazon and various other book distributors. All hard-copy books come with a free coupon to download the E-book version. Here is a review from a PostGIS Raster user [PostGIS raster applied to land classification urban forestry](#)

4. *Wie installiere Ich die Rasterunterstützung in meiner PostGIS Datenbank?*

The easiest is to download binaries for PostGIS and Raster which are currently available for windows and latest versions of Mac OSX. First you need a working PostGIS 2.0.0 or above and be running PostgreSQL 8.4, 9.0, or 9.1. Note in PostGIS 2.0 PostGIS Raster is fully integrated, so it will be compiled when you compile PostGIS. Instructions for installing and running under windows are available at [How to Install and Configure PostGIS raster on windows](#). If you are on windows, you can compile yourself, or use the [pre-compiled PostGIS Raster windows binaries](#). If you are on Mac OSX Leopard or Snow Leopard, there are binaries available at [Kyng Chaos Mac OSX PostgreSQL/GIS binaries](#). Then to enable raster support in your database, run the `rtpostgis.sql` file in your database. To upgrade an existing install use `rtpostgis_upgrade_minor..sql` instead of `rtpostgis.sql`. For other platforms, you generally need to compile yourself. Dependencies are PostGIS and GDAL. For more details about compiling from source, please refer to [Installing PostGIS Raster from source \(in prior versions of PostGIS\)](#)

5. *Ich bekomme die Fehlermeldung: 'could not load library "C:/Program Files/PostgreSQL/8.4/lib/rtpostgis.dll": The specified module could not be found'. Oder 'could not load library on Linux when trying to run rtpostgis.sql'*

`rtpostgis.so/dll` is built with dependency on `libgdal.dll/so`. Make sure for Windows you have `libgdal-1.dll` in the bin folder of your PostgreSQL install. For Linux `libgdal` has to be in your path or bin folder. You may also run into different errors if you don't have PostGIS installed in your database. Make sure to install PostGIS first in your database before trying to install the raster support.

6. Wie kann ich Rasterdaten in PostGIS laden?

The latest version of PostGIS comes packaged with a `raster2pgsql` raster loader executable capable of loading many kinds of rasters and also generating lower resolution overviews without any additional software. Please refer to Section 5.1.1 for more details. Pre-2.0 versions came with a `raster2pgsql.py` that required python with numpy and GDAL. This is no longer needed.

7. Welche Rasterformate kann Ich in meine Datenbank laden?

Any that your GDAL library supports. GDAL supported formats are documented [GDAL File Formats](#). Your particular GDAL install may not support all formats. To verify the ones supported by your particular GDAL install, you can use

```
raster2pgsql -G
```

8. Kann Ich meine PostGIS Rasterdaten in ein anderes Format exportieren?

Yes GDAL 1.7+ enthält einen PostGIS Raster-Treiber; dieser ist allerdings nur dann vorhanden, wenn GDAL mit PostgreSQL-Unterstützung kompiliert wurde. The driver currently doesn't support irregularly blocked rasters, although you can store irregularly blocked rasters in PostGIS raster data type. If you are compiling from source, you need to include in your configure

```
--with-pg=path/to/pg_config
```

to enable the driver. Refer to [GDAL Build Hints](#) for tips on building GDAL against in various OS platforms. If your version of GDAL is compiled with the PostGIS Raster driver you should see PostGIS Raster in list when you do

```
gdalinfo --formats
```

Um eine Zusammenfassung Ihrer Raster über GDAL zu erhalten, benutzen Sie bitte `gdalinfo`:

```
gdalinfo "PG:host=localhost port=5432 dbname='mygisdb' user='postgres' password=' ←  
whatever' schema='someschema' table=sometable"
```

To export data to other raster formats, use `gdal_translate` the below will export all data from a table to a PNG file at 10% size. Depending on your pixel band types, some translations may not work if the export format does not support that Pixel type. For example floating point band types and 32 bit unsigned ints will not translate easily to JPG or some others. Ein Beispiel für eine einfache Übersetzung:

```
gdal_translate -of PNG -outsize 10% 10% "PG:host=localhost port=5432 dbname='mygisdb' ←  
user='postgres' password='whatever' schema='someschema' table=sometable" C:\ ←  
somefile.png
```

You can also use SQL where clauses in your export using the `where=...` in your driver connection string. Below are some using a where clause

```
gdal_translate -of PNG -outsize 10% 10% "PG:host=localhost port=5432 dbname='mygisdb' ←  
user='postgres' password='whatever' schema='someschema' table=sometable where=' ←  
filename='abcd.sid'" " C:\somefile.png
```

```
gdal_translate -of PNG -outsize 10% 10% "PG:host=localhost port=5432 dbname='mygisdb' ←  
user='postgres' password='whatever' schema='someschema' table=sometable where=' ←  
ST_Intersects(rast, ST_SetSRID(ST_Point(-71.032,42.3793),4326) )" " C:\ ←  
intersectregion.png
```

To see more examples and syntax refer to [Reading Raster Data of PostGIS Raster section](#)

9. Gibt es Binärdateien von GDAL, die für die PostGIS Rasterunterstützung bereits kompiliert sind?

Ja. Siehe [GDAL Binaries](#). Jene, die mit PostgreSQL-Unterstützung kompiliert sind, sollten PostGIS Raster implementiert haben. PostGIS Raster is undergoing many changes. If you want to get the latest nightly build for Windows -- then check out the Tamas Szekeres nightly builds built with Visual Studio which contain GDAL trunk, Python Bindings and MapServer executables and PostGIS Raster driver built-in. Just click the SDK bat and run your commands from there. <http://www.gisinternals.com>. Also available are VS project files. [Die neueste, stabile Version von FWTools für Windows ist mit Rasterunterstützung kompiliert.](#)

10. Welche Werkzeuge kann Ich benutzen, um Rasterdaten, die sich in PostGIS befinden, anzusehen?

You can use MapServer compiled with GDAL 1.7+ and PostGIS Raster driver support to view Raster data. QuantumGIS (QGIS) now supports viewing of PostGIS Raster if you have PostGIS raster driver installed. In theory any tool that renders data using GDAL can support PostGIS raster data or support it with fairly minimal effort. Again for Windows, Tamas' binaries <http://www.gisinternals.com> are a good choice if you don't want the hassle of having to setup to compile your own.

11. Wie kann Ich einen PostGIS-Raster zu meiner MapServer-Karte hinzufügen?

First you need GDAL 1.7 or higher compiled with PostGIS raster support. GDAL 1.8 or above is preferred since many issues have been fixed in 1.8 and more PostGIS raster issues fixed in trunk version. You can much like you can with any other raster. Refer to [MapServer Raster processing options](#) for list of various processing functions you can use with MapServer raster layers. What makes PostGIS raster data particularly interesting, is that since each tile can have various standard database columns, you can segment it in your data source. Unterhalb ein Beispiel, wie Sie einen PostGIS-Rasterlayer in MapServer anlegen können.



Note

Der Parameter, mode=2, wird für geteilte Raster benötigt und wurde in PostGIS 2.0.0

```
-- Einen Raster mit den Standardeinstellungen darstellen
LAYER
    NAME coolwktraster
    TYPE raster
    STATUS ON
    DATA "PG:host=localhost port=5432 dbname='somedb' user='someuser' password=' ←
        whatever'
        schema='someschema' table='coolttable' mode='2'"
    PROCESSING "NODATA=0"
    PROCESSING "SCALE=AUTO"
    #... other standard raster processing functions here
    #... classes are optional but useful for 1 band data
    CLASS
        NAME "boring"
        EXPRESSION ([pixel] < 20)
        COLOR 250 250 250
    END
    CLASS
        NAME "mildly interesting"
        EXPRESSION ([pixel] > 20 AND [pixel] < 1000)
        COLOR 255 0 0
    END
    CLASS
        NAME "very interesting"
        EXPRESSION ([pixel] >= 1000)
        COLOR 0 255 0
    END
END
END
```

```
-- Einen Raster mit den Standardeinstellungen und einer WHERE-Klausel darstellen
LAYER
    NAME soil_survey2009
    TYPE raster
    STATUS ON
    DATA "PG:host=localhost port=5432 dbname='somedb' user='someuser' password=' ←
        whatever'
        schema='someschema' table='coolttable' where='survey_year=2009' mode ←
        ='2'"
    PROCESSING "NODATA=0"
```

```
#... other standard raster processing functions here
#... classes are optional but useful for 1 band data
END
```

12. *Welche Funktionen kann Ich zurzeit mit meinen Rasterdaten nutzen?*

Siehe Chapter 9. Es gibt noch mehr Funktionen, doch diese befinden sich noch in der Aufbauphase. Refer to the [PostGIS Raster roadmap page](#) for details of what you can expect in the future.

13. *Ich erhalte die Fehlermeldung "ERROR: function st_intersects(raster, unknown) is not unique or st_union(geometry,text) is not unique." Wie kann ich das beheben?*

The function is not unique error happens if one of your arguments is a textual representation of a geometry instead of a geometry. In these cases, PostgreSQL marks the textual representation as an unknown type, which means it can fall into the `st_intersects(raster, geometry)` or `st_intersects(raster,raster)` thus resulting in a non-unique case since both functions can in theory support your request. To prevent this, you need to cast the textual representation of the geometry to a geometry. Wenn Ihr Code zum Beispiel folgendermaßen aussieht:

```
SELECT rast
FROM my_raster
WHERE ST_Intersects(rast, 'SRID=4326;POINT(-10 10)');
```

Cast the textual geometry representation to a geometry by changing your code to this:

```
SELECT rast
FROM my_raster
WHERE ST_Intersects(rast, 'SRID=4326;POINT(-10 10)::geometry');
```

14. *Wie unterscheidet sich PostGIS Raster von Oracle GeoRaster (SDO_GEORASTER) und SDO_RASTER-Typen?*

Für eine ausführlichere Erörterung dieses Themas siehe Jorge Arévalo [Oracle GeoRaster and PostGIS Raster: First impressions](#). Der wesentliche Vorteil von "one-georeference-by-raster" gegenüber "one-georeference-by-layer" ist: * coverages to be not necessarily rectangular (which is often the case of raster coverage covering large extents. See the possible raster arrangements in the documentation) * Raster können überlappen (dies ermöglicht die verlustfreie Konvertierung von Vektor nach Raster) These arrangements are possible in Oracle as well, but they imply the storage of multiple SDO_GEORASTER objects linked to as many SDO_RASTER tables. A complex coverage can lead to hundreds of tables in the database. With PostGIS Raster you can store a similar raster arrangement into a unique table. It's a bit like if PostGIS would force you to store only full rectangular vector coverage without gaps or overlaps (a perfect rectangular topological layer). This is very practical in some applications but practice has shown that it is not realistic or desirable for most geographical coverages. Vector structures needs the flexibility to store discontinuous and non-rectangular coverages. We think it is a big advantage that raster structure should benefit as well.

15. *raster2pgsql load of large file fails with String of N bytes is too long for encoding conversion?*

`raster2pgsql` doesn't make any connections to your database when generating the file to load. If your database has set an explicit client encoding different from your database encoding, then when loading large raster files (above 30 MB in size), you may run into a `bytes is too long for encoding conversion`. This generally happens if for example you have your database in UTF8, but to support windows apps, you have the client encoding set to WIN1252. To work around this make sure the client encoding is the same as your database encoding during load. You can do this by explicitly setting the encoding in your load script. Example, if you are on windows:

```
set PGCLIENTENCODING=UTF8
```

Falls Sie sich auf Unix/Linux befinden

```
export PGCLIENTENCODING=UTF8
```

Mörderische Einzelheiten zu diesem Thema finden sich unter <http://trac.osgeo.org/postgis/ticket/2209>

Chapter 11

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 über das was Topologie ist, wie sie verwendet wird, und stellt auch verschiedene FOSS4G Werkzeuge zur Unterstützung vor.

Ein Beispiel für eine topologische Geodatenbank ist die [US Census Topologically Integrated Geographic Encoding and Referencing System \(TIGER\)](#) Datenbank. Zum Experimentieren mit der PostGIS Topologie stehen unter [Topology_Load_Tiger](#) Daten zur Verfügung.

Das PostGIS Modul "Topologie" gab es auch schon in früheren Versionen von PostGIS, es war aber nie Teil der offiziellen PostGIS Dokumentation. In PostGIS 2.0.0 fand eine umfangreiche Überarbeitung statt, um überholte Funktionen zu entfernen, bekannte Probleme mit der Bedienbarkeit zu bereinigen, bessere Dokumentation der Funktionalität, Einführung neuer Funktionen, und eine bessere Übereinstimmung mit den SQL-MM Normen zu erreichen.

Genauere Angaben zu diesem Projekt finden sich unter [PostGIS Topology Wiki](#)

Alle Funktionen und Tabellen, die zu diesem Modul gehören, sind im Schema mit der Bezeichnung `topology` installiert.

Funktionen die im SQL/MM Standard definiert sind erhalten das Präfix `ST_`, PostGIS eigene Funktionen erhalten kein Präfix.

Ab PostGIS 2.0 wird die Topologie Unterstützung standardmäßig mitkompiliert und kann bei der Konfiguration mittels der Konfigurationsoption `--without-topology`, wie in [Chapter 2](#) beschrieben, deaktiviert werden.

11.1 Topologische Datentypen

11.1.1 `getfaceedges_returntype`

`getfaceedges_returntype` — Ein zusammengesetzter Typ, der aus einer Sequenzzahl und einer Kantenzahl besteht. Dies ist der von `ST_GetFaceEdges` zurückgegebene Typ.

Beschreibung

Ein zusammengesetzter Datentyp, der aus einer Sequenznummer und einer Kantenummer besteht. Dies ist der von der Funktion `ST_GetFaceEdges` zurückgegebene Datentyp.

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.
-

11.1.2 TopoGeometry

TopoGeometry — Ein zusammengesetzter Typ, der eine topologisch festgelegte Geometrie darstellt.

Beschreibung

Ein zusammengesetzter Datentyp, der auf eine topologische Geometrie in einem bestimmten topologischen Layer verweist und einen spezifischen Datentyp und eine eindeutige ID hat. Folgende Bestandteile bilden die Elemente einer TopoGeometry: `topology_id`, `layer_id`, `id` Ganzzahl, `type` Ganzzahl.

1. `topology_id` ist eine Ganzzahl: Sie verweist auf eine Topologie, die in der Tabelle `topology.topology` definiert ist. In dieser Tabelle sind das Schema, das die Topologie enthält, und die SRID verzeichnet.
2. `layer_id` ist eine Ganzzahl: Die `layer_id` in der Tabelle `topology.layer` zu der die TopoGeometry gehört. Die Kombination aus `topology_id` und `layer_id` liefert eine eindeutige Referenz in der Tabelle `topology.layer`.
3. `id` ist eine Ganzzahl: Die `id` ist eine automatisch erzeugte Sequenznummer, welche die TopoGeometry in dem jeweiligen topologischen Layer eindeutig ausweist.
4. `type` ist eine Ganzzahl zwischen 1 und 4, welche den geometrischen Datentyp festlegt: 1:[Multi]Point, 2:[Multi]Line, 3:[Multi]Polygon, 4:GeometryCollection

Verhaltensweise bei der Typumwandlung

In diesem Abschnitt sind die für diesen Datentyp erlaubten impliziten und expliziten Typumwandlungen beschrieben.

Typumwandlung nach	Verhaltensweise
Geometrie	implizit

Siehe auch

[CreateTopoGeom](#)

11.1.3 validate_topology_returntype

`validate_topology_returntype` — Ein zusammengesetzter Datentyp, der aus einer Fehlermeldung und `id1` und `id2` besteht. `id1` und `id2` deuten auf die Stelle hin, an der der Fehler auftrat. Dies ist der von `ValidateTopology` zurückgegebene Datentyp.

Beschreibung

Ein zusammengesetzter Datentyp, der aus einer Fehlermeldung und zwei Ganzzahlen besteht. Die Funktion `ValidateTopology` gibt eine Menge dieser Datentypen zurück, um bei der Validierung gefundene Fehler zu beschreiben. Unter `id1` und `id2` sind die ids der topologischen Objekte verzeichnet, die an dem Fehler beteiligt sind.

1. `error` ist varchar: Gibt die Art des Fehlers an.
Aktuell existieren folgende Fehlerbeschreibungen: zusammenfallende Knoten/coincident nodes, Kante ist nicht simple/edge not simple, Kanten- und Endknotengeometrie stimmen nicht überein/edge end node geometry mis-match, Kanten- und Anfangsknotengeometrie stimmen nicht überein/edge start node geometry mismatch, Masche überlappt Masche/face overlaps face, Masche innerhalb einer Masche/face within face.
2. `id1` ist eine ganze Zahl: Gibt den Identifikator einer Kante / Masche / Knoten in der Fehlermeldung an.
3. `id2` ist eine ganze Zahl: Wenn 2 Objekte in den Fehler involviert sind verweist diese Zahl auf die zweite Kante / oder Knoten.

Siehe auch[ValidateTopology](#)

11.2 Topologische Domänen

11.2.1 TopoElement

TopoElement — Ein Feld mit 2 Ganzzahlen, welches in der Regel für die Auffindung einer Komponente einer TopoGeometry dient.

Beschreibung

Ein Feld mit 2 Ganzzahlen, welches einen Bestandteil einer einfachen oder hierarchischen **TopoGeometry** abbildet.

Im Falle einer einfachen TopoGeometry ist das erste Element des Feldes der Identifikator einer topologischen Elementarstruktur und das zweite Element der Typ (1:Knoten, 2:Kante, 3:Masche). Im Falle einer hierarchischen TopoGeometry ist das erste Element des Feldes der Identifikator der Kind-TopoGeometry und das zweite Element der Identifikator des Layers.

**Note**

For any given hierarchical TopoGeometry all child TopoGeometry elements will come from the same child layer, as specified in the topology.layer record for the layer of the TopoGeometry being defined.

Beispiele

```
SELECT te[1] AS id, te[2] AS type FROM
( SELECT ARRAY[1,2]::topology.topoelement AS te ) f;
 id | type
-----+-----
  1 |    2
```

```
SELECT ARRAY[1,2]::topology.topoelement;
 te
-----
{1,2}
```

```
--Beispiel was passiert, wenn versucht wird ein aus 3 Elementen bestehendes Feld in ein ↔
  TopoElement umzuwandeln
-- Anmerkung: TopoElement muss ein Feld mit 2 Elementen sein und somit scheitert die ↔
  Dimensionsprüfung
SELECT ARRAY[1,2,3]::topology.topoelement;
ERROR:  value for domain topology.topoelement violates check constraint "dimensions"
```

Siehe auch

[GetTopoGeomElements](#), [TopoElementArray](#), [TopoGeometry](#), [TopoGeom_addElement](#), [TopoGeom_remElement](#)

11.2.2 TopoElementArray

TopoElementArray — Ein Feld mit TopoElement Objekten

Beschreibung

Ein Feld mit 1 oder mehreren TopoElement Objekten; wird hauptsächlich verwendet, um Bestandteile einer TopoGeometry heruzureichen.

Beispiele

```
SELECT '{{1,2},{4,3}}'::topology.topoelementarray As tea;
      tea
-----
{{1,2},{4,3}}
```

```
-- langatmige Version --
SELECT ARRAY[ARRAY[1,2], ARRAY[4,3]]::topology.topoelementarray As tea;

      tea
-----
{{1,2},{4,3}}
```

```
--Verwendung der Feld-Aggregatsfunktion von Topology --
SELECT topology.TopoElementArray_Agg(ARRAY[e,t]) As tea
  FROM generate_series(1,4) As e CROSS JOIN generate_series(1,3) As t;
      tea
-----
{{1,1},{1,2},{1,3},{2,1},{2,2},{2,3},{3,1},{3,2},{3,3},{4,1},{4,2},{4,3}}
```

```
SELECT '{{1,2,4},{3,4,5}}'::topology.topoelementarray As tea;
ERROR:  value for domain topology.topoelementarray violates check constraint "dimensions"
```

Siehe auch

[TopoElement](#), [GetTopoGeomElementArray](#), [TopoElementArray_Agg](#)

11.3 Verwaltung von Topologie und TopoGeometry

11.3.1 AddTopoGeometryColumn

AddTopoGeometryColumn — Fügt ein TopoGeometry Attribut an eine bestehende Tabelle an, registriert dieses neue Attribut als einen Layer in topology.layer und gibt die neue layer_id zurück.

Synopsis

```
integer AddTopoGeometryColumn(varchar topology_name, varchar schema_name, varchar table_name, varchar column_name,
varchar feature_type);
integer AddTopoGeometryColumn(varchar topology_name, varchar schema_name, varchar table_name, varchar column_name,
varchar feature_type, integer child_layer);
```

Beschreibung

Jedes TopoGeometry Objekt gehört zu einem bestimmten Layer einer bestimmten Topologie. Bevor Sie ein TopoGeometry Objekt erzeugen, müssen Sie dessen topologischen Layer erzeugen. Ein topologischer Layer ist eine Assoziation einer Featuretabelle mit der Topologie. Er enthält auch Angaben zum Typ und zur Hierarchie. Man erzeugt einen Layer mittels der Funktion AddTopoGeometryColumn():

Diese Funktion fügt sowohl das angeforderte Attribut an die Tabelle als auch einen Datensatz mit der angegebenen Information in die Tabelle topology.layer.

Wenn Sie den [child_layer] nicht angeben (oder auf NULL setzen), dann enthält dieser Layer elementare TopoGeometries (aus topologischen Elementarstrukturen zusammengesetzt). Andernfalls enthält dieser Layer hierarchische TopoGeometries (aus den TopoGeometries des child_layer zusammengesetzt).

Sobald der Layer erstellt wurde (seine id von der Funktion AddTopoGeometryColumn zurückgegeben wurde), können Sie TopoGeometry Objekte in ihm erzeugen

Gültige feature_types sind: POINT, LINE, POLYGON, COLLECTION

Verfügbarkeit: 1.?

Beispiele

```
-- Beachten Sie bitte, dass wir für dieses Beispiel eine neue Tabelle im ma_topo Schema erzeugt haben ↵
-- Wir hätten sie auch in einem anderen Schema erstellen können -- in diesem Fall würden ↵
-- sich topology_name und schema_name unterscheiden
CREATE SCHEMA ma;
CREATE TABLE ma.parcels(gid serial, parcel_id varchar(20) PRIMARY KEY, address text);
SELECT topology.AddTopoGeometryColumn('ma_topo', 'ma', 'parcels', 'topo', 'POLYGON');
```

```
CREATE SCHEMA ri;
CREATE TABLE ri.roads(gid serial PRIMARY KEY, road_name text);
SELECT topology.AddTopoGeometryColumn('ri_topo', 'ri', 'roads', 'topo', 'LINE');
```

Siehe auch

[CreateTopology](#), [CreateTopoGeom](#)

11.3.2 DropTopology

DropTopology — Bitte mit Vorsicht verwenden: Löscht ein topologisches Schema und dessen Referenz in der Tabelle topology.topology, sowie die Referenzen zu den Tabellen in diesem Schema aus der Tabelle geometry_columns.

Synopsis

integer **DropTopology**(varchar topology_schema_name);

Beschreibung

Löscht ein topologisches Schema und dessen Referenz in der Tabelle topology.topology, sowie die Referenzen zu den Tabellen in diesem Schema aus der Tabelle geometry_columns. Diese Funktion sollte MIT VORSICHT BENUTZT werden, da damit unabsichtlich Daten zerstört werden können. Falls das Schema nicht existiert, werden nur die Referenzeinträge des bezeichneten Schemas gelöscht.

Verfügbarkeit: 1.?

Beispiele

Löscht das Schema "ma_topo" kaskadierend und entfernt alle Referenzen in topology.topology und in geometry_columns.

```
SELECT topology.DropTopology('ma_topo');
```


Siehe auch

11.3.3 DropTopoGeometryColumn

DropTopoGeometryColumn — Entfernt ein TopoGeometry-Attribut aus der Tabelle mit der Bezeichnung `table_name` im Schema `schema_name` und entfernt die Registrierung der Attribute aus der Tabelle "topology.layer".

Synopsis

text **DropTopoGeometryColumn**(varchar `schema_name`, varchar `table_name`, varchar `column_name`);

Beschreibung

Entfernt ein TopoGeometry-Attribut aus der Tabelle mit der Bezeichnung `table_name` im Schema `schema_name` und entfernt die Registrierung der Attribute aus der Tabelle "topology.layer". Gibt eine Zusammenfassung des Löschstatus aus. ANMERKUNG: vor dem Löschen werden alle Werte auf NULL gesetzt, um die Überprüfung der referenziellen Integrität zu umgehen.

Verfügbarkeit: 1.?

Beispiele

```
SELECT topology.DropTopoGeometryColumn('ma_topo', 'parcel_topo', 'topo');
```

Siehe auch

[AddTopoGeometryColumn](#)

11.3.4 Populate_Topology_Layer

Populate_Topology_Layer — Fügt fehlende Einträge zu der Tabelle `topology.layer` hinzu, indem Metadaten aus den topologischen Tabellen ausgelesen werden.

Synopsis

setof record **Populate_Topology_Layer**();

Beschreibung

Trägt fehlende Einträge in der Tabelle `topology.layer` ein, indem die topologischen Constraints und Tabellen inspiziert werden. Diese Funktion ist nach der Wiederherstellung von Schemata mit topologischen Daten sinnvoll, um Einträge im Topologie-Katalog zu reparieren.

Wirft eine Liste der erzeugten Einträge aus. Die zurückgegebenen Attribute sind `schema_name`, `table_name` und `feature_column`.

Verfügbarkeit: 2.3.0

Beispiele

```
SELECT CreateTopology('strk_topo');
CREATE SCHEMA strk;
CREATE TABLE strk.parcels(gid serial, parcel_id varchar(20) PRIMARY KEY, address text);
SELECT topology.AddTopoGeometryColumn('strk_topo', 'strk', 'parcels', 'topo', 'POLYGON');
-- diese Abfrage gibt keine Datensätze zurück, da bereits registriert wurde
SELECT *
  FROM topology.Populate_Topology_Layer();

-- Erneut aufbauen
TRUNCATE TABLE topology.layer;

SELECT *
  FROM topology.Populate_Topology_Layer();

SELECT topology_id, layer_id, schema_name As sn, table_name As tn, feature_column As fc
FROM topology.layer;
```

```
schema_name | table_name | feature_column
-----+-----+-----
strk        | parcels    | topo
(1 row)

topology_id | layer_id | sn  | tn    | fc
-----+-----+----+-----+-----
2          | 2        | strk | parcels | topo
(1 row)
```

Siehe auch

[AddTopoGeometryColumn](#)

11.3.5 TopologySummary

TopologySummary — Nimmt den Namen einer Topologie und liefert eine Zusammenfassung der Gesamtsummen der Typen und Objekte in der Topologie

Synopsis

```
text TopologySummary(varchar topology_schema_name);
```

Beschreibung

Nimmt den Namen einer Topologie und liefert eine Zusammenfassung der Gesamtsummen der Typen und Objekte in der Topologie.

Verfügbarkeit: 2.0.0

Beispiele

```
SELECT topology.topologysummary('city_data');
           topologysummary
-----
Topology city_data (329), SRID 4326, precision: 0
22 nodes, 24 edges, 10 faces, 29 topogeoms in 5 layers
Layer 1, type Polygonal (3), 9 topogeoms
  Deploy: features.land_parcels.feature
Layer 2, type Puntal (1), 8 topogeoms
  Deploy: features.traffic_signs.feature
Layer 3, type Lineal (2), 8 topogeoms
  Deploy: features.city_streets.feature
Layer 4, type Polygonal (3), 3 topogeoms
  Hierarchy level 1, child layer 1
    Deploy: features.big_parcels.feature
Layer 5, type Puntal (1), 1 topogeoms
  Hierarchy level 1, child layer 2
    Deploy: features.big_signs.feature
```

Siehe auch

[Topology_Load_Tiger](#)

11.3.6 ValidateTopology

ValidateTopology — Liefert eine Menge `validatetopology_returntype` Objekte, die Probleme mit der Topologie beschreiben

Synopsis

```
setof validatetopology_returntype ValidateTopology(varchar topology_schema_name);
```

Beschreibung

Liefert eine Menge `validatetopology_returntype` Objekte, die Probleme mit der Topologie beschreiben. Mögliche Fehler und wofür die zurückgegebenen ids stehen ist in der folgenden Liste dargestellt:

Error	id1	id2
edge crosses node	edge_id	node_id
invalid edge	edge_id	null
edge not simple	edge_id	null
edge crosses edge	edge_id	edge_id
edge start node geometry mis-match	edge_id	node_id
edge end node geometry mis-match	edge_id	node_id
face without edges	face_id	null
face has no rings	face_id	null
face overlaps face	face_id	face_id
face within face	inner face_id	outer face_id

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.

Beispiele

```
SELECT * FROM topology.ValidateTopology('ma_topo');
      error      | id1 | id2
-----+-----+-----
face without edges |    0 |
```

Siehe auch

[validatetopology_returntype](#), [Topology_Load_Tiger](#)

11.4 Topologie Konstruktoren

11.4.1 CreateTopology

CreateTopology — Erstellt ein neues topologisches Schema und registriert das neue Schema in der Tabelle topology.topology.

Synopsis

```
integer CreateTopology(varchar topology_schema_name);
integer CreateTopology(varchar topology_schema_name, integer srid);
integer CreateTopology(varchar topology_schema_name, integer srid, double precision prec);
integer CreateTopology(varchar topology_schema_name, integer srid, double precision prec, boolean hasz);
```

Beschreibung

Erzeugt ein neues Schema mit der Bezeichnung `topology_name`, das aus den Tabellen (`edge_data`, `face`, `node`, `relation`) besteht, und registriert diese neue Topologie in der Tabelle `topology.topology`. Gibt die id der Topologie in der Topologietabelle aus. Die SRID entspricht dem Identifikator des Koordinatenreferenzsystems in der Tabelle `spatial_ref_sys` für diese Topologie. Die Bezeichnung der Topologien muss eindeutig sein. Die Toleranz wird in den Einheiten des Koordinatenreferenzsystems gemessen. Wenn die Toleranz (`prec`) nicht angegeben ist, wird sie auf 0 gesetzt.

Dies ist ähnlich zu SQL/MM [ST_InitTopoGeo](#), aber ein bißchen funktioneller. Wenn `hasz` nicht angegeben wird, wird es standardmäßig auf FALSE gesetzt.

Verfügbarkeit: 1.?

Beispiele

Dieses Beispiel erzeugt ein neues Schema mit der Bezeichnung "ma_topo" und speichert Kanten, Maschen und Relationen in Massachusetts State Plane Meter (NAD83 / Massachusetts Mainland). Die Toleranz liegt bei 1/2 Meter, da das Koordinatenreferenzsystem auf Meter basiert.

```
SELECT topology.CreateTopology('ma_topo', 26986, 0.5);
```

Erzeugt die Rhode Island Topologie in State Plane ft (NAD83 / Rhode Island (ftUS))

```
SELECT topology.CreateTopology('ri_topo', 3438) As topoid;
topoid
-----
2
```

Siehe auch

Section [4.3.1](#), [ST_InitTopoGeo](#), [Topology_Load_Tiger](#)

11.4.2 CopyTopology

CopyTopology — Erzeugt eine Kopie einer topologischen Struktur (Knoten, Kanten, Maschen, Layer und TopoGeometries).

Synopsis

integer **CopyTopology**(varchar existing_topology_name, varchar new_name);

Beschreibung

Erzeugt eine neue Topologie mit der Bezeichnung `new_topology_name`. Die SRID und die Genauigkeit werden von `existing_topology_name` übernommen. Sämtliche Knoten, Kanten und Maschen, die Layer und die TopoGeometrien werden von der existierenden Topologie kopiert.

**Note**

Die neuen Zeilen in `topology.layer` enthalten künstlich erzeugte Werte für `schema_name`, `table_name` und `feature_column`. Der Grund dafür ist, dass die TopoGeometry nur als Definition existiert, aber zur Zeit auf Benutzerebene in keiner Tabelle verfügbar ist.

Verfügbarkeit: 2.0.0

Beispiele

Dieses Beispiel erstellt eine Kopie der Topologie "ma_topo"

```
SELECT topology.CopyTopology('ma_topo', 'ma_topo_bakup');
```

Siehe auch

Section [4.3.1](#), [CreateTopology](#)

11.4.3 ST_InitTopoGeo

ST_InitTopoGeo — Erstellt ein neues topologisches Schema und registriert das neue Schema in der Tabelle `topology.topology`.
Gibt eine Zusammenfassung des Prozessablaufs aus.

Synopsis

text **ST_InitTopoGeo**(varchar topology_schema_name);

Beschreibung

Dies ist das SQL-MM Pendant zu `CreateTopology`, allerdings ohne die Koordinatenreferenz und die Toleranzoptionen von `CreateTopology`; gibt einen textliche Erstellungsbericht anstelle der id der Topologie aus.

Verfügbarkeit: 1.?



This method implements the SQL/MM specification. SQL-MM 3 Topo-Geo and Topo-Net 3: Routine Details: X.3.17

Beispiele

```
SELECT topology.ST_InitTopoGeo('topo_schema_to_create') AS topocreation;
               astopocreation
-----
Topology-Geometry 'topo_schema_to_create' (id:7) created.
```

Siehe auch

[CreateTopology](#)

11.4.4 ST_CreateTopoGeo

ST_CreateTopoGeo — Fügt eine Sammlung von Geometrien an eine leere Topologie an und gibt eine Bestätigungsmeldung aus.

Synopsis

text **ST_CreateTopoGeo**(varchar atopology, geometry acollection);

Beschreibung

Fügt eine Sammelgeometrie einer leeren Topologie hinzu und gibt eine Bestätigungsmeldung aus.

Nützlich um eine leere Topologie zu befüllen.

Verfügbarkeit: 2.0



This method implements the SQL/MM specification. SQL-MM: Topo-Geo and Topo-Net 3: Routine Details -- X.3.18

Beispiele

```
-- Die Topologie bestücken --
SELECT topology.ST_CreateTopoGeo('ri_topo',
  ST_GeomFromText('MULTILINESTRING((384744 236928,384750 236923,384769 236911,384799 ↵
    236895,384811 236890,384833 236884,
    384844 236882,384866 236881,384879 236883,384954 236898,385087 236932,385117 236938,
    385167 236938,385203 236941,385224 236946,385233 236950,385241 236956,385254 236971,
    385260 236979,385268 236999,385273 237018,385273 237037,385271 237047,385267 237057,
    385225 237125,385210 237144,385192 237161,385167 237192,385162 237202,385159 237214,
    385159 237227,385162 237241,385166 237256,385196 237324,385209 237345,385234 237375,
    385237 237383,385238 237399,385236 237407,385227 237419,385213 237430,385193 237439,
    385174 237451,385170 237455,385169 237460,385171 237475,385181 237503,385190 237521,
    385200 237533,385206 237538,385213 237541,385221 237542,385235 237540,385242 237541,
    385249 237544,385260 237555,385270 237570,385289 237584,385292 237589,385291 ↵
    237596,385284 237630))',3438)
  );

      st_createtopogeo
-----
Topology ri_topo populated

-- Eine Tabelle und TopoGeometry erzeugen --
CREATE TABLE ri.roads(gid serial PRIMARY KEY, road_name text);

SELECT topology.AddTopoGeometryColumn('ri_topo', 'ri', 'roads', 'topo', 'LINE');
```

Siehe auch

[AddTopoGeometryColumn](#), [CreateTopology](#), [DropTopology](#)

11.4.5 TopoGeo_AddPoint

TopoGeo_AddPoint — Fügt einen Punkt, unter Berücksichtigung einer Toleranz, an eine bestehende Topologie an. Existierende Kanten werden eventuell aufgetrennt.

Synopsis

```
integer TopoGeo_AddPoint(varchar toponame, geometry apoint, float8 tolerance);
```

Beschreibung

Fügt einen Punkt an eine bestehende Topologie an und gibt dessen Identifikator zurück. Der vorgegebene Punkt wird von bestehenden Knoten oder Kanten, innerhalb einer gegebenen Toleranz, gefangen. Eine existierende Kante kann durch den gefangenen Punkt aufgetrennt werden.

Verfügbarkeit: 2.0.0

Siehe auch

[TopoGeo_AddLineString](#), [TopoGeo_AddPolygon](#), [AddNode](#), [CreateTopology](#)

11.4.6 TopoGeo_AddLineString

TopoGeo_AddLineString — Fügt einen Linienzug, unter Berücksichtigung einer Toleranz, an eine bestehende Topologie an. Existierende Kanten/Maschen werden eventuell aufgetrennt. Gibt den Identifikator der Kante aus

Synopsis

```
SETOF integer TopoGeo_AddLineString(varchar toponame, geometry aline, float8 tolerance);
```

Beschreibung

Fügt einen Linienzug an eine bestehende Topologie an und gibt die Identifikatoren der Kanten zurück, die diesen 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_AddPoint](#), [TopoGeo_AddPolygon](#), [AddEdge](#), [CreateTopology](#)

11.4.7 TopoGeo_AddPolygon

TopoGeo_AddPolygon — Fügt ein Polygon, unter Berücksichtigung einer Toleranz und eventuellem Auftrennen existierender Kanten/Maschen, zu einer bestehenden Topologie hinzu.

Synopsis

integer **TopoGeo_AddPolygon**(varchar atopology, geometry apoly, float8 atolerance);

Beschreibung

Fügt ein Polygon zu einer existierenden Topologie hinzu und gibt die Identifikatoren der Maschen aus, aus denen es gebildet ist. Die Begrenzung des gegebenen Polygons wird innerhalb der angegebenen Toleranz an bestehenden Knoten und Kanten gefangen. Gegebenenfalls werden existierende Kanten und Maschen an der Begrenzung des neuen Polygons geteilt.

Verfügbarkeit: 2.0.0

Siehe auch

[TopoGeo_AddPoint](#), [TopoGeo_AddLineString](#), [AddFace](#), [CreateTopology](#)

11.5 Topologie Editoren

11.5.1 ST_AddIsoNode

ST_AddIsoNode — Fügt einen isolierten Knoten zu einer Masche in einer Topologie hinzu und gibt die "nodeid" des neuen Knotens aus. Falls die Masche NULL ist, wird der Knoten dennoch erstellt.

Synopsis

integer **ST_AddIsoNode**(varchar atopology, integer aface, geometry apoint);

Beschreibung

Fügt einen isolierten Knoten mit der Punktlage `apoint` zu einer bestehenden Masche mit der "faceid" `aface` zu einer Topologie `atopology` und gibt die "nodeid" des neuen Knoten aus.

Wenn das Koordinatenreferenzsystem (SRID) der Punktgeometrie nicht mit dem der Topologie übereinstimmt, `apoint` keine Punktgeometrie ist, der Punkt NULL ist, oder der Punkt eine bestehende Kante (auch an den Begrenzungen) schneidet, wird eine Fehlermeldung ausgegeben. Falls der Punkt bereits als Knoten existiert, wird ebenfalls eine Fehlermeldung ausgegeben.

Wenn `aface` nicht NULL ist und `apoint` nicht innerhalb der Masche liegt, wird eine Fehlermeldung ausgegeben.

Verfügbarkeit: 1.?



This method implements the SQL/MM specification. SQL-MM: Topo-Net Routines: X+1.3.1

Beispiele

Siehe auch

[AddNode](#), [CreateTopology](#), [DropTopology](#), [ST_Intersects](#)

11.5.2 ST_AddIsoEdge

ST_AddIsoEdge — Fügt eine isolierte Kante, die durch die Geometrie `alinesstring` festgelegt wird zu einer Topologie hinzu, indem zwei bestehende isolierte Knoten `anode` und `anothernode` verbunden werden. Gibt die "edgeid" der neuen Kante aus.

Synopsis

integer **ST_AddIsoEdge**(varchar atopology, integer anode, integer anothernode, geometry alinestring);

Beschreibung

Fügt eine isolierte Kante, die durch die Geometrie `alinestring` festgelegt wird zu einer Topologie hinzu, indem zwei bestehende isolierte Knoten `anode` und `anothernode` verbunden werden. Gibt die "edgeid" der neuen Kante aus.

Wenn das Koordinatenreferenzsystem (SRID) der Geometrie `alinestring` nicht mit dem der Topologie übereinstimmt, irgendein Eingabewert NULL ist, die Knoten in mehreren Maschen enthalten sind, oder die Knoten Anfangs- oder Endknoten einer bestehenden Kante darstellen, wird eine Fehlermeldung ausgegeben.

Wenn `alinestring` nicht innerhalb der Masche liegt zu der `anode` und `anothernode` gehören, dann wird eine Fehlermeldung ausgegeben.

Wenn der `anode` und der `anothernode` nicht der Anfangs- und Endpunkte von `alinestring` sind, wird eine Fehlermeldung ausgegeben.

Verfügbarkeit: 1.?



This method implements the SQL/MM specification. SQL-MM: Topo-Geo and Topo-Net 3: Routine Details: X.3.4

Beispiele

Siehe auch

[ST_AddIsoNode](#), [ST_IsSimple](#), [ST_Within](#)

11.5.3 ST_AddEdgeNewFaces

ST_AddEdgeNewFaces — Fügt eine Kante hinzu. Falls dabei eine Masche aufgetrennt wird, so wird die ursprüngliche Masche gelöscht und durch zwei neue Maschen ersetzt.

Synopsis

integer **ST_AddEdgeNewFaces**(varchar atopology, integer anode, integer anothernode, geometry acurve);

Beschreibung

Fügt eine Kante hinzu. Falls dabei eine Masche aufgetrennt wird, so wird die ursprüngliche Masche gelöscht und durch zwei neue Maschen ersetzt. Gibt die ID der hinzugefügten Kante aus.

Führt ein entsprechendes Update auf alle verbundenen Kanten und Beziehungen durch.

Wenn irgendwelche Übergabewerte NULL sind, die angegebenen Knoten unbekannt sind (müssen bereits in der `node` Tabelle des Schemas "topology" existieren), `acurve` kein LINESTRING ist, oder `anode` und `anothernode` nicht die Anfangs- und Endpunkte von `acurve` sind, dann wird eine Fehlermeldung ausgegeben.

Wenn das Koordinatenreferenzsystem (SRID) der Geometrie `acurve` nicht mit jener der Topologie übereinstimmt, wird eine Fehlermeldung ausgegeben.

Verfügbarkeit: 2.0



This method implements the SQL/MM specification. SQL-MM: Topo-Geo and Topo-Net 3: Routine Details: X.3.12

Beispiele

Siehe auch

[ST_RemEdgeNewFace](#)

[ST_AddEdgeModFace](#)

11.5.4 ST_AddEdgeModFace

ST_AddEdgeModFace — Fügt eine Kante hinzu. Falls dabei eine Masche aufgetrennt wird, so wird die ursprüngliche Masche angepasst und eine weitere Masche hinzugefügt.

Synopsis

integer **ST_AddEdgeModFace**(varchar atopology, integer anode, integer anothernode, geometry acurve);

Beschreibung

Fügt eine Kante hinzu. Falls dabei eine Masche aufgetrennt wird, so wird die ursprüngliche Masche angepasst und eine weitere Masche hinzugefügt.



Note

Wenn möglich, wird die neue Masche auf der linken Seite der neuen Kante erstellt. Dies ist jedoch nicht möglich, wenn die Masche auf der linken Seite die (unbegrenzte) Grundmenge der Maschen darstellt.

Gibt die id der hinzugefügten Kante zurück.

Führt ein entsprechendes Update auf alle verbundenen Kanten und Beziehungen durch.

Wenn irgendwelche Übergabewerte NULL sind, die angegebenen Knoten unbekannt sind (müssen bereits in der `node` Tabelle des Schemas "topology" existieren), `acurve` kein `LINESTRING` ist, oder `anode` und `anothernode` nicht die Anfangs- und Endpunkte von `acurve` sind, dann wird eine Fehlermeldung ausgegeben.

Wenn das Koordinatenreferenzsystem (SRID) der Geometrie `acurve` nicht mit jener der Topologie übereinstimmt, wird eine Fehlermeldung ausgegeben.

Verfügbarkeit: 2.0



This method implements the SQL/MM specification. SQL-MM: Topo-Geo and Topo-Net 3: Routine Details: X.3.13

Beispiele

Siehe auch

[ST_RemEdgeModFace](#)

[ST_AddEdgeNewFaces](#)

11.5.5 ST_RemEdgeNewFace

ST_RemEdgeNewFace — Entfernt eine Kante. Falls die gelöschte Kante zwei Maschen voneinander getrennt hat, werden die ursprünglichen Maschen gelöscht und durch einer neuen Masche ersetzt.

Synopsis

integer **ST_RemEdgeNewFace**(varchar atopolology, integer anedge);

Beschreibung

Entfernt eine Kante. Falls die gelöschte Kante zwei Maschen voneinander getrennt hat, werden die ursprünglichen Maschen gelöscht und durch einer neuen Masche ersetzt.

Gibt die ID der neu erzeugten Masche aus; oder NULL wenn keine Masche erstellt wurde. Es wird keine neue Masche erstellt, wenn die gelöschte Kante defekt oder isoliert ist, oder wenn sie die Begrenzung der Maschengrundmenge darstellt (wodurch die Grundmenge womöglich von der anderen Seite in die Masche fließen könnte).

Führt ein entsprechendes Update auf alle verbundenen Kanten und Beziehungen durch.

Wenn eine Kante an der Definition einer bestehenden TopoGeometry beteiligt ist, kann sie nicht gelöscht werden. Wenn irgendeine TopoGeometry nur durch eine der beiden Maschen definiert ist (und nicht auch durch die andere), können die beiden Maschen nicht "geheilt" werden.

Wenn irgendwelche Übergabewerte NULL sind, die angegebene Kante unbekannt ist (muss bereits in der edge Tabelle des Schemas "topology" existieren), oder die Bezeichnung der Topologie ungültig ist, dann wird eine Fehlermeldung ausgegeben.

Verfügbarkeit: 2.0



This method implements the SQL/MM specification. SQL-MM: Topo-Geo and Topo-Net 3: Routine Details: X.3.14

Beispiele

Siehe auch

[ST_RemEdgeModFace](#)

[ST_AddEdgeNewFaces](#)

11.5.6 ST_RemEdgeModFace

ST_RemEdgeModFace — Entfernt eine Kante. Falls die gelöschte Kante zwei Maschen voneinander getrennt hat, wird eine der Maschen gelöscht und die andere so geändert, dass sie den Platz der beiden ursprünglichen Maschen einnimmt.

Synopsis

integer **ST_RemEdgeModFace**(varchar atopolology, integer anedge);

Beschreibung

Entfernt eine Kante. Falls die gelöschte Kante zwei Maschen voneinander getrennt hat, wird eine der Maschen gelöscht und die andere so geändert, dass sie den Platz der beiden ursprünglichen Maschen einnimmt. Vorzugsweise wird die Masche auf der rechten Seite beibehalten, so wie bei **ST_AddEdgeModFace**. Gibt die ID der verbliebenen Masche aus.

Führt ein entsprechendes Update auf alle verbundenen Kanten und Beziehungen durch.

Wenn eine Kante an der Definition einer bestehenden TopoGeometry beteiligt ist, kann sie nicht gelöscht werden. Wenn irgendeine TopoGeometry nur durch eine der beiden Maschen definiert ist (und nicht auch durch die andere), können die beiden Maschen nicht "geheilt" werden.

Wenn irgendwelche Übergabewerte NULL sind, die angegebene Kante unbekannt ist (muss bereits in der edge Tabelle des Schemas "topology" existieren), oder die Bezeichnung der Topologie ungültig ist, dann wird eine Fehlermeldung ausgegeben.

Verfügbarkeit: 2.0



This method implements the SQL/MM specification. SQL-MM: Topo-Geo and Topo-Net 3: Routine Details: X.3.15

Beispiele**Siehe auch**[ST_AddEdgeModFace](#)[ST_RemEdgeNewFace](#)**11.5.7 ST_ChangeEdgeGeom**

ST_ChangeEdgeGeom — Ändert die geometrische Form der Kante, ohne sich auf topologische Struktur auszuwirken.

Synopsis

integer **ST_ChangeEdgeGeom**(varchar atopology, integer anedge, geometry acurve);

Beschreibung

Ändert die geometrische Form der Kante, ohne sich auf topologische Struktur auszuwirken.

Wenn irgendwelche Übergabewerte NULL sind, die angegebene Kante nicht in der `edge` Tabelle des Schemas "topology" existiert, `acurve` kein `LINESTRING` ist, `anode` und `anothernode` nicht die Anfangs- und Endpunkte von `acurve` sind, oder die Modifikation die zugrundeliegende Topologie ändern würde, dann wird eine Fehlermeldung ausgegeben.

Wenn das Koordinatenreferenzsystem (SRID) der Geometrie `acurve` nicht mit jener der Topologie übereinstimmt, wird eine Fehlermeldung ausgegeben.

Wenn die neue `acurve` nicht "simple" ist, wird eine Fehlermeldung ausgegeben.

Wenn beim Verschieben der Kante von der alten auf die neue Position ein Hindernis auftritt, wird eine Fehlermeldung ausgegeben.

Verfügbarkeit: 1.1.0

Erweiterung: 2.0.0 Erzwingung topologischer Konsistenz hinzugefügt



This method implements the SQL/MM specification. SQL-MM: Topo-Geo and Topo-Net 3: Routine Details X.3.6

Beispiele

```
SELECT topology.ST_ChangeEdgeGeom('ma_topo', 1,
                                ST_GeomFromText('LINESTRING(227591.9 893900.4,227622.6 893844.3,227641.6
                                893816.6, 227704.5 893778.5)', 26986) );
-----
Edge 1 changed
```

Siehe auch[ST_AddEdgeModFace](#)[ST_RemEdgeModFace](#)[ST_ModEdgeSplit](#)**11.5.8 ST_ModEdgeSplit**

ST_ModEdgeSplit — Trennt eine Kante auf, indem ein neuer Knoten entlang einer bestehenden Kante erstellt wird. Ändert die ursprüngliche Kante und fügt eine neue Kante hinzu.

Synopsis

integer **ST_ModEdgeSplit**(varchar atopology, integer anedge, geometry apoint);

Beschreibung

Trennt eine Kante auf, indem ein neuer Knoten entlang einer bestehenden Kante erstellt wird. Ändert die ursprüngliche Kante und fügt eine neue Kante hinzu. Alle bestehenden Kanten und Beziehungen werden entsprechend aktualisiert. Gibt den Identifikator des neu hinzugefügten Knotens aus.

Verfügbarkeit: 1.?

Änderung: 2.0 - In Vorgängerversionen fälschlicherweise als ST_ModEdgesSplit bezeichnet



This method implements the SQL/MM specification. SQL-MM: Topo-Geo and Topo-Net 3: Routine Details: X.3.9

Beispiele

```
-- Eine Kante hinzufügen --
SELECT topology.AddEdge('ma_topo', ST_GeomFromText('LINESTRING(227592 893910, 227600 893910)', 26986) ) As edgeid;

-- edgeid-
3

-- Eine Kante spalten/teilen --
SELECT topology.ST_ModEdgeSplit('ma_topo', 3, ST_SetSRID(ST_Point(227594,893910),26986) ) As node_id;
       node_id
-----
7
```

Siehe auch

[ST_NewEdgesSplit](#), [ST_ModEdgeHeal](#), [ST_NewEdgeHeal](#), [AddEdge](#)

11.5.9 ST_ModEdgeHeal

ST_ModEdgeHeal — "Heilt" zwei Kanten, indem der verbindende Knoten gelöscht wird, die erste Kante modifiziert und die zweite Kante gelöscht wird. Gibt die ID des gelöschten Knoten zurück.

Synopsis

int **ST_ModEdgeHeal**(varchar atopology, integer anedge, integer anotheredge);

Beschreibung

"Heilt" zwei Kanten, indem der verbindende Knoten gelöscht wird, die erste Kante modifiziert und die zweite Kante gelöscht wird. Gibt die ID des gelöschten Knoten zurück. Aktualisiert alle verbundenen Kanten und Beziehungen dementsprechend.

Verfügbarkeit: 2.0



This method implements the SQL/MM specification. SQL-MM: Topo-Geo and Topo-Net 3: Routine Details: X.3.9

Siehe auch

[ST_ModEdgeSplit](#) [ST_NewEdgesSplit](#)

11.5.10 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 Ausrichtung wie die erste Kante hat.

Synopsis

```
int ST_NewEdgeHeal(varchar atopology, integer anedge, integer anotheredge);
```

Beschreibung

"Heilt" zwei Kanten, indem der verbindende Knoten und beide Kanten gelöscht werden. Die beiden Kanten werden durch eine Kante ersetzt, welche dieselbe Ausrichtung wie die erste Kante hat. Gibt die ID der neuen Kante aus. Aktualisiert alle verbundenen Kanten und Beziehungen dementsprechend.

Verfügbarkeit: 2.0



This method implements the SQL/MM specification. SQL-MM: Topo-Geo and Topo-Net 3: Routine Details: X.3.9

Siehe auch

[ST_ModEdgeHeal](#) [ST_ModEdgeSplit](#) [ST_NewEdgesSplit](#)

11.5.11 ST_MoveIsoNode

`ST_MoveIsoNode` — Verschiebt einen isolierten Knoten in einer Topologie von einer Stelle an eine andere. Falls die neue Geometrie `apoint` bereits als Knoten existiert, wird eine Fehlermeldung ausgegeben. Gibt eine Beschreibung der Verschiebung aus.

Synopsis

```
text ST_MoveIsoNode(varchar atopology, integer anedge, geometry apoint);
```

Beschreibung

Verschiebt einen isolierten Knoten in einer Topologie von einer Stelle an eine andere. Falls die neue Geometrie `apoint` bereits als Knoten existiert, wird eine Fehlermeldung ausgegeben.

Wenn irgendein Übergabewert `NULL` ist, `apoint` keine Punktgeometrie ist, der bestehende Knoten nicht isoliert ist (Anfangs- oder Endpunkt einer bestehenden Kante ist), oder die Lage des neuen Knoten eine bestehende Kante schneidet (auch an den Endpunkten), dann wird eine Fehlermeldung ausgegeben.

Wenn das Koordinatenreferenzsystem (SRID) der Punktgeometrie nicht mit jener der Topologie übereinstimmt, wird eine Fehlermeldung ausgegeben.

Verfügbarkeit: 1.?



This method implements the SQL/MM specification. SQL-MM: Topo-Net Routines: X.3.2

Beispiele

```
-- Einen freistehenden/isolierten Knoten ohne Masche hinzufügen --
SELECT topology.ST_AddIsoNode('ma_topo', NULL, ST_GeomFromText('POINT(227579 893916)', 26986) ) As nodeid;
nodeid
-----
      7
-- Den neuen Knoten bewegen --
SELECT topology.ST_MoveIsoNode('ma_topo', 7, ST_GeomFromText('POINT(227579.5 893916.5)', 26986) ) As descrip;
descrip
-----
Isolated Node 7 moved to location 227579.5,893916.5
```

Siehe auch

[ST_AddIsoNode](#)

11.5.12 ST_NewEdgesSplit

ST_NewEdgesSplit — Trennt eine Kante auf, indem ein neuer Knoten entlang einer bestehenden Kante erstellt, die ursprüngliche Kante gelöscht und durch zwei neue Kanten ersetzt wird. Gibt die ID des neu erstellten Knotens aus, der die neuen Kanten verbindet.

Synopsis

integer **ST_NewEdgesSplit**(varchar atopology, integer anedge, geometry apoint);

Beschreibung

Trennt eine Kante mit der Kanten-ID *anedge* auf, indem ein neuer Knoten mit der Punktlage *apoint* entlang der aktuellen Kante erstellt, die ursprüngliche Kante gelöscht und durch zwei neue Kanten ersetzt wird. Gibt die ID des neu erstellten Knotens aus, der die neuen Kanten verbindet. Aktualisiert alle verbundenen Kanten und Beziehungen dementsprechend.

Wenn das Koordinatenreferenzsystem (SRID) der Punktgeometrie nicht mit dem der Topologie übereinstimmt, *apoint* keine Punktgeometrie ist, der Punkt NULL ist, der Punkt bereits als Knoten existiert, die Kante mit einer bestehenden Kante nicht zusammenpasst, oder der Punkt nicht innerhalb der Kante liegt, dann wird eine Fehlermeldung ausgegeben.

Verfügbarkeit: 1.?



This method implements the SQL/MM specification. SQL-MM: Topo-Net Routines: X.3.8

Beispiele

```
-- Einen Knoten hinzufügen --
SELECT topology.AddEdge('ma_topo', ST_GeomFromText('LINESTRING(227575 893917,227592 893900)', 26986) ) As edgeid;
-- result-
edgeid
-----
      2
-- Split the new edge --
SELECT topology.ST_NewEdgesSplit('ma_topo', 2, ST_GeomFromText('POINT(227578.5 893913.5)', 26986) ) As newnodeid;
```

```
newnodeid
-----
        6
```

Siehe auch

[ST_ModEdgeSplit](#) [ST_ModEdgeHeal](#) [ST_NewEdgeHeal](#) [AddEdge](#)

11.5.13 ST_RemoveIsoNode

ST_RemoveIsoNode — Löscht einen isolierten Knoten und gibt eine Beschreibung der getroffenen Maßnahmen aus. Falls der Knoten nicht isoliert ist (ist der Anfangs- oder der Endpunkt einer Kante), wird eine Fehlermeldung ausgegeben.

Synopsis

```
text ST_RemoveIsoNode(varchar atopology, integer anode);
```

Beschreibung

Löscht einen isolierten Knoten und gibt eine Beschreibung der getroffenen Maßnahmen aus. Falls der Knoten nicht isoliert ist (ist der Anfangs- oder der Endpunkt einer Kante), wird eine Fehlermeldung ausgegeben.

Verfügbarkeit: 1.?



This method implements the SQL/MM specification. SQL-MM: Topo-Geo and Topo-Net 3: Routine Details: X+1.3.3

Beispiele

```
-- Einen alleinstehenden Knoten, ohne Masche, entfernen --
SELECT topology.ST_RemoveIsoNode('ma_topo', 7 ) As result;
           result
-----
Isolated node 7 removed
```

Siehe auch

[ST_AddIsoNode](#)

11.5.14 ST_RemoveIsoEdge

ST_RemoveIsoEdge — Removes an isolated edge and returns description of action. If the edge is not isolated, then an exception is thrown.

Synopsis

```
text ST_RemoveIsoEdge(varchar atopology, integer anedge);
```


Beschreibung

Removes an isolated edge and returns description of action. If the edge is not isolated, then an exception is thrown.

Verfügbarkeit: 1.?



This method implements the SQL/MM specification. SQL-MM: Topo-Geo and Topo-Net 3: Routine Details: X+1.3.3

Beispiele

```
-- Einen alleinstehenden Knoten, ohne Masche, entfernen --
SELECT topology.ST_RemoveIsoNode('ma_topo', 7 ) As result;
        result
-----
Isolated node 7 removed
```

Siehe auch

[ST_AddIsoNode](#)

11.6 Zugriffsfunktionen zur Topologie

11.6.1 GetEdgeByPoint

GetEdgeByPoint — Findet die edge-id einer Kante die einen gegebenen Punkt schneidet.

Synopsis

integer **GetEdgeByPoint**(varchar atopology, geometry apoint, float8 tol);

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 der Punkt keine Kante schneidet, wird 0 (Null) zurückgegeben.

Wenn eine tolerance > 0 angegeben ist und mehr als eine Kante in diesem Bereich existiert, wird eine Fehlermeldung ausgegeben.



Note

Wenn tolerance = 0, dann benutzt die Funktion ST_Intersects, sonst ST_DWithin.

Verfügbarkeit: 2.0.0 - benötigt GEOS >= 3.3.0.

Beispiele

Die folgenden Beispiele benutzen die Kanten, die wir in [AddEdge](#) erzeugt haben

```
SELECT topology.GetEdgeByPoint('ma_topo',geom, 1) As withlmtol, topology.GetEdgeByPoint(' ←
      ma_topo',geom,0) As withnotol
FROM ST_GeomFromEWKT('SRID=26986;POINT(227622.6 893843)') As geom;
withlmtol | withnotol
-----+-----
          2 |          0
```

```
SELECT topology.GetEdgeByPoint('ma_topo',geom, 1) As nearnode
FROM ST_GeomFromEWKT('SRID=26986;POINT(227591.9 893900.4)') As geom;

-- gibt eine Fehlermeldung zurück --
ERROR:  Two or more edges found
```

Siehe auch

[AddEdge](#), [GetNodeByPoint](#), [GetFaceByPoint](#)

11.6.2 GetFaceByPoint

GetFaceByPoint — Findet die face-id einer Masche, die einen gegebenen Punkt schneidet

Synopsis

integer **GetFaceByPoint**(varchar atopology, geometry apoint, float8 tol);

Beschreibung

Die ID einer Masche abfragen, die einen Punkt schneidet.

Die Funktion gibt eine Ganzzahl (id-face) für eine Topologie, einen POINT und eine Toleranz aus. Wenn tolerance = 0, dann muss der Punkt die Masche schneiden.

Wenn der Punkt keine Masche schneidet, wird 0 (Null) ausgegeben.

Wenn eine tolerance > 0 angegeben ist und mehr als eine Masche in diesem Bereich existiert, wird eine Fehlermeldung ausgegeben.



Note

Wenn tolerance = 0, wird von der Funktion ST_Intersects angewendet, ansonsten ST_DWithin.

Verfügbarkeit: 2.0.0 - benötigt GEOS >= 3.3.0.

Beispiele

Die folgenden Beispiele benutzen die Kanten und Maschen, die wir in [AddFace](#) erzeugt haben

```
SELECT topology.GetFaceByPoint('ma_topo',geom, 10) As with1mtol, topology.GetFaceByPoint('ma_topo',geom,0) As withnotol
FROM ST_GeomFromEWKT('POINT(234604.6 899382.0)') As geom;

with1mtol | withnotol
-----+-----
1 | 0
```

```
SELECT topology.GetFaceByPoint('ma_topo',geom, 1) As nearnode
FROM ST_GeomFromEWKT('POINT(227591.9 893900.4)') As geom;

-- Fehlermeldung --
ERROR: Two or more faces found
```

Siehe auch

[AddFace](#), [GetNodeByPoint](#), [GetEdgeByPoint](#)

11.6.3 GetNodeByPoint

GetNodeByPoint — Findet zu der Lage eines Punktes die ID eines Knotens

Synopsis

integer **GetNodeByPoint**(varchar atopology, geometry point, float8 tol);

Ruft zu der Lage eines Punktes die ID eines Knotens ab

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 an dieser Stelle kein Knoten existiert, wird 0 (null) zurückgegeben.

Wenn eine tolerance > 0 angegeben ist und mehr als ein Knoten in diesem Bereich existiert, wird eine Fehlermeldung ausgegeben.



Note

Wenn tolerance = 0, dann benutzt die Funktion ST_Intersects, sonst ST_DWithin.

Verfügbarkeit: 2.0.0 - benötigt GEOS >= 3.3.0.

Beispiele

Die folgenden Beispiele benutzen die Kanten, die wir in [AddEdge](#) erzeugt haben

```
SELECT topology.GetNodeByPoint('ma_topo',geom, 1) As nearnode
FROM ST_GeomFromEWKT('SRID=26986;POINT(227591.9 893900.4)') As geom;

-----
2
```

```
SELECT topology.GetNodeByPoint('ma_topo',geom, 1000) As too_much_tolerance
FROM ST_GeomFromEWKT('SRID=26986;POINT(227591.9 893900.4)') As geom;

----Fehlermeldung--
ERROR:  Two or more nodes found
```

Siehe auch

[AddEdge](#), [GetEdgeByPoint](#), [GetFaceByPoint](#)

11.6.4 GetTopologyID

GetTopologyID — Gibt für den Namen einer Topologie die ID der Topologie in der Tabelle "topology.topology" aus.

Synopsis

integer **GetTopologyID**(varchar toponame);

Beschreibung

Gibt für den Namen einer Topologie, die ID der Topologie in der Tabelle "topology.topology" aus.

Verfügbarkeit: 1.?

Beispiele

```
SELECT topology.GetTopologyID('ma_topo') As topo_id;
 topo_id
-----
      1
```

Siehe auch

[CreateTopology](#), [DropTopology](#), [GetTopologyName](#), [GetTopologySRID](#)

11.6.5 GetTopologySRID

GetTopologySRID — Gibt für den Namen einer Topologie, die SRID der Topologie in der Tabelle "topology.topology" aus.

Synopsis

integer **GetTopologyID**(varchar toponame);

Beschreibung

Gibt für den Namen einer Topologie, die ID des Koordinatenreferenzsystems in der Tabelle "topology.topology" aus.

Verfügbarkeit: 2.0.0

Beispiele

```
SELECT topology.GetTopologySRID('ma_topo') As SRID;
SRID
-----
4326
```

Siehe auch

[CreateTopology](#), [DropTopology](#), [GetTopologyName](#), [GetTopologyID](#)

11.6.6 GetTopologyName

`GetTopologyName` — Gibt für die ID der Topologie, den Namen der Topologie (Schema) zurück.

Synopsis

`varchar` **GetTopologyName**(integer topology_id);

Beschreibung

Gibt für die ID einer Topologie, den Namen (Schema) der Topologie in der Tabelle "topology.topology" aus.

Verfügbarkeit: 1.?

Beispiele

```
SELECT topology.GetTopologyName(1) As topo_name;
topo_name
-----
ma_topo
```

Siehe auch

[CreateTopology](#), [DropTopology](#), [GetTopologyID](#), [GetTopologySRID](#)

11.6.7 ST_GetFaceEdges

`ST_GetFaceEdges` — Gibt die Kanten, die `aface` begrenzen, sortiert aus.

Synopsis

`getfaceedges_returntype` **ST_GetFaceEdges**(varchar atopology, integer aface);

Beschreibung

Gibt die Kanten, die `aface` begrenzen, sortiert aus. Jede Ausgabe besteht aus einer Sequenz und einer "edgeid". Die Sequenznummern beginnen mit dem Wert 1.

Die Aufzählung der Kanten des Rings beginnt mit der Kante mit dem niedrigsten Identifikator. Die Reihenfolge der Kanten folgt der Drei-Finger-Regel (die begrenzte Masche liegt links von den gerichteten Kanten).

Verfügbarkeit: 2.0



This method implements the SQL/MM specification. SQL-MM 3 Topo-Geo and Topo-Net 3: Routine Details: X.3.5

Beispiele

```
-- Gibt die Kanten zurück, welche die Masche 1 begrenzen
SELECT (topology.ST_GetFaceEdges('tt', 1)).*;
```

```
-- result --
sequence | edge
-----+-----
         1 |    -4
         2 |     5
         3 |     7
         4 |    -6
         5 |     1
         6 |     2
         7 |     3
(7 rows)
```

```
-- Gibt die Sequenz, die ID und die Geometrie der Kanten zurück,
-- welche die Masche 1 begrenzen
-- Wenn Sie nur die Geometrie und die Sequenzen benötigen, können Sie
-- ST_GetFaceGeometry verwenden
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](#)

11.6.8 ST_GetFaceGeometry

ST_GetFaceGeometry — Gibt für eine Topologie und eine bestimmte Maschen-ID das Polygon zurück.

Synopsis

geometry **ST_GetFaceGeometry**(varchar atopology, integer aface);

Beschreibung

Gibt für eine Topologie und eine bestimmte Maschen-ID das Polygon zurück. Erstellt das Polygon aus den Kanten, die die Masche aufbauen.

Verfügbarkeit: 1.?



This method implements the SQL/MM specification. SQL-MM 3 Topo-Geo and Topo-Net 3: Routine Details: X.3.16

Beispiele

```
-- Gibt den WKT des mit AddFace hinzugefügten Polygons zurück
SELECT ST_AsText(topology.ST_GetFaceGeometry('ma_topo', 1)) As facegeomwkt;
-- result --
facegeomwkt
-----
POLYGON((234776.9 899563.7,234896.5 899456.7,234914 899436.4,234946.6 899356.9,
234872.5 899328.7,234891 899285.4,234992.5 899145,234890.6 899069,
234755.2 899255.4,234612.7 899379.4,234776.9 899563.7))
```

Siehe auch

[AddFace](#)

11.6.9 GetRingEdges

GetRingEdges — Gibt eine sortierte Liste von mit Vorzeichen versehenen Identifikatoren der Kanten zurück, die angetroffen werden, wenn man an der Seite der gegebenen Kante entlangwandert.

Synopsis

```
getfaceedges_returntype GetRingEdges(varchar atopology, integer aring, integer max_edges=null);
```

Beschreibung

Gibt eine sortierte Liste von mit Vorzeichen versehenen Identifikatoren der Kanten zurück, die angetroffen werden, wenn man an der Seite der gegebenen Kante entlangwandert. Jede Ausgabe besteht aus einer Sequenz und einer mit einem Vorzeichen versehenen ID der Kante. Die Sequenz beginnt mit dem Wert 1.

Wenn Sie eine positive ID für die Kante übergeben, beginnt der Weg auf der linken Seite der entsprechenden Kante und folgt der Ausrichtung der Kante. Wenn Sie eine negative ID für die Kante übergeben, beginnt der Weg auf der rechten Seite der Kante und verläuft rückwärts.

Wenn `max_edges` nicht NULL ist, so beschränkt dieser Parameter die Anzahl der von dieser Funktion ausgegebenen Datensätze. Ist als Sicherheitsparameter für den Umgang mit möglicherweise invaliden Topologien gedacht.



Note

Diese Funktion verwendet Metadaten um die Kanten eines Ringes zu verbinden.

Verfügbarkeit: 2.0.0

Siehe auch

[ST_GetFaceEdges](#), [GetNodeEdges](#)

11.6.10 GetNodeEdges

GetNodeEdges — Gibt für einen Knoten die sortierte Menge der einfallenden Kanten aus.

Synopsis

```
getfaceedges_returntype GetNodeEdges(varchar atopology, integer anode);
```

Beschreibung

Returns an ordered set of edges incident to the given node. Each output consists of a sequence and a signed edge id. Sequence numbers start with value 1. A positive edge starts at the given node. A negative edge ends into the given node. Closed edges will appear twice (with both signs). Order is clockwise starting from northbound.

**Note**

Diese Funktion errechnet die Reihenfolge, anstatt sie aus den Metadaten abzuleiten und kann daher verwendet werden, um die Kanten eines Ringes zu verbinden.

Verfügbarkeit: 2.0

Siehe auch

[GetRingEdges](#), [ST_Azimuth](#)

11.7 Topologie Verarbeitung

11.7.1 Polygonize

Polygonize — Findet und registriert alle Maschen, die durch die Kanten der Topologie festgelegt sind

Synopsis

text **Polygonize**(varchar toponame);

Beschreibung

Registriert alle Maschen, die aus den Kanten der topologischen Elementarstrukturen erstellt werden können

Von der Zieltopologie wird angenommen, dass sie keine sich selbst überschneidenden Kanten enthält.

**Note**

Da bereits bekannte Maschen erkannt werden, kann Polygonize gefahrlos mehrere Male auf die selbe Topologie angewendet werden.

**Note**

Von dieser Funktion werden die Attribute "next_left_edge" und "next_right_edge" der Tabelle "edge" weder verwendet noch gesetzt.

Verfügbarkeit: 2.0.0

Siehe auch

[AddFace](#), [ST_Polygonize](#)

11.7.2 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 zurückgegeben.

Synopsis

integer **AddNode**(varchar toponame, geometry apoint, boolean allowEdgeSplitting=false, boolean computeContainingFace=false);

Beschreibung

Fügt einen Knotenpunkt zu der Tabelle "node" in dem vorgegebenen topologischen Schema hinzu. Die Funktion **AddEdge** fügt die Anfangs- und Endpunkte einer Kante automatisch hinzu, wenn sie aufgerufen wird. Deshalb ist es nicht notwendig die Knoten einer Kante explizit anzufügen.

Falls eine Kante aufgefunden wird, die den Knoten kreuzt, dann wird entweder eine Fehlermeldung ausgegeben, oder die Kante aufgetrennt. Dieses Verhalten hängt vom Wert des Parameters `allowEdgeSplitting` ab.

Wenn `computeContainingFace` `TRUE` ist, dann wird für einen neu hinzugefügten Knoten die richtige Begrenzung der Masche berechnet.



Note

Wenn die Geometrie `apoint` bereits als Knoten existiert, dann wird der Knoten nicht hinzugefügt und der bestehende Knoten ausgegeben.

Verfügbarkeit: 2.0.0

Beispiele

```
SELECT topology.AddNode('ma_topo', ST_GeomFromText('POINT(227641.6 893816.5)', 26986) ) As ↵
       nodeid;
-- result --
nodeid
-----
4
```

Siehe auch

AddEdge, **CreateTopology**

11.7.3 AddEdge

AddEdge — Fügt die Kante eines Linienzugs in der Tabelle "edge", und die zugehörigen Anfangs- und Endpunkte in die Knotenpunkttabelle, des jeweiligen topologischen Schemas ein. Dabei wird die übergebene Linienzuggeometrie verwendet und die `edgeid` der neuen (oder bestehenden) Kante ausgegeben.

Synopsis

integer **AddEdge**(varchar toponame, geometry aline);

Beschreibung

Fügt eine Kante in der Tabelle "edge", und die zugehörigen Knoten in die Tabelle "node", des jeweiligen Schemas `toponame` ein. Dabei wird die übergebene Linienzuggeometrie verwendet und die `edgeid` des neuen oder des bestehenden Datensatzes ausgegeben. Die neu hinzugefügte Kante hat auf beiden Seiten die Masche für die Grundmenge/"Universum" und verweist auf sich selbst.



Note

Wenn die Geometrie `aline` eine bestehende Kante kreuzt, überlagert, beinhaltet oder in ihr enthalten ist, dann wird eine Fehlermeldung ausgegeben und die Kante wird nicht hinzugefügt.



Note

Die Geometrie von `aline` muss dieselbe `SRID` aufweisen wie die Topologie, ansonsten wird die Fehlermeldung "invalid spatial reference sys error" ausgegeben.

Verfügbarkeit: 2.0.0 benötigt GEOS $\geq$ 3.3.0.

Beispiele

```
SELECT topology.AddEdge('ma_topo', ST_GeomFromText('LINESTRING(227575.8 893917.2,227591.9 893900.4)', 26986) ) As edgeid;
-- Ergebnis-
edgeid
-----
1

SELECT topology.AddEdge('ma_topo', ST_GeomFromText('LINESTRING(227591.9 893900.4,227622.6 893844.2,227641.6 893816.5,
227704.5 893778.5)', 26986) ) As edgeid;
-- Ergebnis --
edgeid
-----
2

SELECT topology.AddEdge('ma_topo', ST_GeomFromText('LINESTRING(227591.2 893900, 227591.9 893900.4,
227704.5 893778.5)', 26986) ) As edgeid;
-- Fehlermeldung --
ERROR:  Edge intersects (not on endpoints) with existing edge 1
```

Siehe auch

[TopoGeo_AddLineString](#), [CreateTopology](#), [Section 4.3.1](#)

11.7.4 AddFace

AddFace — Registriert die Elementarstruktur einer Masche in einer Topologie und gibt den Identifikator der Masche aus.

Synopsis

integer **AddFace**(varchar toponame, geometry apolygon, boolean force_new=false);

Beschreibung

Registriert die Elementarstruktur einer Masche in einer Topologie und gibt den Identifikator der Masche aus.

Bei einer neu hinzugefügten Masche werden die Kanten die ihre Begrenzung bilden und jene die innerhalb der Masche liegen aktualisiert, damit diese die richtigen Werte in den Attributen "left_face" und "right_face" aufweisen. Isolierte Knoten innerhalb der Masche werden ebenfalls aktualisiert, damit das Attribut "containing_face" die richtigen Werte aufweist.



Note

Von dieser Funktion werden die Attribute "next_left_edge" und "next_right_edge" der Tabelle "edge" weder verwendet noch gesetzt.

Es wird angenommen, dass die Zieltopologie valide ist (keine sich selbst überschneidenden Kanten enthält). Eine Fehlermeldung wird ausgegeben, wenn: Die Begrenzung des Polygons nicht vollständig durch bestehende Kanten festgelegt ist oder das Polygon eine bereits bestehende Masche überlappt.

Falls die Geometrie `apolygon` bereits als Masche existiert, dann: wenn `force_new` FALSE (der Standardwert) ist, dann wird die bestehende Masche zurückgegeben; wenn `force_new` TRUE ist, dann wird der neu registrierten Masche eine neue ID zugewiesen.



Note

Wenn eine bestehende Masche neu registriert wird (`force_new=true`), werden keine Maßnahmen durchgeführt um hängende Verweise auf eine bestehende Masche in den Tabellen "edge", "node" und "relation" zu bereinigen, noch wird der das Attribut MBR der bestehenden Masche aktualisiert. Es ist die Aufgabe des Aufrufers sich um dies zu kümmern.



Note

Die Geometrie von `apolygon` muss dieselbe SRID aufweisen wie für die Topologie festgelegt, ansonsten wird die Fehlermeldung "invalid spatial reference sys error" ausgegeben.

Verfügbarkeit: 2.0.0

Beispiele

```
-- zuert werden die Kanten hinzugefügt - mittels "generate_series" als Iterator
-- (nur bei Polygonen mit < 10000 Punkten, wegen des Maximums in "generate_series")
SELECT topology.AddEdge('ma_topo', ST_MakeLine(ST_PointN(geom,i), ST_PointN(geom, i + 1) )) ↔
  As edgeid
  FROM (SELECT ST_NPoints(geom) AS npt, geom
        FROM
          (SELECT ST_Boundary(ST_GeomFromText('POLYGON((234896.5 899456.7,234914 ↔
            899436.4,234946.6 899356.9,234872.5 899328.7,
            234891 899285.4,234992.5 899145, 234890.6 899069,234755.2 899255.4,
            234612.7 899379.4,234776.9 899563.7,234896.5 899456.7))', 26986) ) As geom
        ) As geoms) As facen CROSS JOIN generate_series(1,10000) As i
  WHERE i < npt;
-- result --
edgeid
-----
3
4
5
6
```

```

      7
      8
      9
     10
     11
     12
(10 rows)

-- dann wird die Masche hinzugefügt -
SELECT topology.AddFace('ma_topo',
  ST_GeomFromText('POLYGON((234896.5 899456.7,234914 899436.4,234946.6 899356.9,234872.5  ←
    899328.7,
    234891 899285.4,234992.5 899145, 234890.6 899069,234755.2 899255.4,
    234612.7 899379.4,234776.9 899563.7,234896.5 899456.7))', 26986) ) As faceid;
-- result --
faceid
-----
1

```

Siehe auch

[AddEdge](#), [CreateTopology](#), [Section 4.3.1](#)

11.7.5 ST_Simplify

ST_Simplify — Gibt für eine TopoGeometry eine "vereinfachte" geometrische Version zurück. Verwendet den Douglas-Peucker Algorithmus.

Synopsis

```
geometry ST_Simplify(TopoGeometry geomA, float tolerance);
```

Beschreibung

Gibt für eine TopoGeometry eine "vereinfachte" geometrische Version zurück. Wendet den Douglas-Peucker Algorithmus auf jede Kante an.

**Note**

Es kann vorkommen, dass die zurückgegebene Geometrie weder "simple" noch valide ist. Das Auftrennen der Kanten kann helfen, die Simplizität/Validität zu erhalten.

Wird vom GEOS Modul ausgeführt

Verfügbarkeit: 2.1.0

Siehe auch

Geometrie [ST_Simplify](#), [ST_IsSimple](#), [ST_IsValid](#), [ST_ModEdgeSplit](#)

11.8 TopoGeometry Konstruktoren

11.8.1 CreateTopoGeom

CreateTopoGeom — Erzeugt ein neues topologisch geometrisches Objekt aus einem Feld mit topologischen Elementen - tg_type: 1:[multi]point, 2:[multi]line, 3:[multi]poly, 4:collection

Synopsis

```
topogeometry CreateTopoGeom(varchar toponame, integer tg_type, integer layer_id, topoelementarray tg_objs);
topogeometry CreateTopoGeom(varchar toponame, integer tg_type, integer layer_id);
```

Beschreibung

Erstellt ein TopoGeometry Objekt für den Layer, der über die layer_id angegeben wird, und registriert es in der Tabelle "relation" in dem Schema toponame.

tg_type ist eine Ganzzahl: 1:[multi]point (punktförmig), 2:[multi]line (geradlinig), 3:[multi]poly (flächenhaft), 4:collection. layer_id ist der Identifikator des Layers in der Tabelle "topology.layer".

Punktförmige Layer werden aus Knoten gebildet, linienförmige Layer aus Kanten, flächige Layer aus Maschen und Kollektionen können aus einer Mischung von Knoten, Kanten und Maschen gebildet werden.

Wird das Feld mit den Komponenten weggelassen, wird ein leeres TopoGeometrie Objekt erstellt.

Verfügbarkeit: 1.?

Beispiele: Aus bestehenden Kanten bilden

Erstellt eine TopoGeometry im Schema "ri_topo" für den Layer 2 (ri_roads), vom Datentyp (2) LINE, für die erste Kante (die wir unter ST_CreateTopoGeo geladen haben).

```
INSERT INTO ri.ri_roads(road_name, topo) VALUES('Unknown', topology.CreateTopoGeom('ri_topo' ←
    ',2,2','{1,2}'::topology.topoelementarray);
```

Beispiele: Eine flächige Geometrie in die vermutete TopoGeometry konvertieren

Angenommen wir wollen eine Geometrie aus einer Kollektion von Maschen bilden. Wir haben zum Beispiel die Tabelle "blockgroups" und wollen die TopoGeometry von jeder "blockgroup" wissen. Falls unsere Daten perfekt ausgerichtet sind, können wir folgendes ausführen:

```
-- erstellt die TopoGeometry Spalte --
SELECT topology.AddTopoGeometryColumn(
    'topo_boston',
    'boston', 'blockgroups', 'topo', 'POLYGON');

-- addtopogeometrycolumn --
1

-- Aktualisierung der Spalte unter der Annahme,
-- dass Alles perfekt an den Kanten ausgerichtet ist
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;

```

```

--die Welt ist selten perfekt und gestattet ein paar Fehler
--berechnet, ob 50% der Masche dorthinein fallen,
--wo wir die Begrenzung unserer "Blockgroup" annehmen
UPDATE boston.blockgroups AS bg
    SET topo = topology.CreateTopoGeom('topo_boston'
        ,3,1
        , foo.bfaces)
FROM (SELECT b.gid, topology.TopoElementArray_Agg(ARRAY[f.face_id,3]) As bfaces
    FROM boston.blockgroups As b
        INNER JOIN topo_boston.face As f ON b.geom && f.mbr
        WHERE ST_Covers(b.geom, topology.ST_GetFaceGeometry('topo_boston', f.face_id))
    OR
    ( ST_Intersects(b.geom, topology.ST_GetFaceGeometry('topo_boston', f.face_id))
        AND ST_Area(ST_Intersection(b.geom, topology.ST_GetFaceGeometry('topo_boston', ↵
        f.face_id) ) ) >
        ST_Area(topology.ST_GetFaceGeometry('topo_boston', f.face_id))*0.5
    )
    GROUP BY b.gid) As foo
WHERE foo.gid = bg.gid;

-- und falls wir die TopoGeometry zurück konvertieren wollen,
-- in eine denormalisierte Geometrie die an unseren Maschen und Kanten ausgerichtet ist,
-- wandeln wir den topologischen Datentyp in eine Geometrie um
-- Das richtig Fetzigke daran ist, dass die neue Geometrie
-- nun an den Mittelachsen der "Tiger"-Strassen ausgerichtet ist
UPDATE boston.blockgroups SET new_geom = topo::geometry;

```

Siehe auch

[AddTopoGeometryColumn](#), [toTopoGeom](#) [ST_CreateTopoGeo](#), [ST_GetFaceGeometry](#), [TopoElementArray](#), [TopoElementArray_Agg](#)

11.8.2 toTopoGeom

toTopoGeom — Wandelt eine einfache Geometrie in eine TopoGeometry um

Synopsis

```

topogeometry toTopoGeom(geometry geom, varchar toponame, integer layer_id, float8 tolerance);
topogeometry toTopoGeom(geometry geom, topogeometry topogeom, float8 tolerance);

```

Beschreibung

Wandelt eine einfache Geometrie in eine [TopoGeometry](#) um.

Die topologischen Elementarstrukturen, die benötigt werden um die Übergabegeometrie darzustellen, werden der zugrunde liegenden Topologie hinzugefügt. Dabei können bestehende Strukturen aufgetrennt werden, die dann mit der ausgegebenen TopoGeometry in der Tabelle `relation` zusammengeführt werden.

Bestehende Objekte einer TopoGeometry (mit der möglichen Ausnahme von `topogeom`, falls angegeben) behalten ihre geometrische Gestalt.

Wenn `tolerance` angegeben ist, wird diese zum Fangen der Eingabegeometrie an bestehenden Elementarstrukturen verwendet.

Bei der ersten Form wird eine neue TopoGeometry für den Layer (`layer_id`) einer Topologie (`toponame`) erstellt.

Bei der zweiten Form werden die aus der Konvertierung entstehenden Elementarstrukturen zu der bestehenden TopoGeometry (`topogeom`) hinzugefügt. Dabei wird möglicherweise zusätzlicher Raum aufgefüllt, um die endgültige geometrische Gestalt zu erreichen. Um die alte geometrische Gestalt zur Gänze durch eine neue zu ersetzen, siehe [clearTopoGeom](#).

Verfügbarkeit: 2.0

Erweiterung: 2.1.0 die Version, welche eine bestehende TopoGeometry entgegennimmt, wurde hinzugefügt.

Beispiele

Dies ist ein in sich selbst vollkommen abgeschlossener Arbeitsablauf

```
-- führen Sie dies bitte aus, wenn Wie noch keine Topologie aufgesetzt haben
-- erstellt eine Topologie, ohne eine Toleranz zu erlauben
SELECT topology.CreateTopology('topo_boston_test', 2249);
-- erstellt eine neue Tabelle
CREATE TABLE nei_topo(gid serial primary key, nei varchar(30));
--fügt eine TopoGeometry-Spalte hinzu
SELECT topology.AddTopoGeometryColumn('topo_boston_test', 'public', 'nei_topo', 'topo', ' ←
MULTIPOLYGON') As new_layer_id;
new_layer_id
-----
1

--verwendet die neue "layer-id" um die neue TopoGeometry-Spalte zu befüllen
-- wir fügen die TopoGeometry mit der Toleranz 0 zu dem neuen Layer hinzu
INSERT INTO nei_topo(nei, topo)
SELECT nei, topology.toTopoGeom(geom, 'topo_boston_test', 1)
FROM neighborhoods
WHERE gid BETWEEN 1 and 15;

--schauen was passiert ist --
SELECT * FROM
    topology.TopologySummary('topo_boston_test');

-- summary--
Topology topo_boston_test (5), SRID 2249, precision 0
61 nodes, 87 edges, 35 faces, 15 topogeoms in 1 layers
Layer 1, type Polygonal (3), 15 topogeoms
Deploy: public.nei_topo.topo

-- Schrumpft alle Polygone der TopoGeometry um 10 Mmeter
UPDATE nei_topo SET topo = ST_Buffer(clearTopoGeom(topo), -10);

-- das Niemandsland erhalten, das durch den oberen Vorgang übriggelassen wurde
-- Ich glaube bei GRASS wird dieses mit "polygon0 layer" bezeichnet
SELECT ST_GetFaceGeometry('topo_boston_test', f.face_id)
FROM topo_boston_test.face f
WHERE f.face_id
> 0 -- don't consider the universe face
AND NOT EXISTS ( -- check that no TopoGeometry references the face
    SELECT * FROM topo_boston_test.relation
    WHERE layer_id = 1 AND element_id = f.face_id
);
```

Siehe auch

[CreateTopology](#), [AddTopoGeometryColumn](#), [CreateTopoGeom](#), [TopologySummary](#), [clearTopoGeom](#)

11.8.3 TopoElementArray_Agg

TopoElementArray_Agg — Gibt für eine Menge an element_id, type Feldern (topoelements) ein topoelementarray zurück

Synopsis

topoelementarray **TopoElementArray_Agg**(topoelement set tefield);

Beschreibung

Verwendet um ein **TopoElementArray** aus einer Menge an **TopoElement** zu erstellen.

Verfügbarkeit: 2.0.0

Beispiele

```
SELECT topology.TopoElementArray_Agg(ARRAY[e,t]) As tea
FROM generate_series(1,3) As e CROSS JOIN generate_series(1,4) As t;
tea
-----
{{1,1},{1,2},{1,3},{1,4},{2,1},{2,2},{2,3},{2,4},{3,1},{3,2},{3,3},{3,4}}
```

Siehe auch

TopoElement, **TopoElementArray**

11.9 TopoGeometry Editoren

11.9.1 clearTopoGeom

clearTopoGeom — Löscht den Inhalt einer TopoGeometry

Synopsis

topogeometry **clearTopoGeom**(topogeometry topogeom);

Beschreibung

Löscht den Inhalt einer **TopoGeometry** und wandelt sie in eine leere um. Am nützlichsten in Verbindung mit **toTopoGeom**, um die geometrische Gestalt bestehende Objekte und alle abhängigen Objekte in höheren hierarchischen Ebenen zu ersetzen.

Verfügbarkeit: 2.1

Beispiele

```
-- Alle Polygone einer TopoGeometry um 10 Meter schrumpfen
UPDATE nei_topo SET topo = ST_Buffer(clearTopoGeom(topo), -10);
```

Siehe auch

toTopoGeom

11.9.2 TopoGeom_addElement

TopoGeom_addElement — Fügt ein Element zu der Definition einer TopoGeometry hinzu

Synopsis

topogeometry **TopoGeom_addElement**(topogeometry tg, topoelement el);

Beschreibung

Fügt ein **TopoElement** zur Definition eines TopoGeometry-Objekts hinzu. Wenn das Element bereits Teil der Definition ist, führt dies zu keinem Fehler.

Verfügbarkeit: 2.3

Beispiele

```
-- Die Kante 5 zu der TopoGeometry "tg" hinzufügen
UPDATE mylayer SET tg = TopoGeom_addElement(tg, '{5,2}');
```

Siehe auch

[TopoGeom_remElement](#), [CreateTopoGeom](#)

11.9.3 TopoGeom_remElement

TopoGeom_remElement — Entfernt ein Element aus der Definition einer TopoGeometry

Synopsis

topogeometry **TopoGeom_remElement**(topogeometry tg, topoelement el);

Beschreibung

Entfernt ein **TopoElement** aus der Ausgestaltung des TopoGeometry Objekts.

Verfügbarkeit: 2.3

Beispiele

```
-- Entfernt Masche 43 aus der TopoGeometry tg
UPDATE mylayer SET tg = TopoGeom_remElement(tg, '{43,3}');
```

Siehe auch

[TopoGeom_addElement](#), [CreateTopoGeom](#)

11.9.4 toTopoGeom

toTopoGeom — Fügt eine Geometrie zu einer bestehenden TopoGeometry hinzu

Beschreibung

Siehe [toTopoGeom](#)

11.10 TopoGeometry Accessors

11.10.1 GetTopoGeomElementArray

`GetTopoGeomElementArray` — Gibt ein `topoelementarray` (ein Feld von `topoelements`) zurück, das die topologischen Elemente und den Datentyp der gegebenen TopoGeometry (die Elementarstrukturen) enthält.

Synopsis

```
topoelementarray GetTopoGeomElementArray(varchar toponame, integer layer_id, integer tg_id);
```

```
topoelementarray topoelement GetTopoGeomElementArray(topogeometry tg);
```

Beschreibung

Gibt ein [TopoElementArray](#) zurück, das die topologischen Elemente und den Datentyp der gegebenen TopoGeometry (die Elementarstrukturen) enthält. Dies ist ähnlich dem `GetTopoGeomElements`, ausser dass die Elemente als Feld statt als Datensatz ausgegeben werden.

`tg_id` steht für die ID des TopoGeometry Objekts der Topologie eines Layers, der durch die `layer_id` der Tabelle "topology.layer" angegeben wird.

Verfügbarkeit: 1.?

Beispiele

Siehe auch

[GetTopoGeomElements](#), [TopoElementArray](#)

11.10.2 GetTopoGeomElements

`GetTopoGeomElements` — Gibt für eine TopoGeometry (Elementarstrukturen) einen Satz an `topoelement` Objekten zurück, welche die topologische `element_id` und den `element_type` beinhalten

Synopsis

```
setof topoelement GetTopoGeomElements(varchar toponame, integer layer_id, integer tg_id);
```

```
setof topoelement GetTopoGeomElements(topogeometry tg);
```

Beschreibung

Gibt für ein TopoGeometry Objekt im Schema `toponame`, eine Menge an `element_id,element_type` (`topoelements`) aus.

`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](#)

11.11 TopoGeometry Ausgabe

11.11.1 AsGML

AsGML — Gibt die GML-Darstellung einer TopoGeometry zurück.

Synopsis

```
text AsGML(topogeometry tg);
text AsGML(topogeometry tg, text nsprefix_in);
text AsGML(topogeometry tg, regclass visitedTable);
text AsGML(topogeometry tg, regclass visitedTable, text nsprefix);
text AsGML(topogeometry tg, text nsprefix_in, integer precision, integer options);
text AsGML(topogeometry tg, text nsprefix_in, integer precision, integer options, regclass visitedTable);
text AsGML(topogeometry tg, text nsprefix_in, integer precision, integer options, regclass visitedTable, text idprefix);
text AsGML(topogeometry tg, text nsprefix_in, integer precision, integer options, regclass visitedTable, text idprefix, int gmlversion);
```

Beschreibung

Gibt die GML-Darstellung einer TopoGeometry im GML3-Format aus. Wenn kein `nsprefix_in` angegeben ist, dann wird `gml` verwendet. Übergeben sie eine leere Zeichenfolge für "nsprefix" um keinen bestimmten Namensraum festzulegen. Wenn die Parameter "precision" (Standardwert: 15) und "options" (Standardwert: 1) angegeben sind, werden diese unangetastet an den zugrunde liegenden Aufruf von `ST_AsGML` übergeben.

The `visitedTable` parameter, if given, is used for keeping track of the visited Node and Edge elements so to use cross-references (`xlink:xref`) rather than duplicating definitions. The table is expected to have (at least) two integer fields: `'element_type'` and `'element_id'`. The calling user must have both read and write privileges on the given table. For best performance, an index should be defined on `element_type` and `element_id`, in that order. Such index would be created automatically by adding a unique constraint to the fields. Example:

```
CREATE TABLE visited (
  element_type integer, element_id integer,
  unique(element_type, element_id)
);
```

Wird der Parameter `idprefix` angegeben, so wird dieser den Identifikatoren der Tags von Kanten und Knoten vorangestellt.

Wird der Parameter `gmlver` angegeben, so wird dieser and das zugrunde liegende `ST_AsGML` übergeben. Standardmäßig wird 3 angenommen.

Verfügbarkeit: 2.0.0

Beispiele

Hier wird die TopoGeometry verwendet, die wir unter [CreateTopoGeom](#) erstellt haben

```

SELECT topology.AsGML(topo) As rdgml
FROM ri.roads
WHERE road_name = 'Unknown';

-- rdgml--
<gml:TopoCurve>
  <gml:directedEdge>
    <gml:Edge gml:id="E1">
      <gml:directedNode orientation="-">
        <gml:Node gml:id="N1"/>
      </gml:directedNode>
      <gml:directedNode>
    </gml:directedNode>
    <gml:curveProperty>
      <gml:Curve srsName="urn:ogc:def:crs:EPSG::3438">
        <gml:segments>
          <gml:LineStringSegment>
            <gml:posList srsDimension="2"
>384744 236928 384750 236923 384769 236911 384799 236895 384811 236890
          384833 236884 384844 236882 384866 236881 384879 236883 384954 ←
          236898 385087 236932 385117 236938
          385167 236938 385203 236941 385224 236946 385233 236950 385241 ←
          236956 385254 236971
          385260 236979 385268 236999 385273 237018 385273 237037 385271 ←
          237047 385267 237057 385225 237125
          385210 237144 385192 237161 385167 237192 385162 237202 385159 ←
          237214 385159 237227 385162 237241
          385166 237256 385196 237324 385209 237345 385234 237375 385237 ←
          237383 385238 237399 385236 237407
          385227 237419 385213 237430 385193 237439 385174 237451 385170 ←
          237455 385169 237460 385171 237475
          385181 237503 385190 237521 385200 237533 385206 237538 385213 ←
          237541 385221 237542 385235 237540 385242 237541
          385249 237544 385260 237555 385270 237570 385289 237584 385292 ←
          237589 385291 237596 385284 237630</gml:posList>
            </gml:LineStringSegment>
          </gml:segments>
        </gml:Curve>
      </gml:curveProperty>
    </gml:Edge>
  </gml:directedEdge>
</gml:TopoCurve>
>

```

Selbes Beispiel wie das Vorige, aber ohne Namensraum

```

SELECT topology.AsGML(topo, '') As rdgml
FROM ri.roads
WHERE road_name = 'Unknown';

-- rdgml--
<TopoCurve>
  <directedEdge>
    <Edge id="E1">
      <directedNode orientation="-">
        <Node id="N1"/>
      </directedNode>
      <directedNode>
    </directedNode>
    <curveProperty>
      <Curve srsName="urn:ogc:def:crs:EPSG::3438">

```

```

        <segments>
          <LineStringSegment>
            <posList srsDimension="2"
>384744 236928 384750 236923 384769 236911 384799 236895 384811 236890
              384833 236884 384844 236882 384866 236881 384879 236883 384954 ←
              236898 385087 236932 385117 236938
              385167 236938 385203 236941 385224 236946 385233 236950 385241 ←
              236956 385254 236971
              385260 236979 385268 236999 385273 237018 385273 237037 385271 ←
              237047 385267 237057 385225 237125
              385210 237144 385192 237161 385167 237192 385162 237202 385159 ←
              237214 385159 237227 385162 237241
              385166 237256 385196 237324 385209 237345 385234 237375 385237 ←
              237383 385238 237399 385236 237407
              385227 237419 385213 237430 385193 237439 385174 237451 385170 ←
              237455 385169 237460 385171 237475
              385181 237503 385190 237521 385200 237533 385206 237538 385213 ←
              237541 385221 237542 385235 237540 385242 237541
              385249 237544 385260 237555 385270 237570 385289 237584 385292 ←
              237589 385291 237596 385284 237630</posList>
          </LineStringSegment>
        </segments>
      </Curve>
    </curveProperty>
  </Edge>
</directedEdge>
</TopoCurve>
>

```

Siehe auch

[CreateTopoGeom](#), [ST_CreateTopoGeo](#)

11.11.2 AsTopoJSON

AsTopoJSON — Gibt die TopoJSON-Darstellung einer TopoGeometry zurück.

Synopsis

```
text AsTopoJSON(topogeometry tg, regclass edgeMapTable);
```

Beschreibung

Gibt eine TopoGeometry in der TopoJSON-Darstellung zurück. Wenn `edgeMapTable` nicht NULL ist, wird diese als Lookup/Speicher für die Abbildung der Identifikatoren der Kanten auf die Indizes der Kreisbögen verwendet. Dadurch wird ein kompaktes Feld "arcs" im endgültigen Dokument ermöglicht.

Wenn die Tabelle angegeben ist, wird die Existenz der Attribute "arc_id" vom Datentyp "serial" und "edge_id" vom Typ "integer" vorausgesetzt; da der Code die Tabelle nach der "edge_id" abfragt, sollte ein Index für dieses Attribut erstellt werden.



Note

Die Kreisbögen in der TopoJSON Ausgabe sind von 0 weg indiziert, während sie in der Tabelle "edgeMapTable" 1-basiert sind.

Ein vollständiges TopoJson Dokument benötigt zusätzlich zu den von dieser Funktion ausgegebenen Schnipseln, die tatsächlichen Bögen und einige Header. Siehe [TopoJSON specification](#).

Verfügbarkeit: 2.1.0

Erweiterung: 2.2.1 Unterstützung für punktförmige Eingabewerte hinzugefügt

Siehe auch

[ST_AsGeoJSON](#)

Beispiele

```
CREATE TEMP TABLE edgemap(arc_id serial, edge_id int unique);

-- Header
SELECT '{ "type": "Topology", "transform": { "scale": [1,1], "translate": [0,0] }, "objects" ←
      ": {'

-- Objekte
UNION ALL SELECT ''' || feature_name || '": ' || AsTopoJSON(feature, 'edgemap')
FROM features.big_parcel_s WHERE feature_name = 'P3P4';

-- Bögen
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_dumpoints(geom)).* FROM edges
), compare AS (
  SELECT p2.arc_id,
    CASE WHEN p1.path IS NULL THEN p2.geom
    ELSE ST_Translate(p2.geom, -ST_X(p1.geom), -ST_Y(p1.geom))
    END AS geom
  FROM points p2 LEFT OUTER JOIN points p1
  ON ( p1.arc_id = p2.arc_id AND p2.path[1] = p1.path[1]+1 )
  ORDER BY arc_id, p2.path
), arcdump AS (
  SELECT arc_id, (regexp_matches( ST_AsGeoJSON(geom), '\[.*\]'))[1] as t
  FROM compare
), arcs AS (
  SELECT arc_id, '[' || array_to_string(array_agg(t), ',') || ']' as a FROM arcdump
  GROUP BY arc_id
  ORDER BY arc_id
)
SELECT '}', "arcs": [' UNION ALL
SELECT array_to_string(array_agg(a), E',\n') from arcs

-- Footer
UNION ALL SELECT ']}'::text as t;

-- Result:
{ "type": "Topology", "transform": { "scale": [1,1], "translate": [0,0] }, "objects": {
"P3P4": { "type": "MultiPolygon", "arcs": [[[-1]], [[6,5,-5,-4,-3,1]]]
}, "arcs": [
  [[25,30],[6,0],[0,10],[-14,0],[0,-10],[8,0]],
  [[35,6],[0,8]],
  [[35,6],[12,0]],
  [[47,6],[0,8]],
  [[47,14],[0,8]],
```

```
[ [35, 22], [12, 0] ],  
[ [35, 14], [0, 8] ]  
] }
```

11.12 Räumliche Beziehungen einer Topologie

11.12.1 Equals

Equals — Gibt TRUE zurück, wenn zwei TopoGeometry Objekte aus denselben topologischen Elementarstrukturen bestehen.

Synopsis

boolean **Equals**(topogeometry tg1, topogeometry tg2);

Beschreibung

Gibt TRUE zurück, wenn zwei TopoGeometry Objekte aus denselben topologischen Elementarstrukturen: Maschen, Kanten, Knoten, bestehen.



Note

Diese Funktion unterstützt keine TopoGeometry aus einer Sammelgeometrie. Es kann auch keine TopoGeometry Objekte unterschiedlicher Topologien vergleichen

Verfügbarkeit: 1.1.0



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

Beispiele

Siehe auch

[GetTopoGeomElements](#), [ST_Equals](#)

11.12.2 Intersects

Intersects — Gibt TRUE zurück, wenn sich kein beliebiges Paar von Elementarstrukturen zweier TopoGeometry Objekte überschneidet.

Synopsis

boolean **Intersects**(topogeometry tg1, topogeometry tg2);

Beschreibung

Gibt TRUE zurück, wenn sich kein beliebiges Paar von Elementarstrukturen zweier TopoGeometry Objekte überschneidet.



Note

Diese Funktion unterstützt keine TopoGeometry aus einer Sammelgeometrie. Es kann auch keine TopoGeometry Objekte unterschiedlicher Topologien vergleichen. Eine hierarchische TopoGeometry (eine TopoGeometry die sich aus anderen TopoGeometry Objekten zusammensetzt) wird zur Zeit ebenfalls nicht unterstützt.

Verfügbarkeit: 1.1.0



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

Beispiele

Siehe auch

[ST_Intersects](#)

Chapter 12

Adressennormierer

This is a fork of the [PAGC standardizer](#) (original code for this portion was [PAGC PostgreSQL Address Standardizer](#)).

The address standardizer is a single line address parser that takes an input address and normalizes it based on a set of rules stored in a table and helper lex and gaz tables.

The code is built into a single postgresql extension library called `address_standardizer` which can be installed with `CREATE EXTENSION address_standardizer;`. In addition to the `address_standardizer` extension, a sample data extension called `address_standardizer_data_us` extensions is built, which contains gaz, lex, and rules tables for US data. This extensions can be installed via: `CREATE EXTENSION address_standardizer_data_us;`

The code for this extension can be found in the PostGIS `extensions/address_standardizer` and is currently self-contained.

Für eine Installationsanleitung siehe: [Section 2.7](#).

12.1 Funktionsweise des Parsers

The parser works from right to left looking first at the macro elements for postcode, state/province, city, and then looks micro elements to determine if we are dealing with a house number street or intersection or landmark. It currently does not look for a country code or name, but that could be introduced in the future.

Country code Assumed to be US or CA based on: postcode as US or Canada state/province as US or Canada else US

Postcode/zipcode These are recognized using Perl compatible regular expressions. These regexs are currently in the `parseaddress-api.c` and are relatively simple to make changes to if needed.

State/province These are recognized using Perl compatible regular expressions. These regexs are currently in the `parseaddress-api.c` but could get moved into includes in the future for easier maintenance.

12.2 Adressennormierer Datentypen

12.2.1 stdaddr

`stdaddr` — Ein zusammengesetzter Datentyp, der aus den Elementen einer Adresse besteht. Dies ist der zurückgegebene Datentyp der `standardize_address` Funktion.

Beschreibung

A composite type that consists of elements of an address. This is the return type for `standardize_address` function. Some descriptions for elements are borrowed from [PAGC Postal Attributes](#).

Die Token-Nummern geben die Referenznummer der Ausgabe in der [rules Tabelle](#) an.



This method needs `address_standardizer` extension.

building ist ein Text (Token-Nummer 0): Verweist auf die Hausnummer oder Namen. Gebäude Identifikatoren und Typen nicht geparkt. Bei den meisten Adressen üblicherweise leer.

house_num ist ein Text (Token-Nummer 1): Die Hausnummer einer Straße. Beispiel 75 in 75 State Street.

predir ist ein Text (Token-Nummer 2): STREET NAME PRE-DIRECTIONAL, wie Nord, Süd, Ost, West etc.

qual ist ein Text (Token-Nummer 3): STREET NAME PRE-MODIFIER Beispiel *OLD* in 3715 OLD HIGHWAY 99.

pretype ist ein Text (Token-Nummer 4): STREET PREFIX TYPE

name ist ein Text (Token-Nummer 5): STREET NAME

suftype ist ein Text (Token-Nummer 6): STREET POST TYPE z.B. St, Ave, Cir. Ein dem Straßennamen angehängter Straßentyp. Beispiel *STREET* in 75 State Street.

sufdir ist ein Text (Token-Nummer 7): STREET POST-DIRECTIONAL Eine Richtungsangabe, die dem Straßennamen folg. Beispiel *WEST* in 3715 TENTH AVENUE WEST.

ruralroute ist text (token number 8): RURAL ROUTE . Example 7 in RR 7.

extra ist ein Text: Zusätzliche Information, wie die Geschossnummer/Stockwerk.

city ist ein Text (Token-Nummer 10): Beispiel Boston.

state ist ein Text (Token-Nummer 11): Beispiel MASSACHUSETTS

country ist ein Text (Token-Nummer 12): Beispiel USA

postcode ist ein Text POSTAL CODE (ZIP CODE) (Token-Nummer 13): Beispiel 02109

box ist ein Text POSTAL BOX NUMBER (Token-Nummer 14 und 15): Beispiel 02109

unit ist ein Text Wohnungs- oder Suite-Nummer (Token-Nummer 17): Beispiel *3B* in APT 3B.

12.3 Adressennormierer Tabellen

12.3.1 rules Tabelle

rules Tabelle — The rules table contains a set of rules that maps address input sequence tokens to standardized output sequence. A rule is defined as a set of input tokens followed by -1 (terminator) followed by set of output tokens followed by -1 followed by number denoting kind of rule followed by ranking of rule.

Beschreibung

A rules table must have at least the following columns, though you are allowed to add more for your own uses.

id Der Primärschlüssel der Tabelle

rule text field denoting the rule. Details at [PAGC Address Standardizer Rule records](#).

A rule consists of a set of non-negative integers representing input tokens, terminated by a -1, followed by an equal number of non-negative integers representing postal attributes, terminated by a -1, followed by an integer representing a rule type, followed by an integer representing the rank of the rule. The rules are ranked from 0 (lowest) to 17 (highest).

So for example the rule 2 0 2 22 3 -1 5 5 6 7 3 -1 2 6 maps to sequence of output tokens *TYPE NUMBER TYPE DIRECT QUALIF* to the output sequence *STREET STREET SUFTYP SUFDIR QUALIF*. The rule is an ARC_C rule of rank 6.

Numbers for corresponding output tokens are listed in [stdaddr](#).

Eingabe-Token

Each rule starts with a set of input tokens followed by a terminator -1. Valid input tokens excerpted from [PAGC Input Tokens](#) are as follows:

Form-Based Input Tokens

AMPERS (13). Das kaufmännische Und (&) wird häufig zur Abkürzung des Wortes "und" verwendet.

DASH (9). Ein Satzzeichen.

DOUBLE (21). Eine Sequenz mit zwei Buchstaben. Wird oft als Identifikator verwendet.

FRACT (25). Brüche kommen manchmal bei Hausnummern oder Blocknummern vor.

MIXED (23). Eine alphanumerische Zeichenkette, die aus Buchstaben und Ziffern besteht. Wird als Identifikator verwendet.

NUMBER (0). Eine Folge von Ziffern.

ORD (15). Bezeichnungen wie "First" oder 1st. Wird häufig bei Straßennamen benutzt.

ORD (18). Ein einzelner Buchstabe.

WORD (1). Ein Wort ist eine Zeichenfolge beliebiger Länge. Ein einzelnes Zeichen kann sowohl ein **SINGLE** als auch ein **WORD** sein.

Function-based Input Tokens

BOXH (14). Ein Text zur Kennzeichnung von Postfächern. Zum Beispiel *Box* oder *PO Box*.

BUILDH (19). Words used to denote buildings or building complexes, usually as a prefix. For example: *Tower* in *Tower 7A*.

BUILD (24). Words and abbreviations used to denote buildings or building complexes, usually as a suffix. For example: *Shopping Centre*.

DIRECT (22). Text zur Richtungsangabe, zum Beispiel *North*.

MILE (20). Words used to denote milepost addresses.

ROAD (6). Words and abbreviations used to denote highways and roads. For example: the *Interstate* in *Interstate 5*

RR (8). Words and abbreviations used to denote rural routes. *RR*.

TYPE (2). Words and abbreviation used to denote street types. For example: *ST* or *AVE*.

UNIT (16). Words and abbreviation used to denote internal subaddresses. For example, *APT* or *UNIT*.

Postal Type Input Tokens

QUINT (28). Eine 5-stellige Nummer. Gibt den Zip Code an

QUAD (29). Eine 4-stellige Nummer. Gibt den ZIP4 Code an.

PCH (27). A 3 character sequence of letter number letter. Identifies an FSA, the first 3 characters of a Canadian postal code.

PCT (26). A 3 character sequence of number letter number. Identifies an LDU, the last 3 characters of a Canadian postal code.

Stopwords

STOPWORDS combine with WORDS. In rules a string of multiple WORDs and STOPWORDS will be represented by a single WORD token.

STOPWORD (7). A word with low lexical significance, that can be omitted in parsing. For example: *THE*.

Ausgabe-Token

After the first -1 (terminator), follows the output tokens and their order, followed by a terminator -1. Numbers for corresponding output tokens are listed in [stdaddr](#). What are allowed is dependent on kind of rule. Output tokens valid for each rule type are listed in the section called "[Regel Typen und Rang](#)".

Regel Typen und Rang

The final part of the rule is the rule type which is denoted by one of the following, followed by a rule rank. The rules are ranked from 0 (lowest) to 17 (highest).

MACRO_C

(Token-Nummer = "0"). Die Klassenregeln um MACRO Klauseln, wie *PLACE STATE ZIP*, zu parsen.

MACRO_C output tokens (excerpted from <http://www.pagegeo.org/docs/html/pagc-12.html#--r-typ-->).

CITY (Token-Nummer "10"). Beispiel "Albanien"

STATE (Token-Nummer "11"). Beispiel "NY"

NATION (token number "12"). This attribute is not used in most reference files. Example "USA"

POSTAL (token number "13"). (SADS elements "ZIP CODE" , "PLUS 4"). This attribute is used for both the US Zip and the Canadian Postal Codes.

MICRO_C

(token number = "1"). The class of rules for parsing full MICRO clauses (such as House, street, sufdir, predir, pretyp, suftype, qualif) (ie ARC_C plus CIVIC_C). These rules are not used in the build phase.

MICRO_C output tokens (excerpted from <http://www.pagegeo.org/docs/html/pagc-12.html#--r-typ-->).

HOUSE ist ein Text (Token-Nummer 1): Die Hausnummer einer Straße. Beispiel 75 in 75 State Street.

predir ist ein Text (Token-Nummer 2): STREET NAME PRE-DIRECTIONAL, wie Nord, Süd, Ost, West etc.

qual ist ein Text (Token-Nummer 3): STREET NAME PRE-MODIFIER Beispiel *OLD* in 3715 OLD HIGHWAY 99.

pretype ist ein Text (Token-Nummer 4): STREET PREFIX TYPE

street ist ein Text (Token-Nummer 5): STREET NAME

suftype ist ein Text (Token-Nummer 6): STREET POST TYPE z.B. St, Ave, Cir. Ein dem Straßennamen angehängter Straßentyp. Beispiel *STREET* in 75 State Street.

sufdir ist ein Text (Token-Nummer 7): STREET POST-DIRECTIONAL Eine Richtungsangabe, die dem Straßennamen folgt.
Beispiel *WEST* in 3715 TENTH AVENUE WEST.

ARC_C

(token number = "2"). The class of rules for parsing MICRO clauses, excluding the HOUSE attribute. As such uses same set of output tokens as MICRO_C minus the HOUSE token.

CIVIC_C

(Token-Nummer = "3"). Die Klassenregeln zum parsen des HOUSE Attributs.

EXTRA_C

(token number = "4"). The class of rules for parsing EXTRA attributes - attributes excluded from geocoding. These rules are not used in the build phase.

EXTRA_C output tokens (excerpted from <http://www.pgcgeo.org/docs/html/pagc-12.html#--r-typ-->).

BLDNG (token number 0): Unparsed building identifiers and types.

BOXH (token number 14): The **BOX** in BOX 3B

BOXT (Token-Nummer 15): **3B** in BOX 3B

RR (Token-Nummer 8): **RR** in RR 7

UNITH (Token-Nummer 16): **APT** in APT 3B

UNITT (Token-Nummer 17): **3B** in APT 3B

UNKNWN (Token-Nummer 9): Eine nicht näher klassifizierte Ausgabe.

12.3.2 lex Tabelle

lex Tabelle — A lex table is used to classify alphanumeric input and associate that input with (a) input tokens (See the section called “**Eingabe-Token**”) and (b) standardized representations.

Beschreibung

Eine lex (abgekürzt für Lexikon) Tabelle wird verwendet um alphanumerische Eingaben zu gliedern, und die Eingabe mit the section called “**Eingabe-Token**” und (b) genormten Darstellungen zu verbinden. In diesen Tabellen finden Sie Dinge wie ONE abgebildet auf stdword: 1.

A lex has at least the following columns in the table. You may add

id Der Primärschlüssel der Tabelle

seq integer: definition number?

word text: das Eingabewort

stdword text: das normierte Ersatzwort

token integer: the kind of word it is. Only if it is used in this context will it be replaced. Refer to **PAGC Tokens**.

12.3.3 gaz Tabelle

gaz Tabelle — A gaz table is used to standardize place names and associate that input with (a) input tokens (See the section called “**Eingabe-Token**”) and (b) standardized representations.

Beschreibung

A gaz (short for gazeteer) table is used to standardize place names and associate that input with the section called “**Eingabe-Token**” and (b) standardized representations. For example if you are in US, you may load these with State Names and associated abbreviations.

A gaz table has at least the following columns in the table. You may add more columns if you wish for your own purposes.

id Der Primärschlüssel der Tabelle

seq integer: definition number? - identifier used for that instance of the word

word text: das Eingabewort

stdword text: das normierte Ersatzwort

token integer: the kind of word it is. Only if it is used in this context will it be replaced. Refer to [PAGC Tokens](#).

12.4 Adressennormierer Funktionen

12.4.1 parse_address

parse_address — Takes a 1 line address and breaks into parts

Synopsis

record **parse_address**(text address);

Beschreibung

Returns takes an address as input, and returns a record output consisting of fields *num*, *street*, *street2*, *address1*, *city*, *state*, *zip*, *zipplus*, *country*.

Verfügbarkeit: 2.2.0



This method needs address_standardizer extension.

Beispiele

Einzelne Adresse

```
SELECT num, street, city, zip, zipplus
       FROM parse_address('1 Devonshire Place, Boston, MA 02109-1234') AS a;
```

num	street	city	zip	zipplus
1	Devonshire Place	Boston	02109	1234

Tabelle mit Adressen

```
-- basic table
CREATE TABLE places(addid serial PRIMARY KEY, address text);

INSERT INTO places(address)
VALUES ('529 Main Street, Boston MA, 02129'),
      ('77 Massachusetts Avenue, Cambridge, MA 02139'),
```

```

('25 Wizard of Oz, Walaford, KS 99912323'),
('26 Capen Street, Medford, MA'),
('124 Mount Auburn St, Cambridge, Massachusetts 02138'),
('950 Main Street, Worcester, MA 01610');

-- parse the addresses
-- if you want all fields you can use (a).*
SELECT addid, (a).num, (a).street, (a).city, (a).state, (a).zip, (a).zipplus
FROM (SELECT addid, parse_address(address) As a
FROM places) AS p;

```

addid	num	street	city	state	zip	zipplus
1	529	Main Street	Boston	MA	02129	
2	77	Massachusetts Avenue	Cambridge	MA	02139	
3	25	Wizard of Oz	Walaford	KS	99912	323
4	26	Capen Street	Medford	MA		
5	124	Mount Auburn St	Cambridge	MA	02138	
6	950	Main Street	Worcester	MA	01610	

(6 rows)

Siehe auch

12.4.2 standardize_address

`standardize_address` — Returns an `stdaddr` form of an input address utilizing `lex`, `gaz`, and `rule` tables.

Synopsis

```

stdaddr standardize_address(text lextab, text gaztab, text rultab, text address);
stdaddr standardize_address(text lextab, text gaztab, text rultab, text micro, text macro);

```

Beschreibung

Returns an **stdaddr** form of an input address utilizing **lex Tabelle** table name, **gaz Tabelle**, and **rules Tabelle** table names and an address.

Variant 1: Takes an address as a single line.

Variant 2: Takes an address as 2 parts. A `micro` consisting of standard first line of postal address e.g. `house_num street`, and a `macro` consisting of standard postal second line of an address e.g. `city, state postal_code country`.

Verfügbarkeit: 2.2.0



This method needs `address_standardizer` extension.

Beispiele

Verwendung der `address_standardizer_data_us` Erweiterung

```
CREATE EXTENSION address_standardizer_data_us; -- muss nur einmal vollzogen werden
```

Variant 1: Single line address. This doesn't work well with non-US addresses

```

SELECT house_num, name, suftype, city, country, state, unit FROM standardize_address(' ←
us_lex',
                                     'us_gaz', 'us_rules', 'One Devonshire Place, PH 301, Boston, MA ←
                                     02109');

```

house_num	name	suftype	city	country	state	unit
1	DEVONSHIRE	PLACE	BOSTON	USA	MASSACHUSETTS	# PENTHOUSE 301

Using tables packaged with tiger geocoder. This example only works if you installed `postgis_tiger_geocoder`.

```
SELECT * FROM standardize_address('tiger.pagc_lex',
    'tiger.pagc_gaz', 'tiger.pagc_rules', 'One Devonshire Place, PH 301, Boston, MA 02109-1234');
```

Make easier to read we'll dump output using `hstore` extension `CREATE EXTENSION hstore`; you need to install

```
SELECT (each(hstore(p))).*
FROM standardize_address('tiger.pagc_lex', 'tiger.pagc_gaz',
    'tiger.pagc_rules', 'One Devonshire Place, PH 301, Boston, MA 02109') As p;
```

key	value
box	
city	BOSTON
name	DEVONSHIRE
qual	
unit	# PENTHOUSE 301
extra	
state	MA
predir	
sufdir	
country	USA
pretype	
suftype	PL
building	
postcode	02109
house_num	1
ruralroute	

(16 rows)

Variant 2: As a two part Address

```
SELECT (each(hstore(p))).*
FROM standardize_address('tiger.pagc_lex', 'tiger.pagc_gaz',
    'tiger.pagc_rules', 'One Devonshire Place, PH 301', 'Boston, MA 02109, US') As p;
```

key	value
box	
city	BOSTON
name	DEVONSHIRE
qual	
unit	# PENTHOUSE 301
extra	
state	MA
predir	
sufdir	
country	USA
pretype	
suftype	PL
building	
postcode	02109
house_num	1
ruralroute	

(16 rows)

Siehe auch

[stdaddr](#), [rules Tabelle](#), [lex Tabelle](#), [gaz Tabelle](#), [Pgac_Normalize_Address](#)

Chapter 13

PostGIS Extras

Dieses Kapitel beschreibt Features die sich in den zusätzlichen Verzeichnissen des PostGIS Quellcodes - Tarballs und Repository - befinden. Diese sind nicht immer mit den binären PostGIS Veröffentlichung/Release paketierte, sind aber im allgemeinen plpgsql basiert oder es handelt sich um Shell Skripts die direkt aufgerufen werden können.

13.1 Tiger Geocoder

Es existieren eine Reihe weiterer Open Source Geokodierer für PostGIS, welche im Gegensatz zu dem Tiger Geocoder den Vorteil haben, dass sie mehrere Länder unterstützen

- **Nominatim** uses OpenStreetMap gazeteer formatted data. It requires osm2pgsql for loading the data, PostgreSQL 8.4+ and PostGIS 1.5+ to function. It is packaged as a webservice interface and seems designed to be called as a webservice. Just like the tiger geocoder, it has both a geocoder and a reverse geocoder component. From the documentation, it is unclear if it has a pure SQL interface like the tiger geocoder, or if a good deal of the logic is implemented in the web interface.
- **GIS Graphy** also utilizes PostGIS and like Nominatim works with OpenStreetMap (OSM) data. It comes with a loader to load OSM data and similar to Nominatim is capable of geocoding not just US. Much like Nominatim, it runs as a webservice and relies on Java 1.5, Servlet apps, Solr. GisGraphy is cross-platform and also has a reverse geocoder among some other neat features.

13.1.1 Drop_Indexes_Generate_Script

`Drop_Indexes_Generate_Script` — Erzeugt ein Skript, welches alle Indizes aus dem Datenbankschema "Tiger" oder aus einem vom Anwender angegebenen Schema löscht, wenn die Indizes nicht auf den Primärschlüssel gelegt und nicht "unique" sind. Wenn kein Schema angegeben ist wird standardmäßig auf das `tiger_data` Schema zugegriffen.

Synopsis

```
text Drop_Indexes_Generate_Script(text param_schema=tiger_data);
```

Beschreibung

Erzeugt ein Skript, welches alle Indizes aus dem Datenbankschema "Tiger" oder aus einem vom Anwender angegebenen Schema löscht, wenn die Indizes nicht auf den Primärschlüssel gelegt und nicht "unique" sind. Wenn kein Schema angegeben ist wird standardmäßig auf das `tiger_data` Schema zugegriffen.

This is useful for minimizing index bloat that may confuse the query planner or take up unnecessary space. Use in combination with **Install_Missing_Indexes** to add just the indexes used by the geocoder.

Verfügbarkeit: 2.0.0

Beispiele

```
SELECT drop_indexes_generate_script() As actionsql;
actionsql
-----
DROP INDEX tiger.idx_tiger_countysub_lookup_lower_name;
DROP INDEX tiger.idx_tiger_edges_countyfp;
DROP INDEX tiger.idx_tiger_faces_countyfp;
DROP INDEX tiger.tiger_place_the_geom_gist;
DROP INDEX tiger.tiger_edges_the_geom_gist;
DROP INDEX tiger.tiger_state_the_geom_gist;
DROP INDEX tiger.idx_tiger_addr_least_address;
DROP INDEX tiger.idx_tiger_addr_tlid;
DROP INDEX tiger.idx_tiger_addr_zip;
DROP INDEX tiger.idx_tiger_county_countyfp;
DROP INDEX tiger.idx_tiger_county_lookup_lower_name;
DROP INDEX tiger.idx_tiger_county_lookup_snd_name;
DROP INDEX tiger.idx_tiger_county_lower_name;
DROP INDEX tiger.idx_tiger_county_snd_name;
DROP INDEX tiger.idx_tiger_county_the_geom_gist;
DROP INDEX tiger.idx_tiger_countysub_lookup_snd_name;
DROP INDEX tiger.idx_tiger_cousub_countyfp;
DROP INDEX tiger.idx_tiger_cousub_cousubfp;
DROP INDEX tiger.idx_tiger_cousub_lower_name;
DROP INDEX tiger.idx_tiger_cousub_snd_name;
DROP INDEX tiger.idx_tiger_cousub_the_geom_gist;
DROP INDEX tiger_data.idx_tiger_data_ma_addr_least_address;
DROP INDEX tiger_data.idx_tiger_data_ma_addr_tlid;
DROP INDEX tiger_data.idx_tiger_data_ma_addr_zip;
DROP INDEX tiger_data.idx_tiger_data_ma_county_countyfp;
DROP INDEX tiger_data.idx_tiger_data_ma_county_lookup_lower_name;
DROP INDEX tiger_data.idx_tiger_data_ma_county_lookup_snd_name;
DROP INDEX tiger_data.idx_tiger_data_ma_county_lower_name;
DROP INDEX tiger_data.idx_tiger_data_ma_county_snd_name;
:
:
```

Siehe auch

[Install_Missing_Indexes](#), [Missing_Indexes_Generate_Script](#)

13.1.2 Drop_Nation_Tables_Generate_Script

Drop_Nation_Tables_Generate_Script — Erzeugt ein Skript, welches alle Tabellen in dem angegebenen Schema löscht, die mit `county_all`, `state_all` oder dem Ländercode gefolgt von `county` oder `state` beginnen.

Synopsis

```
text Drop_Nation_Tables_Generate_Script(text param_schema=tiger_data);
```

Beschreibung

Erzeugt ein Skript, welches alle Tabellen in dem angegebenen Schema löscht, die mit `county_all`, `state_all` oder dem Ländercode gefolgt von `county` oder `state` beginnen. Dies ist dann notwendig, wenn Sie von `tiger_2010` auf `tiger_2011` Daten upgraden.

Verfügbarkeit: 2.1.0

Beispiele

```
SELECT drop_nation_tables_generate_script();
DROP TABLE tiger_data.county_all;
DROP TABLE tiger_data.county_all_lookup;
DROP TABLE tiger_data.state_all;
DROP TABLE tiger_data.ma_county;
DROP TABLE tiger_data.ma_state;
```

Siehe auch

[Loader_Generate_Nation_Script](#)

13.1.3 Drop_State_Tables_Generate_Script

Drop_State_Tables_Generate_Script — Erzeugt ein Skript, das alle Tabellen in dem angegebenen Schema löscht, die als Präfix einen Ländercode haben. Wenn kein Schema angegeben ist wird standardmäßig auf das `tiger_data` Schema zugegriffen.

Synopsis

text **Drop_State_Tables_Generate_Script**(text param_state, text param_schema=tiger_data);

Beschreibung

Erzeugt ein Skript, das alle Tabellen in dem angegebenen Schema löscht, die als Präfix einen Ländercode haben. Wenn kein Schema angegeben ist wird standardmäßig auf das `tiger_data` Schema zugegriffen. Wenn beim Import etwas schiefgegangen ist, können mit dieser Funktion die Tabellen eines Staates unmittelbar vor dem erneuten Import, gelöscht werden.

Verfügbarkeit: 2.0.0

Beispiele

```
SELECT drop_state_tables_generate_script('PA');
DROP TABLE tiger_data.pa_addr;
DROP TABLE tiger_data.pa_county;
DROP TABLE tiger_data.pa_county_lookup;
DROP TABLE tiger_data.pa_cousub;
DROP TABLE tiger_data.pa_edges;
DROP TABLE tiger_data.pa_faces;
DROP TABLE tiger_data.pa_featnames;
DROP TABLE tiger_data.pa_place;
DROP TABLE tiger_data.pa_state;
DROP TABLE tiger_data.pa_zip_lookup_base;
DROP TABLE tiger_data.pa_zip_state;
DROP TABLE tiger_data.pa_zip_state_loc;
```

Siehe auch

[Loader_Generate_Script](#)

13.1.4 Geocode

Geocode — Takes in an address as a string (or other normalized address) and outputs a set of possible locations which include a point geometry in NAD 83 long lat, a normalized address for each, and the rating. The lower the rating the more likely the match. Results are sorted by lowest rating first. Can optionally pass in maximum results, defaults to 10, and restrict_region (defaults to NULL)

Synopsis

setof record **geocode**(varchar address, integer max_results=10, geometry restrict_region=NULL, norm_addy OUT addy, geometry OUT geomout, integer OUT rating);

setof record **geocode**(norm_addy in_addy, integer max_results=10, geometry restrict_region=NULL, norm_addy OUT addy, geometry OUT geomout, integer OUT rating);

Beschreibung

Takes in an address as a string (or already normalized address) and outputs a set of possible locations which include a point geometry in NAD 83 long lat, a `normalized_address` (addy) for each, and the rating. The lower the rating the more likely the match. Results are sorted by lowest rating first. Uses Tiger data (edges,faces,addr), PostgreSQL fuzzy string matching (soundex,levenshtein) and PostGIS line interpolation functions to interpolate address along the Tiger edges. The higher the rating the less likely the geocode is right. The geocoded point is defaulted to offset 10 meters from center-line off to side (L/R) of street address is located on.

Enhanced: 2.0.0 to support Tiger 2010 structured data and revised some logic to improve speed, accuracy of geocoding, and to offset point from centerline to side of street address is located on. The new parameter `max_results` useful for specifying number of best results or just returning the best result.

Beispiele: Grundlagen

The below examples timings are on a 3.0 GHZ single processor Windows 7 machine with 2GB ram running PostgreSQL 9.1rc1/PostGIS 2.0 loaded with all of MA,MN,CA, RI state Tiger data loaded.

Genaue Übereinstimmungen haben eine kürzere Rechenzeit (61ms)

```
SELECT g.rating, ST_X(g.geomout) As lon, ST_Y(g.geomout) As lat,
       (addy).address As stno, (addy).streetname As street,
       (addy).streettypeabbrev As styp, (addy).location As city, (addy).stateabbrev As st, ( ←
       addy).zip
FROM geocode('75 State Street, Boston MA 02109') As g;
rating |          lon          |          lat          | stno | street | styp | city | st | zip
-----+-----+-----+-----+-----+-----+-----+-----+-----+----- ←
      0 | -71.0556722990239 | 42.3589914927049 | 75 | State | St | Boston | MA | 02109
```

Sogar wenn der Zip-Code nicht übergeben wird, kann ihn der Geokodierer erraten (dauerte ca. 122-150ms)

```
SELECT g.rating, ST_AsText(ST_SnapToGrid(g.geomout,0.00001)) As wktmlonlat,
       (addy).address As stno, (addy).streetname As street,
       (addy).streettypeabbrev As styp, (addy).location As city, (addy).stateabbrev As st, ( ←
       addy).zip
FROM geocode('226 Hanover Street, Boston, MA',1) As g;
rating |          wktmlonlat          | stno | street | styp | city | st | zip
-----+-----+-----+-----+-----+-----+-----+-----+----- ←
      1 | POINT(-71.05528 42.36316) | 226 | Hanover | St | Boston | MA | 02113
```

Can handle misspellings and provides more than one possible solution with ratings and takes longer (500ms).

```

SELECT g.rating, ST_AsText(ST_SnapToGrid(g.geomout,0.00001)) As wktnlonlat,
      (addy).address As stno, (addy).streetname As street,
      (addy).streettypeabbrev As styp, (addy).location As city, (addy).stateabbrev As st, ( ←
      addy).zip
FROM geocode('31 - 37 Stewart Street, Boston, MA 02116') As g;
rating |          wktnlonlat          | stno | street | styp | city | st | zip
-----+-----+-----+-----+-----+-----+-----+-----
70 | POINT(-71.06459 42.35113) | 31 | Stuart | St | Boston | MA | 02116

```

Using to do a batch geocode of addresses. Easiest is to set max_results=1. Only process those not yet geocoded (have no rating).

```

CREATE TABLE addresses_to_geocode(addid serial PRIMARY KEY, address text,
      lon numeric, lat numeric, new_address text, rating integer);

INSERT INTO addresses_to_geocode(address)
VALUES ('529 Main Street, Boston MA, 02129'),
      ('77 Massachusetts Avenue, Cambridge, MA 02139'),
      ('25 Wizard of Oz, Walaford, KS 99912323'),
      ('26 Capen Street, Medford, MA'),
      ('124 Mount Auburn St, Cambridge, Massachusetts 02138'),
      ('950 Main Street, Worcester, MA 01610');

-- only update the first 3 addresses (323-704 ms - there are caching and shared memory ←
-- effects so first geocode you do is always slower) --
-- for large numbers of addresses you don't want to update all at once
-- since the whole geocode must commit at once
-- For this example we rejoin with LEFT JOIN
-- and set to rating to -1 rating if no match
-- to ensure we don't regeocode a bad address
UPDATE addresses_to_geocode
  SET (rating, new_address, lon, lat)
    = ( COALESCE((g.geo).rating,-1), pprint_addy((g.geo).addy),
        ST_X((g.geo).geomout)::numeric(8,5), ST_Y((g.geo).geomout)::numeric(8,5) )
FROM (SELECT addid
      FROM addresses_to_geocode
      WHERE rating IS NULL ORDER BY addid LIMIT 3) As a
LEFT JOIN (SELECT addid, (geocode(address,1)) As geo
      FROM addresses_to_geocode As ag
      WHERE ag.rating IS NULL ORDER BY addid LIMIT 3) As g ON a.addid = g.addid
WHERE a.addid = addresses_to_geocode.addid;

result
-----
Query returned successfully: 3 rows affected, 480 ms execution time.

SELECT * FROM addresses_to_geocode WHERE rating is not null;

addid |          address          | lon | lat | ←
-----+-----+-----+-----+-----+-----+-----+-----
1 | 529 Main Street, Boston MA, 02129 | -71.07181 | 42.38359 | 529 Main St, ←
  | Boston, MA 02129 | 0
2 | 77 Massachusetts Avenue, Cambridge, MA 02139 | -71.09428 | 42.35988 | 77 ←
  | Massachusetts Ave, Cambridge, MA 02139 | 0
3 | 25 Wizard of Oz, Walaford, KS 99912323 | | | ←
  | | -1

```

Beispiele: Verwendung eines Geometrie-Filters

```
SELECT g.rating, ST_AsText(ST_SnapToGrid(g.geomout,0.00001)) As wktnlonlat,
      (addy).address As stno, (addy).streetname As street,
      (addy).streettypeabbrev As styp,
      (addy).location As city, (addy).stateabbrev As st, (addy).zip
FROM geocode('100 Federal Street, MA',
      3,
      (SELECT ST_Union(the_geom)
      FROM place WHERE statefp = '25' AND name = 'Lynn')::geometry
      ) As g;
```

rating	wktnlonlat	stno	street	styp	city	st	zip
8	POINT(-70.96796 42.4659)	100	Federal	St	Lynn	MA	01905

Total query runtime: 245 ms.

Siehe auch

[Normalize_Address](#), [Pprint_Addy](#), [ST_AsText](#), [ST_SnapToGrid](#), [ST_X](#), [ST_Y](#)

13.1.5 Geocode_Intersection

Geocode_Intersection — Takes in 2 streets that intersect and a state, city, zip, and outputs a set of possible locations on the first cross street that is at the intersection, also includes a geomout as the point location in NAD 83 long lat, a `normalized_address` (addy) for each location, and the rating. The lower the rating the more likely the match. Results are sorted by lowest rating first. Can optionally pass in maximum results, defaults to 10. Uses Tiger data (edges, faces, addr), PostgreSQL fuzzy string matching (soundex, levenshtein).

Synopsis

setof record **geocode_intersection**(text roadway1, text roadway2, text in_state, text in_city, text in_zip, integer max_results=10, norm_addy OUT addy, geometry OUT geomout, integer OUT rating);

Beschreibung

Takes in 2 streets that intersect and a state, city, zip, and outputs a set of possible locations on the first cross street that is at the intersection, also includes a point geometry in NAD 83 long lat, a normalized address for each location, and the rating. The lower the rating the more likely the match. Results are sorted by lowest rating first. Can optionally pass in maximum results, defaults to 10. Returns `normalized_address` (addy) for each, geomout as the point location in nad 83 long lat, and the rating. The lower the rating the more likely the match. Results are sorted by lowest rating first. Uses Tiger data (edges,faces,addr), PostgreSQL fuzzy string matching (soundex,levenshtein)

Verfügbarkeit: 2.0.0

Beispiele: Grundlagen

The below examples timings are on a 3.0 GHZ single processor Windows 7 machine with 2GB ram running PostgreSQL 9.0/PostGIS 1.5 loaded with all of MA state Tiger data loaded. Currently a bit slow (3000 ms)

Testing on Windows 2003 64-bit 8GB on PostGIS 2.0 PostgreSQL 64-bit Tiger 2011 data loaded -- (41ms)

```
SELECT pprint_addy(addy), st_astext(geomout),rating
      FROM geocode_intersection( 'Haverford St','Germania St', 'MA', 'Boston', ↵
      '02130',1);
```

pprint_addy	st_astext	rating
98 Haverford St, Boston, MA 02130	POINT(-71.101375 42.31376)	0

Even if zip is not passed in the geocoder can guess (took about 3500 ms on the windows 7 box), on the windows 2003 64-bit 741 ms

```
SELECT pprint_addy(addy), st_astext(geomout),rating
      FROM geocode_intersection('Weld', 'School', 'MA', 'Boston');
      pprint_addy      |      st_astext      | rating
-----+-----+-----
 98 Weld Ave, Boston, MA 02119 | POINT(-71.099 42.314234) |      3
 99 Weld Ave, Boston, MA 02119 | POINT(-71.099 42.314234) |      3
```

Siehe auch

[Geocode](#), [Pprint_Addy](#), [ST_AsText](#)

13.1.6 Get_Geocode_Setting

Get_Geocode_Setting — Returns value of specific setting stored in tiger.geocode_settings table.

Synopsis

text **Get_Geocode_Setting**(text setting_name);

Beschreibung

Returns value of specific setting stored in tiger.geocode_settings table. Settings allow you to toggle debugging of functions. Later plans will be to control rating with settings. Current list of settings are as follows:

name	setting	unit	category	↔	short_desc
debug_geocode_address	false	boolean	debug	↔	outputs debug information in notice log such as queries when geocode_address is called if true
debug_geocode_intersection	false	boolean	debug	↔	outputs debug information in notice log such as queries when geocode_intersection is called if true
debug_normalize_address	false	boolean	debug	↔	outputs debug information in notice log such as queries and intermediate expressions when normalize_address is called if true
debug_reverse_geocode	false	boolean	debug	↔	if true, outputs debug information in notice log such as queries and intermediate expressions when reverse_geocode
reverse_geocode_numbered_roads	0	integer	rating	↔	For state and county highways, 0 - no preference in name, 1 - prefer the numbered highway name, 2 - prefer local state/county name
use_papc_address_parser	false	boolean	normalize	↔	If set to true, will try to use the address_standardizer extension (via papc_normalize_address) instead of tiger normalize_address built one

Changed: 2.2.0 : default settings are now kept in a table called `geocode_settings_default`. Use customized settingsa are in `geocode_settings` and only contain those that have been set by user.

Verfügbarkeit: 2.1.0

Example return debugging setting

```
SELECT get_geocode_setting('debug_geocode_address') As result;
result
-----
false
```

Siehe auch

[Set_Geocode_Setting](#)

13.1.7 Get_Tract

`Get_Tract` — Returns census tract or field from tract table of where the geometry is located. Default to returning short name of tract.

Synopsis

text **get_tract**(geometry loc_geom, text output_field=name);

Beschreibung

Given a geometry will return the census tract location of that geometry. NAD 83 long lat is assumed if no spatial ref sys is specified.

Note

This function uses the census `tract` which is not loaded by default. If you have already loaded your state table, you can load `tract` as well as `bg`, and `tabblock` using the [Loader_Generate_Census_Script](#) script.



If you have not loaded your state data yet and want these additional tables loaded, do the following

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

then they will be included by the [Loader_Generate_Script](#).

Verfügbarkeit: 2.0.0

Beispiele: Grundlagen

```
SELECT get_tract(ST_Point(-71.101375, 42.31376) ) As tract_name;
tract_name
-----
1203.01
```

```
--this one returns the tiger geoid
SELECT get_tract(ST_Point(-71.101375, 42.31376), 'tract_id' ) As tract_id;
tract_id
-----
25025120301
```

Siehe auch[Geocode](#) >

13.1.8 Install_Missing_Indexes

Install_Missing_Indexes — Finds all tables with key columns used in geocoder joins and filter conditions that are missing used indexes on those columns and will add them.

Synopsis

```
boolean Install_Missing_Indexes();
```

Beschreibung

Finds all tables in `tiger` and `tiger_data` schemas with key columns used in geocoder joins and filters that are missing indexes on those columns and will output the SQL DDL to define the index for those tables and then execute the generated script. This is a helper function that adds new indexes needed to make queries faster that may have been missing during the load process. This function is a companion to [Missing_Indexes_Generate_Script](#) that in addition to generating the create index script, also executes it. It is called as part of the `update_geocode.sql` upgrade script.

Verfügbarkeit: 2.0.0

Beispiele

```
SELECT install_missing_indexes();
       install_missing_indexes
-----
t
```

Siehe auch

[Loader_Generate_Script](#), [Missing_Indexes_Generate_Script](#)

13.1.9 Loader_Generate_Census_Script

Loader_Generate_Census_Script — Generates a shell script for the specified platform for the specified states that will download Tiger census state `tract`, `bg`, and `tabblocks` data tables, stage and load into `tiger_data` schema. Each state script is returned as a separate record.

Synopsis

```
setof text loader_generate_census_script(text[] param_states, text os);
```

Beschreibung

Generates a shell script for the specified platform for the specified states that will download Tiger data census state `tract`, block groups `bg`, and `tabblocks` data tables, stage and load into `tiger_data` schema. Each state script is returned as a separate record.

It uses `unzip` on Linux (7-zip on Windows by default) and `wget` to do the downloading. It uses [Section 4.4.2](#) to load in the data. Note the smallest unit it does is a whole state. It will only process the files in the staging and temp folders.

It uses the following control tables to control the process and different OS shell syntax variations.

1. `loader_variables` keeps track of various variables such as census site, year, data and staging schemas
2. `loader_platform` profiles of various platforms and where the various executables are located. Comes with windows and linux. More can be added.
3. `loader_lookuptables` each record defines a kind of table (state, county), whether to process records in it and how to load them in. Defines the steps to import data, stage data, add, removes columns, indexes, and constraints for each. Each table is prefixed with the state and inherits from a table in the tiger schema. e.g. creates `tiger_data.ma_faces` which inherits from `tiger.faces`

Verfügbarkeit: 2.0.0



Note

Loader_Generate_Script includes this logic, but if you installed tiger geocoder prior to PostGIS 2.0.0 alpha5, you'll need to run this on the states you have already done to get these additional tables.

Beispiele

Generate script to load up data for select states in Windows shell script format.

```
SELECT loader_generate_census_script (ARRAY['MA'], 'windows');
-- result --
set STATEDIR="\gisdata\www2.census.gov\geo\pvs\tiger2010st\25_Massachusetts"
set TMPDIR=\gisdata\temp\
set UNZIPTOOL="C:\Program Files\7-Zip\7z.exe"
set WGETTOOL="C:\wget\wget.exe"
set PGBIN=C:\projects\pg\pg91win\bin\
set PGPORT=5432
set PGHOST=localhost
set PGUSER=postgres
set PGPASSWORD=yourpasswordhere
set PGDATABASE=tiger_postgis20
set PSQL="%PGBIN%psql"
set SHP2PGSQL="%PGBIN%shp2pgsql"
cd \gisdata

%WGETTOOL% http://www2.census.gov/geo/pvs/tiger2010st/25_Massachusetts/25/ --no-parent -- \
    relative --accept=*bg10.zip,*tract10.zip,*tabblock10.zip --mirror --reject=html
del %TMPDIR%\*.* /Q
%PSQL% -c "DROP SCHEMA tiger_staging CASCADE;"
%PSQL% -c "CREATE SCHEMA tiger_staging;"
cd %STATEDIR%
for /r %%z in (*.zip) do %UNZIPTOOL% e %%z -o%TMPDIR%
cd %TMPDIR%
%PSQL% -c "CREATE TABLE tiger_data.MA_tract(CONSTRAINT pk_MA_tract PRIMARY KEY (tract_id) ) \
    INHERITS(tiger.tract); "
%SHP2PGSQL% -c -s 4269 -g the_geom -W "latin1" tl_2010_25_tract10.dbf tiger_staging. \
    ma_tract10 | %PSQL%
%PSQL% -c "ALTER TABLE tiger_staging.MA_tract10 RENAME geoid10 TO tract_id; SELECT \
    loader_load_staged_data(lower('MA_tract10'), lower('MA_tract')); "
%PSQL% -c "CREATE INDEX tiger_data_MA_tract_the_geom_gist ON tiger_data.MA_tract USING gist \
    (the_geom); "
%PSQL% -c "VACUUM ANALYZE tiger_data.MA_tract;"
%PSQL% -c "ALTER TABLE tiger_data.MA_tract ADD CONSTRAINT chk_statefp CHECK (statefp = \
    '25');"
:
```

Erzeugt ein Shell-Skript

```

STATEDIR="/gisdata/www2.census.gov/geo/pvs/tiger2010st/25_Massachusetts"
TMPDIR="/gisdata/temp/"
UNZIPTOOL=unzip
WGETTOOL="/usr/bin/wget"
export PGBIN=/usr/pgsql-9.0/bin
export PGPORT=5432
export PGHOST=localhost
export PGUSER=postgres
export PGPASSWORD=yourpasswordhere
export PGDATABASE=geocoder
PSQL=${PGBIN}/psql
SHP2PGSQL=${PGBIN}/shp2pgsql
cd /gisdata

wget http://www2.census.gov/geo/pvs/tiger2010st/25_Massachusetts/25/ --no-parent --relative ←
    --accept=*bgl0.zip,*tract10.zip,*tabblock10.zip --mirror --reject=html
rm -f ${TMPDIR}/*. *
${PSQL} -c "DROP SCHEMA tiger_staging CASCADE;"
${PSQL} -c "CREATE SCHEMA tiger_staging;"
cd $STATEDIR
for z in *.zip; do $UNZIPTOOL -o -d $TMPDIR $z; done
:
:

```

Siehe auch

[Loader_Generate_Script](#)

13.1.10 Loader_Generate_Script

Loader_Generate_Script — Generates a shell script for the specified platform for the specified states that will download Tiger data, stage and load into `tiger_data` schema. Each state script is returned as a separate record. Latest version supports Tiger 2010 structural changes and also loads census tract, block groups, and blocks tables.

Synopsis

setof text **loader_generate_script**(text[] param_states, text os);

Beschreibung

Generates a shell script for the specified platform for the specified states that will download Tiger data, stage and load into `tiger_data` schema. Each state script is returned as a separate record.

It uses unzip on Linux (7-zip on Windows by default) and wget to do the downloading. It uses Section 4.4.2 to load in the data. Note the smallest unit it does is a whole state, but you can overwrite this by downloading the files yourself. It will only process the files in the staging and temp folders.

It uses the following control tables to control the process and different OS shell syntax variations.

1. `loader_variables` keeps track of various variables such as census site, year, data and staging schemas
2. `loader_platform` profiles of various platforms and where the various executables are located. Comes with windows and linux. More can be added.

3. `loader_lookuptables` each record defines a kind of table (state, county), whether to process records in it and how to load them in. Defines the steps to import data, stage data, add, removes columns, indexes, and constraints for each. Each table is prefixed with the state and inherits from a table in the tiger schema. e.g. creates `tiger_data.ma_faces` which inherits from `tiger.faces`

Availability: 2.0.0 to support Tiger 2010 structured data and load census tract (tract), block groups (bg), and blocks (tabblocks) tables .



Note

If you are using pgAdmin 3, be warned that by default pgAdmin 3 truncates long text. To fix, change *File -> Options -> Query Tool -> Query Editor -> Max. characters per column* to larger than 50000 characters.

Beispiele

Using `psql` where `gistest` is your database and `/gisdata/data_load.sh` is the file to create with the shell commands to run.

```
psql -U postgres -h localhost -d gistest -A -t \
-c "SELECT Loader_Generate_Script (ARRAY['MA'], 'gistest') "
> /gisdata/data_load.sh;
```

Generate script to load up data for 2 states in Windows shell script format.

```
SELECT loader_generate_script (ARRAY['MA','RI'], 'windows') AS result;
-- result --
set TMPDIR=\gisdata\temp\
set UNZIPTOOL="C:\Program Files\7-Zip\7z.exe"
set WGETTOOL="C:\wget\wget.exe"
set PGBIN=C:\Program Files\PostgreSQL\9.4\bin\
set PGPORT=5432
set PGHOST=localhost
set PGUSER=postgres
set PGPASSWORD=yourpasswordhere
set PGDATABASE=geocoder
set PSQL="%PGBIN%psql"
set SHP2PGSQL="%PGBIN%shp2pgsql"
cd \gisdata

cd \gisdata
%WGETTOOL% ftp://ftp2.census.gov/geo/tiger/TIGER2015/PLACE/tl*_25_* --no-parent --relative ↵
--recursive --level=2 --accept=zip --mirror --reject=html
cd \gisdata/ftp2.census.gov/geo/tiger/TIGER2015/PLACE
:
:
```

Erzeugt ein Shell-Skript

```
SELECT loader_generate_script (ARRAY['MA','RI'], 'sh') AS result;
-- result --
TMPDIR="/gisdata/temp/"
UNZIPTOOL=unzip
WGETTOOL="/usr/bin/wget"
export PGBIN=/usr/lib/postgresql/9.4/bin
export PGPORT=5432
export PGHOST=localhost
export PGUSER=postgres
export PGPASSWORD=yourpasswordhere
export PGDATABASE=geocoder
PSQL=${PGBIN}/psql
```

```

SHP2PGSQL=${PGBIN}/shp2pgsql
cd /gisdata

cd /gisdata
wget ftp://ftp2.census.gov/geo/tiger/TIGER2015/PLACE/tl*_25* --no-parent --relative -- \
    recursive --level=2 --accept=zip --mirror --reject=html
cd /gisdata/ftp2.census.gov/geo/tiger/TIGER2015/PLACE
rm -f ${TMPDIR}/*. *
:
:

```

Siehe auch

Section [2.8.1](#), [Loader_Generate_Nation_Script](#)

13.1.11 Loader_Generate_Nation_Script

Loader_Generate_Nation_Script — Erzeugt für die angegebene Plattform ein Shell-Skript, welches die County und State Lookup Tabellen ladet.

Synopsis

```
text loader_generate_nation_script(text os);
```

Beschreibung

Generates a shell script for the specified platform that loads in the `county_all`, `county_all_lookup`, `state_all` tables into `tiger_data` schema. These inherit respectively from the `county`, `county_lookup`, `state` tables in `tiger` schema.

It uses `unzip` on Linux (7-zip on Windows by default) and `wget` to do the downloading. It uses Section [4.4.2](#) to load in the data.

It uses the following control tables `tiger.loader_platform`, `tiger.loader_variables`, and `tiger.loader_lookuptables` to control the process and different OS shell syntax variations.

1. `loader_variables` keeps track of various variables such as census site, year, data and staging schemas
2. `loader_platform` profiles of various platforms and where the various executables are located. Comes with windows and linux/unix. More can be added.
3. `loader_lookuptables` each record defines a kind of table (state, county), whether to process records in it and how to load them in. Defines the steps to import data, stage data, add, removes columns, indexes, and constraints for each. Each table is prefixed with the state and inherits from a table in the `tiger` schema. e.g. creates `tiger_data.ma_faces` which inherits from `tiger.faces`

Verfügbarkeit: 2.1.0



Note

If you were running `tiger_2010` version and you want to reload as state with `tiger_2011`, you'll need to for the very first load generate and run drop statements [Drop_Nation_Tables_Generate_Script](#) before you run this script.

Beispiele

Generate script to load nation data Windows.

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

Generate script to load up data for Linux/Unix systems.

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

Siehe auch

[Loader_Generate_Script](#)

13.1.12 Missing_Indexes_Generate_Script

Missing_Indexes_Generate_Script — Finds all tables with key columns used in geocoder joins that are missing indexes on those columns and will output the SQL DDL to define the index for those tables.

Synopsis

text **Missing_Indexes_Generate_Script**();

Beschreibung

Finds all tables in `tiger` and `tiger_data` schemas with key columns used in geocoder joins that are missing indexes on those columns and will output the SQL DDL to define the index for those tables. This is a helper function that adds new indexes needed to make queries faster that may have been missing during the load process. As the geocoder is improved, this function will be updated to accommodate new indexes being used. If this function outputs nothing, it means all your tables have what we think are the key indexes already in place.

Verfügbarkeit: 2.0.0

Beispiele

```
SELECT missing_indexes_generate_script();
-- output: This was run on a database that was created before many corrections were made to ←
the loading script --
CREATE INDEX idx_tiger_county_countyfp ON tiger.county USING btree(countyfp);
CREATE INDEX idx_tiger_cousub_countyfp ON tiger.cousub USING btree(countyfp);
CREATE INDEX idx_tiger_edges_tfidr ON tiger.edges USING btree(tfidr);
CREATE INDEX idx_tiger_edges_tfidl ON tiger.edges USING btree(tfidl);
CREATE INDEX idx_tiger_zip_lookup_all_zip ON tiger.zip_lookup_all USING btree(zip);
CREATE INDEX idx_tiger_data_ma_county_countyfp ON tiger_data.ma_county USING btree(countyfp ←
);
CREATE INDEX idx_tiger_data_ma_cousub_countyfp ON tiger_data.ma_cousub USING btree(countyfp ←
);
CREATE INDEX idx_tiger_data_ma_edges_countyfp ON tiger_data.ma_edges USING btree(countyfp);
CREATE INDEX idx_tiger_data_ma_faces_countyfp ON tiger_data.ma_faces USING btree(countyfp);
```

Siehe auch

[Loader_Generate_Script](#), [Install_Missing_Indexes](#)

13.1.13 Normalize_Address

Normalize_Address — Given a textual street address, returns a composite `norm_addy` type that has road suffix, prefix and type standardized, street, streetname etc. broken into separate fields. This function will work with just the lookup data packaged with the `tiger_geocoder` (no need for tiger census data).

Synopsis

```
norm_addy normalize_address(varchar in_address);
```

Beschreibung

Given a textual street address, returns a composite `norm_addy` type that has road suffix, prefix and type standardized, street, streetname etc. broken into separate fields. This is the first step in the geocoding process to get all addresses into normalized postal form. No other data is required aside from what is packaged with the geocoder.

This function just uses the various direction/state/suffix lookup tables preloaded with the `tiger_geocoder` and located in the `tiger` schema, so it doesn't need you to download tiger census data or any other additional data to make use of it. You may find the need to add more abbreviations or alternative namings to the various lookup tables in the `tiger` schema.

It uses various control lookup tables located in `tiger` schema to normalize the input address.

Fields in the `norm_addy` type object returned by this function in this order where () indicates a field required by the geocoder, [] indicates an optional field:

(address) [predirAbbrev] (streetName) [streetTypeAbbrev] [postdirAbbrev] [internal] [location] [stateAbbrev] [zip] [parsed] [zip4] [address_alphanumeric]

Enhanced: 2.4.0 `norm_addy` object includes additional fields `zip4` and `address_alphanumeric`.

1. `address` ist eine Ganzzahl: Die Hausnummer
2. `predirAbbrev` ist ein Textfeld variabler Länge: Ein Präfix für die Straßenrichtung, wie N, S, E, W etc. Wird durch die `direction_lookup` Tabelle gesteuert.
3. `streetName` varchar
4. `streetTypeAbbrev` varchar abbreviated version of street type: e.g. St, Ave, Cir. These are controlled using the `street_type_lookup` table.
5. `postdirAbbrev` varchar abbreviated directional suffice of road N, S, E, W etc. These are controlled using the `direction_lookup` table.
6. `internal` varchar internal address such as an apartment or suite number.
7. `location` varchar usually a city or governing province.
8. `stateAbbrev` varchar two character US State. e.g MA, NY, MI. These are controlled by the `state_lookup` table.
9. `zip` varchar 5-digit zipcode. e.g. 02109.
10. `parsed` boolean - denotes if address was formed from normalize process. The `normalize_address` function sets this to true before returning the address.
11. `zip4` last 4 digits of a 9 digit zip code. Availability: PostGIS 2.4.0.
12. `address_alphanumeric` Full street number even if it has alpha characters like 17R. Parsing of this is better using [Pgcn Normalize_Address](#) function. Availability: PostGIS 2.4.0.

Beispiele

Gibt die ausgewählten Felder aus. Verwenden Sie bitte **Pprint_Addy** wenn Sie einen sauber formatierten Ausgabetext benötigen.

```
SELECT address As orig, (g.na).streetname, (g.na).streettypeabbrev
FROM (SELECT address, normalize_address(address) As na
      FROM addresses_to_geocode) As g;
```

orig	streetname	streettypeabbrev
28 Capen Street, Medford, MA	Capen	St
124 Mount Auburn St, Cambridge, Massachusetts 02138	Mount Auburn	St
950 Main Street, Worcester, MA 01610	Main	St
529 Main Street, Boston MA, 02129	Main	St
77 Massachusetts Avenue, Cambridge, MA 02139	Massachusetts	Ave
25 Wizard of Oz, Walaford, KS 99912323	Wizard of Oz	

Siehe auch

Geocode, **Pprint_Addy**

13.1.14 Pagc_Normalize_Address

Pagc_Normalize_Address — Given a textual street address, returns a composite `norm_addy` type that has road suffix, prefix and type standardized, street, streetname etc. broken into separate fields. This function will work with just the lookup data packaged with the `tiger_geocoder` (no need for tiger census data). Requires `address_standardizer` extension.

Synopsis

```
norm_addy pagc_normalize_address(varchar in_address);
```

Beschreibung

Given a textual street address, returns a composite `norm_addy` type that has road suffix, prefix and type standardized, street, streetname etc. broken into separate fields. This is the first step in the geocoding process to get all addresses into normalized postal form. No other data is required aside from what is packaged with the geocoder.

This function just uses the various `pagc_*` lookup tables preloaded with the `tiger_geocoder` and located in the `tiger` schema, so it doesn't need you to download tiger census data or any other additional data to make use of it. You may find the need to add more abbreviations or alternative namings to the various lookup tables in the `tiger` schema.

It uses various control lookup tables located in `tiger` schema to normalize the input address.

Fields in the `norm_addy` type object returned by this function in this order where () indicates a field required by the geocoder, [] indicates an optional field:

There are slight variations in casing and formatting over the **Normalize_Address**.

Verfügbarkeit: 2.1.0



This method needs `address_standardizer` extension.

```
(address) [predirAbbrev] (streetName) [streetTypeAbbrev] [postdirAbbrev] [internal] [location] [stateAbbrev] [zip]
```

The native `standardaddr` of `address_standardizer` extension is at this time a bit richer than `norm_addy` since its designed to support international addresses (including country). `standardaddr` equivalent fields are:

```
house_num, predir, name, suftype, sufdir, unit, city, state, postcode
```

Enhanced: 2.4.0 `norm_addy` object includes additional fields `zip4` and `address_alphanumeric`.

1. address ist eine Ganzzahl: Die Hausnummer
2. predirAbbrev ist ein Textfeld variabler Länge: Ein Präfix für die Straßenrichtung, wie N, S, E, W etc. Wird durch die direction_lookup Tabelle gesteuert.
3. streetName varchar
4. streetTypeAbbrev varchar abbreviated version of street type: e.g. St, Ave, Cir. These are controlled using the street_type_lookup table.
5. postdirAbbrev varchar abbreviated directional suffice of road N, S, E, W etc. These are controlled using the direction_lookup table.
6. internal varchar internal address such as an apartment or suite number.
7. location varchar usually a city or governing province.
8. stateAbbrev varchar two character US State. e.g MA, NY, MI. These are controlled by the state_lookup table.
9. zip varchar 5-digit zipcode. e.g. 02109.
10. parsed boolean - denotes if address was formed from normalize process. The normalize_address function sets this to true before returning the address.
11. zip4 last 4 digits of a 9 digit zip code. Availability: PostGIS 2.4.0.
12. address_alphanumeric Full street number even if it has alpha characters like 17R. Parsing of this is better using [Pagc_Normalize_Address](#) function. Availability: PostGIS 2.4.0.

Beispiele

Beispiel für einen Einzelaufruf

```
SELECT addy.*
FROM pagc_normalize_address('9000 E ROO ST STE 999, Springfield, CO') AS addy;
```

address	predirabbrev	streetname	streettypeabbrev	postdirabbrev	internal	location	stateabbrev	zip	parsed
9000	E	ROO	ST		SUITE 999	SPRINGFIELD	CO		t

Batch call. There are currently speed issues with the way postgis_tiger_geocoder wraps the address_standardizer. These will hopefully be resolved in later editions. To work around them, if you need speed for batch geocoding to call generate a normaddy in batch mode, you are encouraged to directly call the address_standardizer standardize_address function as shown below which is similar exercise to what we did in [Normalize_Address](#) that uses data created in [Geocode](#).

```
WITH g AS (SELECT address, ROW((sa).house_num, (sa).predir, (sa).name
, (sa).suftype, (sa).sufdir, (sa).unit , (sa).city, (sa).state, (sa).postcode, true)::
norm_addy As na
FROM (SELECT address, standardize_address('tiger.pagc_lex'
, 'tiger.pagc_gaz'
, 'tiger.pagc_rules', address) As sa
FROM addresses_to_geocode) As g)
SELECT address As orig, (g.na).streetname, (g.na).streettypeabbrev
FROM g;
```

orig	streetname	streettypeabbrev
529 Main Street, Boston MA, 02129	MAIN	ST
77 Massachusetts Avenue, Cambridge, MA 02139	MASSACHUSETTS	AVE

25 Wizard of Oz, Walaford, KS 99912323		WIZARD OF	
26 Capen Street, Medford, MA		CAPEN	ST
124 Mount Auburn St, Cambridge, Massachusetts 02138		MOUNT AUBURN	ST
950 Main Street, Worcester, MA 01610		MAIN	ST

Siehe auch

[Normalize_Address](#), [Geocode](#)

13.1.15 Pprint_Addy

Pprint_Addy — Given a `norm_addy` composite type object, returns a pretty print representation of it. Usually used in conjunction with `normalize_address`.

Synopsis

`varchar pprint_addy(norm_addy in_addy);`

Beschreibung

Given a `norm_addy` composite type object, returns a pretty print representation of it. No other data is required aside from what is packaged with the geocoder.

Wird üblicherweise in Verbindung mit [Normalize_Address](#) verwendet.

Beispiele

Pretty print a single address

```
SELECT pprint_addy(normalize_address('202 East Fremont Street, Las Vegas, Nevada 89101'))
      As pretty_address;
      pretty_address
-----
202 E Fremont St, Las Vegas, NV 89101
```

Pretty print address a table of addresses

```
SELECT address As orig, pprint_addy(normalize_address(address)) As pretty_address
      FROM addresses_to_geocode;
```

orig	pretty_address
529 Main Street, Boston MA, 02129	529 Main St, Boston MA, 02129
77 Massachusetts Avenue, Cambridge, MA 02139	77 Massachusetts Ave, Cambridge, MA 02139 ↩
28 Capen Street, Medford, MA	28 Capen St, Medford, MA
124 Mount Auburn St, Cambridge, Massachusetts 02138	124 Mount Auburn St, Cambridge, MA 02138 ↩
950 Main Street, Worcester, MA 01610	950 Main St, Worcester, MA 01610

Siehe auch

[Normalize_Address](#)

13.1.16 Reverse_Geocode

Reverse_Geocode — Takes a geometry point in a known spatial ref sys and returns a record containing an array of theoretically possible addresses and an array of cross streets. If `include_strnum_range = true`, includes the street range in the cross streets.

Synopsis

record **Reverse_Geocode**(geometry pt, boolean include_strnum_range=false, geometry[] OUT intpt, norm_addy[] OUT addy, varchar[] OUT street);

Beschreibung

Takes a geometry point in a known spatial ref and returns a record containing an array of theoretically possible addresses and an array of cross streets. If `include_strnum_range = true`, includes the street range in the cross streets. `include_strnum_range` defaults to false if not passed in. Addresses are sorted according to which road a point is closest to so first address is most likely the right one.

Why do we say theoretical instead of actual addresses. The Tiger data doesn't have real addresses, but just street ranges. As such the theoretical address is an interpolated address based on the street ranges. Like for example interpolating one of my addresses returns a 26 Court St. and 26 Court Sq., though there is no such place as 26 Court Sq. This is because a point may be at a corner of 2 streets and thus the logic interpolates along both streets. The logic also assumes addresses are equally spaced along a street, which of course is wrong since you can have a municipal building taking up a good chunk of the street range and the rest of the buildings are clustered at the end.

Note: Hmm this function relies on Tiger data. If you have not loaded data covering the region of this point, then hmm you will get a record filled with NULLS.

Die zurückgelieferten Elemente des Datensatzes lauten wie folgt:

1. `intpt` is an array of points: These are the center line points on the street closest to the input point. There are as many points as there are addresses.
2. `addy` is an array of `norm_addy` (normalized addresses): These are an array of possible addresses that fit the input point. The first one in the array is most likely. Generally there should be only one, except in the case when a point is at the corner of 2 or 3 streets, or the point is somewhere on the road and not off to the side.
3. `street` an array of `varchar`: These are cross streets (or the street) (streets that intersect or are the street the point is projected to be on).

Verfügbarkeit: 2.0.0

Beispiele

Example of a point at the corner of two streets, but closest to one. This is approximate location of MIT: 77 Massachusetts Ave, Cambridge, MA 02139 Note that although we don't have 3 streets, PostgreSQL will just return null for entries above our upper bound so safe to use. This includes street ranges

```
SELECT pprint_addy(r.addy[1]) As st1, pprint_addy(r.addy[2]) As st2, pprint_addy(r.addy[3]) ←
      As st3,
      array_to_string(r.street, ',') As cross_streets
FROM reverse_geocode(ST_GeomFromText('POINT(-71.093902 42.359446)',4269),true) As r ←
;
```

```
result
-----
st1                                     | st2 | st3 |                                     cross_streets
-----+-----+-----+-----
67 Massachusetts Ave, Cambridge, MA 02139 |      |      | 67 - 127 Massachusetts Ave,32 - 88 ←
Vassar St
```

Here we choose not to include the address ranges for the cross streets and picked a location really really close to a corner of 2 streets thus could be known by two different addresses.

```
SELECT pprint_addy(r.addy[1]) As st1, pprint_addy(r.addy[2]) As st2,
pprint_addy(r.addy[3]) As st3, array_to_string(r.street, ',') As cross_str
FROM reverse_geocode(ST_GeomFromText('POINT(-71.06941 42.34225)',4269)) As r;
```

result

st1	st2	st3	cross_str
5 Bradford St, Boston, MA 02118	49 Waltham St, Boston, MA 02118		Waltham St

For this one we reuse our geocoded example from [Geocode](#) and we only want the primary address and at most 2 cross streets.

```
SELECT actual_addr, lon, lat, pprint_addy((rg).addy[1]) As int_addr1,
(rg).street[1] As cross1, (rg).street[2] As cross2
FROM (SELECT address As actual_addr, lon, lat,
reverse_geocode( ST_SetSRID(ST_Point(lon,lat),4326) ) As rg
FROM addresses_to_geocode WHERE rating
> -1) As foo;
```

actual_addr	int_addr1	lon	lat	cross1	cross2
529 Main Street, Boston MA, 02129		-71.07181	42.38359	527 Main St,	
Boston, MA 02129	Medford St				
77 Massachusetts Avenue, Cambridge, MA 02139		-71.09428	42.35988	77	
Massachusetts Ave, Cambridge, MA 02139	Vassar St				
26 Capen Street, Medford, MA		-71.12377	42.41101	9 Edison Ave,	
Medford, MA 02155	Capen St			Tesla Ave	
124 Mount Auburn St, Cambridge, Massachusetts 02138		-71.12304	42.37328	3 University	
Rd, Cambridge, MA 02138	Mount Auburn St				
950 Main Street, Worcester, MA 01610		-71.82368	42.24956	3 Maywood St,	
Worcester, MA 01603	Main St			Maywood Pl	

Siehe auch

[Pprint_Addy](#), [Geocode](#)

13.1.17 Topology_Load_Tiger

Topology_Load_Tiger — Loads a defined region of tiger data into a PostGIS Topology and transforming the tiger data to spatial reference of the topology and snapping to the precision tolerance of the topology.

Synopsis

text **Topology_Load_Tiger**(varchar topo_name, varchar region_type, varchar region_id);

Beschreibung

Loads a defined region of tiger data into a PostGIS Topology. The faces, nodes and edges are transformed to the spatial reference system of the target topology and points are snapped to the tolerance of the target topology. The created faces, nodes, edges

maintain the same ids as the original Tiger data faces, nodes, edges so that datasets can be in the future be more easily reconciled with tiger data. Returns summary details about the process.

This would be useful for example for redistricting data where you require the newly formed polygons to follow the center lines of streets and for the resulting polygons not to overlap.



Note

This function relies on Tiger data as well as the installation of the PostGIS topology module. For more information, refer to Chapter 11 and Section 2.4.1. If you have not loaded data covering the region of interest, then no topology records will be created. This function will also fail if you have not created a topology using the topology functions.



Note

Most topology validation errors are a result of tolerance issues where after transformation the edges points don't quite line up or overlap. To remedy the situation you may want to increase or lower the precision if you get topology validation failures.

Benötigte Parameter:

1. `topo_name` Die Bezeichnung einer bestehenden PostGIS Topologie, in die Daten geladen werden.
2. `region_type` The type of bounding region. Currently only `place` and `county` are supported. Plan is to have several more. This is the table to look into to define the region bounds. e.g `tiger.place`, `tiger.county`
3. `region_id` This is what TIGER calls the geoid. It is the unique identifier of the region in the table. For place it is the `plcidfp` column in `tiger.place`. For county it is the `cntyidfp` column in `tiger.county`

Verfügbarkeit: 2.0.0

Beispiel: Boston, Massachusetts Topologie

Create a topology for Boston, Massachusetts in Mass State Plane Feet (2249) with tolerance 0.25 feet and then load in Boston city tiger faces, edges, nodes.

```
SELECT topology.CreateTopology('topo_boston', 2249, 0.25);
createtopology
-----
15
-- 60,902 ms ~ 1 minute on windows 7 desktop running 9.1 (with 5 states tiger data loaded)
SELECT tiger.topology_load_tiger('topo_boston', 'place', '2507000');
-- topology_loader_tiger --
29722 edges holding in temporary. 11108 faces added. 1875 edges of faces added. 20576 ↔
nodes added.
19962 nodes contained in a face. 0 edge start end corrected. 31597 edges added.

-- 41 ms --
SELECT topology.TopologySummary('topo_boston');
-- topologysummary--
Topology topo_boston (15), SRID 2249, precision 0.25
20576 nodes, 31597 edges, 11109 faces, 0 topogeoms in 0 layers

-- 28,797 ms to validate yeh returned no errors --
SELECT * FROM
    topology.ValidateTopology('topo_boston');

    error      |    id1    |    id2
-----+-----+-----
```

Beispiel: Suffolk, Massachusetts Topologie

Create a topology for Suffolk, Massachusetts in Mass State Plane Meters (26986) with tolerance 0.25 meters and then load in Suffolk county tiger faces, edges, nodes.

```
SELECT topology.CreateTopology('topo_suffolk', 26986, 0.25);
-- this took 56,275 ms ~ 1 minute on Windows 7 32-bit with 5 states of tiger loaded
-- must have been warmed up after loading boston
SELECT tiger.topology_load_tiger('topo_suffolk', 'county', '25025');
-- topology_loader_tiger --
36003 edges holding in temporary. 13518 faces added. 2172 edges of faces added.
24761 nodes added. 24075 nodes contained in a face. 0 edge start end corrected. 38175 ↔
edges added.
-- 31 ms --
SELECT topology.TopologySummary('topo_suffolk');
-- topologysummary--
Topology topo_suffolk (14), SRID 26986, precision 0.25
24761 nodes, 38175 edges, 13519 faces, 0 topogeoms in 0 layers

-- 33,606 ms to validate --
SELECT * FROM
    topology.ValidateTopology('topo_suffolk');
```

error	id1	id2
coincident nodes	81045651	81064553
edge crosses node	81045651	85737793
edge crosses node	81045651	85742215
edge crosses node	81045651	620628939
edge crosses node	81064553	85697815
edge crosses node	81064553	85728168
edge crosses node	81064553	85733413

Siehe auch

[CreateTopology](#), [CreateTopoGeom](#), [TopologySummary](#), [ValidateTopology](#)

13.1.18 Set_Geocode_Setting

Set_Geocode_Setting — Sets a setting that affects behavior of geocoder functions.

Synopsis

```
text Set_Geocode_Setting(text setting_name, text setting_value);
```

Beschreibung

Sets value of specific setting stored in `tiger.geocode_settings` table. Settings allow you to toggle debugging of functions. Later plans will be to control rating with settings. Current list of settings are listed in [Get_Geocode_Setting](#).

Verfügbarkeit: 2.1.0

Example return debugging setting

If you run [Geocode](#) when this function is true, the NOTICE log will output timing and queries.

```
SELECT set_geocode_setting('debug_geocode_address', 'true') As result;  
result  
-----  
true
```

Siehe auch

[Get_Geocode_Setting](#)

Chapter 14

PostGIS Special Functions Index

14.1 PostGIS Aggregate Functions

The functions given below are spatial aggregate functions provided with PostGIS that can be used just like any other sql aggregate function such as sum, average.

- **ST_3DExtent** - an aggregate function that returns the box3D bounding box that bounds rows of geometries.
- **ST_Accum** - Aggregatfunktion. Erzeugt ein Feld mit Geometrien.
- **ST_AsGeobuf** - Gibt eine Menge an Zeilen in der Geobuf Darstellung aus.
- **ST_AsMVT** - Gibt Zeilen in der Mapbox Vector Tile Darstellung aus.
- **ST_ClusterIntersecting** - Aggregate. Returns an array with the connected components of a set of geometries
- **ST_ClusterWithin** - Aggregate. Returns an array of GeometryCollections, where each GeometryCollection represents a set of geometries separated by no more than the specified distance.
- **ST_Collect** - Gibt einen festgelegten ST_Geometry Wert aus einer Sammlung anderer Geometrien zurück.
- **ST_Extent** - an aggregate function that returns the bounding box that bounds rows of geometries.
- **ST_MakeLine** - Erzeugt einen Linienzug aus einer Punkt-, Mehrfachpunkt- oder Liniengeometrie.
- **ST_MemUnion** - Das gleiche wie ST_Union, nur freundlicher zum Arbeitsspeicher (verwendet weniger Arbeitsspeicher und mehr Rechnerzeit).
- **ST_Polygonize** - Aggregatfunktion. Erzeugt eine Sammelgeometrie/GeometryCollection, welche Polygone enthält, die aus den einzelnen Linien einer Menge von Geometrien gebildet werden können.
- **ST_SameAlignment** - Returns true if rasters have same skew, scale, spatial ref, and offset (pixels can be put on same grid without cutting into pixels) and false if they don't with notice detailing issue.
- **ST_Union** - Gibt eine Geometrie zurück, welche der mengentheoretischen Vereinigung der Geometrien entspricht.
- **TopoElementArray_Agg** - Gibt für eine Menge an element_id, type Feldern (topoelements) ein topoelementarray zurück

14.2 PostGIS Window Functions

The functions given below are spatial window functions provided with PostGIS that can be used just like any other sql window function such as row_number(), lead(), lag(). All these require an SQL OVER() clause.

- **ST_ClusterDBSCAN** - Windowing function that returns integer id for the cluster each input geometry is in based on 2D implementation of Density-based spatial clustering of applications with noise (DBSCAN) algorithm.
 - **ST_ClusterKMeans** - Windowing function that returns integer id for the cluster each input geometry is in.
-

14.3 PostGIS SQL-MM Compliant Functions

The functions given below are PostGIS functions that conform to the SQL/MM 3 standard



Note

SQL-MM defines the default SRID of all geometry constructors as 0. PostGIS uses a default SRID of -1.

- **ST_3DDWithin** - For 3d (z) geometry type Returns true if two geometries 3d distance is within number of units. This method implements the SQL/MM specification. SQL-MM ?
- **ST_3DDistance** - For geometry type Returns the 3-dimensional cartesian minimum distance (based on spatial ref) between two geometries in projected units. This method implements the SQL/MM specification. SQL-MM ?
- **ST_3DIntersects** - Returns TRUE if the Geometries "spatially intersect" in 3d - only for points, linestrings, polygons, polyhedral surface (area). With SFCGAL backend enabled also supports TINS This method implements the SQL/MM specification. SQL-MM 3: ?
- **ST_AddEdgeModFace** - Fügt eine Kante hinzu. Falls dabei eine Masche aufgetrennt wird, so wird die ursprüngliche Masche angepasst und eine weitere Masche hinzugefügt. This method implements the SQL/MM specification. SQL-MM: Topo-Geo and Topo-Net 3: Routine Details: X.3.13
- **ST_AddEdgeNewFaces** - Fügt eine Kante hinzu. Falls dabei eine Masche aufgetrennt wird, so wird die ursprüngliche Masche gelöscht und durch zwei neue Maschen ersetzt. This method implements the SQL/MM specification. SQL-MM: Topo-Geo and Topo-Net 3: Routine Details: X.3.12
- **ST_AddIsoEdge** - Fügt eine isolierte Kante, die durch die Geometrie alinestring festgelegt wird zu einer Topologie hinzu, indem zwei bestehende isolierte Knoten anode und anothernode verbunden werden. Gibt die "edgeid" der neuen Kante aus. This method implements the SQL/MM specification. SQL-MM: Topo-Geo and Topo-Net 3: Routine Details: X.3.4
- **ST_AddIsoNode** - Fügt einen isolierten Knoten zu einer Masche in einer Topologie hinzu und gibt die "nodeid" des neuen Knotens aus. Falls die Masche NULL ist, wird der Knoten dennoch erstellt. This method implements the SQL/MM specification. SQL-MM: Topo-Net Routines: X+1.3.1
- **ST_Area** - Returns the area of the surface if it is a Polygon or MultiPolygon. For geometry, a 2D Cartesian area is determined with units specified by the SRID. For geography, area is determined on a curved surface with units in square meters. This method implements the SQL/MM specification. SQL-MM 3: 8.1.2, 9.5.3
- **ST_AsBinary** - Gibt die Well-known-Binary(WKB)-Darstellung der Geometrie/Geographie ohne die SRID Metadaten zurück. This method implements the SQL/MM specification. SQL-MM 3: 5.1.37
- **ST_AsText** - Gibt die Well-known-Text(WKT) Darstellung der Geometrie/Geographie ohne die SRID Metadaten zurück. This method implements the SQL/MM specification. SQL-MM 3: 5.1.25
- **ST_Boundary** - Gibt die abgeschlossene Hülle aus der kombinierten Begrenzung der Geometrie zurück. This method implements the SQL/MM specification. SQL-MM 3: 5.1.14
- **ST_Buffer** - (T) Gibt eine Geometrie zurück, welche alle Punkte innerhalb einer gegebenen Entfernung von der Eingabegeometrie beinhaltet. This method implements the SQL/MM specification. SQL-MM 3: 5.1.17
- **ST_Centroid** - Gibt den geometrischen Schwerpunkt der Geometrie zurück. This method implements the SQL/MM specification. SQL-MM 3: 8.1.4, 9.5.5
- **ST_ChangeEdgeGeom** - Ändert die geometrische Form der Kante, ohne sich auf topologische Struktur auszuwirken. This method implements the SQL/MM specification. SQL-MM: Topo-Geo and Topo-Net 3: Routine Details X.3.6
- **ST_Contains** - Returns true if and only if no points of B lie in the exterior of A, and at least one point of the interior of B lies in the interior of A. This method implements the SQL/MM specification. SQL-MM 3: 5.1.31

- **ST_ConvexHull** - The convex hull of a geometry represents the minimum convex geometry that encloses all geometries within the set. This method implements the SQL/MM specification. SQL-MM 3: 5.1.16
 - **ST_CoordDim** - Gibt die Dimension der Koordinaten von ST_Geometry zurück. This method implements the SQL/MM specification. SQL-MM 3: 5.1.3
 - **ST_CreateTopoGeo** - Fügt eine Sammlung von Geometrien an eine leere Topologie an und gibt eine Bestätigungsmeldung aus. This method implements the SQL/MM specification. SQL-MM: Topo-Geo and Topo-Net 3: Routine Details -- X.3.18
 - **ST_Crosses** - Returns TRUE if the supplied geometries have some, but not all, interior points in common. This method implements the SQL/MM specification. SQL-MM 3: 5.1.29
 - **ST_CurveToLine** - Wandelt einen CIRCULARSTRING/CURVEPOLYGON in ein LINESTRING/POLYGON um This method implements the SQL/MM specification. SQL-MM 3: 7.1.7
 - **ST_Difference** - Gibt eine Geometrie zurück, die jenen Teil der Geometrie A abbildet, der sich nicht mit der Geometrie B überschneidet. This method implements the SQL/MM specification. SQL-MM 3: 5.1.20
 - **ST_Dimension** - Die inhärente Dimension des geometrischen Objekts muss niedriger oder gleich der Dimension der Koordinaten sein. This method implements the SQL/MM specification. SQL-MM 3: 5.1.2
 - **ST_Disjoint** - Gibt TRUE zurück, wenn sich die Geometrien nicht "räumlich schneiden" - wenn sie sich keinen gemeinsamen Raum teilen This method implements the SQL/MM specification. SQL-MM 3: 5.1.26
 - **ST_Distance** - For geometry type Returns the 2D Cartesian distance between two geometries in projected units (based on spatial ref). For geography type defaults to return minimum geodesic distance between two geographies in meters. This method implements the SQL/MM specification. SQL-MM 3: 5.1.23
 - **ST_EndPoint** - Gibt den Endpunkt einer LINESTRING oder CIRCULARLINESTRING Geometrie als POINT zurück. This method implements the SQL/MM specification. SQL-MM 3: 7.1.4
 - **ST_Envelope** - Gibt eine Geometrie in doppelter Genauigkeit (float8) zurück, welche das Umgebungsrechteck der beigegebenen Geometrie darstellt. This method implements the SQL/MM specification. SQL-MM 3: 5.1.15
 - **ST_Equals** - Returns true if the given geometries represent the same geometry. Directionality is ignored. This method implements the SQL/MM specification. SQL-MM 3: 5.1.24
 - **ST_ExteriorRing** - 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. Funktioniert nicht mit MULTIPOLYGON This method implements the SQL/MM specification. SQL-MM 3: 8.2.3, 8.3.3
 - **ST_GMLToSQL** - Gibt einen spezifizierten ST_Geometry Wert aus einer GML-Darstellung zurück. Dies ist ein Aliasname für ST_GeomFromGML This method implements the SQL/MM specification. SQL-MM 3: 5.1.50 (ausgenommen Unterstützung von Kurven).
 - **ST_GeomCollFromText** - Erzeugt eine Sammelgeometrie mit der gegebenen SRID aus einer WKT-Kollektion. Wenn keine SRID angegeben ist, wird diese standardmäßig auf 0 gesetzt. This method implements the SQL/MM specification.
 - **ST_GeomFromText** - Gibt einen spezifizierten ST_Geometry Wert aus einer Well-known-Text Darstellung (WKT) zurück. This method implements the SQL/MM specification. SQL-MM 3: 5.1.40
 - **ST_GeomFromWKB** - Erzeugt ein geometrisches Objekt aus der Well-known-Binary (WKB) Darstellung und einer optionalen SRID. This method implements the SQL/MM specification. SQL-MM 3: 5.1.41
 - **ST_GeometryFromText** - Gibt einen spezifizierten ST_Geometry-Wert von einer Well-known-Text Darstellung (WKT) zurück. Die Bezeichnung ist ein Alias für ST_GeomFromText This method implements the SQL/MM specification. SQL-MM 3: 5.1.40
 - **ST_GeometryN** - Gibt die auf 1-basierende n-te Geometrie zurück, wenn es sich bei der Geometrie um eine GEOMETRYCOLLECTION, (MULTI)POINT, (MULTI)LINESTRING, MULTICURVE oder (MULTI)POLYGON, POLYHEDRALSURFACE handelt. Anderenfalls wird NULL zurückgegeben. This method implements the SQL/MM specification. SQL-MM 3: 9.1.5
 - **ST_GeometryType** - Gibt den Geometrietyp des ST_Geometry Wertes zurück. This method implements the SQL/MM specification. SQL-MM 3: 5.1.4
-

- **ST_GetFaceEdges** - Gibt die Kanten, die aface begrenzen, sortiert aus. This method implements the SQL/MM specification. SQL-MM 3 Topo-Geo and Topo-Net 3: Routine Details: X.3.5
 - **ST_GetFaceGeometry** - Gibt für eine Topologie und eine bestimmte Maschen-ID das Polygon zurück. This method implements the SQL/MM specification. SQL-MM 3 Topo-Geo and Topo-Net 3: Routine Details: X.3.16
 - **ST_InitTopoGeo** - Erstellt ein neues topologisches Schema und registriert das neue Schema in der Tabelle topology.topology. Gibt eine Zusammenfassung des Prozessablaufs aus. This method implements the SQL/MM specification. SQL-MM 3 Topo-Geo and Topo-Net 3: Routine Details: X.3.17
 - **ST_InteriorRingN** - Gibt den Nten innenliegenden Linienzug des Ringes der Polygeometrie 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. This method implements the SQL/MM specification. SQL-MM 3: 8.2.6, 8.3.5
 - **ST_Intersection** - (T) Gibt eine Geometrie zurück, welche den gemeinsamen Anteil von geomA und geomB repräsentiert. This method implements the SQL/MM specification. SQL-MM 3: 5.1.18
 - **ST_Intersects** - Returns TRUE if the Geometries/Geography "spatially intersect in 2D" - (share any portion of space) and FALSE if they don't (they are Disjoint). For geography -- tolerance is 0.00001 meters (so any points that close are considered to intersect) This method implements the SQL/MM specification. SQL-MM 3: 5.1.27
 - **ST_IsClosed** - Gibt den Wert TRUE zurück, wenn die Anfangs- und Endpunkte des LINESTRING's zusammenfallen. Bei polyedrischen Oberflächen, wenn sie geschlossen (volumetrisch) sind. This method implements the SQL/MM specification. SQL-MM 3: 7.1.5, 9.3.3
 - **ST_IsEmpty** - Gibt den Wert TRUE zurück, falls es sich bei der Geometrie um eine leere GeometryCollection, Polygon, Point etc. handelt. This method implements the SQL/MM specification. SQL-MM 3: 5.1.7
 - **ST_IsRing** - Gibt den Wert TRUE zurück, wenn der LINESTRING geschlossen ist und der Simple Feature Spezifikation entspricht. This method implements the SQL/MM specification. SQL-MM 3: 7.1.6
 - **ST_IsSimple** - Gibt den Wert (TRUE) zurück, wenn die Geometrie keine irregulären Stellen, wie Selbstüberschneidungen oder Selbstberührungen, aufweist. This method implements the SQL/MM specification. SQL-MM 3: 5.1.8
 - **ST_IsValid** - Gibt true zurück, wenn ST_Geometry wohlgeformt ist. This method implements the SQL/MM specification. SQL-MM 3: 5.1.9
 - **ST_Length** - Returns the 2D length of the geometry if it is a LineString or MultiLineString. geometry are in units of spatial reference and geography are in meters (default spheroid) This method implements the SQL/MM specification. SQL-MM 3: 7.1.2, 9.3.4
 - **ST_LineFromText** - Erzeugt eine Geometrie aus einer WKT Darstellung mit der angegebenen SRID. Wenn keine SRID angegeben wird, wird diese standardmäßig auf 0 gesetzt. This method implements the SQL/MM specification. SQL-MM 3: 7.2.8
 - **ST_LineFromWKB** - Erzeugt einen LINESTRING mit gegebener SRID aus einer WKB-Darstellung This method implements the SQL/MM specification. SQL-MM 3: 7.2.9
 - **ST_LinestringFromWKB** - Erzeugt eine Geometrie mit gegebener SRID aus einer WKB-Darstellung. This method implements the SQL/MM specification. SQL-MM 3: 7.2.9
 - **ST_M** - Gibt die M-Koordinate eines Punktes, oder NULL, wenn diese nicht existiert, zurück. Bei der Eingabe muss es sich um eine Punktgeometrie handeln. This method implements the SQL/MM specification.
 - **ST_MLineFromText** - Liest einen festgelegten ST_MultiLineString Wert von einer WKT-Darstellung aus. This method implements the SQL/MM specification. SQL-MM 3: 9.4.4
 - **ST_MPointFromText** - Erzeugt eine Geometrie aus WKT mit der angegebenen SRID. Wenn keine SRID angegeben wird, wird diese standardmäßig auf 0 gesetzt. This method implements the SQL/MM specification. SQL-MM 3: 9.2.4
 - **ST_MPolyFromText** - Erzeugt eine MultiPolygon Geometrie aus WKT mit der angegebenen SRID. Wenn SRID nicht angegeben ist, wird sie standardmäßig auf 0 gesetzt. This method implements the SQL/MM specification. SQL-MM 3: 9.6.4
-

- **ST_ModEdgeHeal** - "Heilt" zwei Kanten, indem der verbindende Knoten gelöscht wird, die erste Kante modifiziert und die zweite Kante gelöscht wird. Gibt die ID des gelöschten Knoten zurück. This method implements the SQL/MM specification. SQL-MM: Topo-Geo and Topo-Net 3: Routine Details: X.3.9
- **ST_ModEdgeSplit** - Trennt eine Kante auf, indem ein neuer Knoten entlang einer bestehenden Kante erstellt wird. Ändert die ursprüngliche Kante und fügt eine neue Kante hinzu. This method implements the SQL/MM specification. SQL-MM: Topo-Geo and Topo-Net 3: Routine Details: X.3.9
- **ST_MoveIsoNode** - Verschiebt einen isolierten Knoten in einer Topologie von einer Stelle an eine andere. Falls die neue Geometrie apoint bereits als Knoten existiert, wird eine Fehlermeldung ausgegeben. Gibt eine Beschreibung der Verschiebung aus. This method implements the SQL/MM specification. SQL-MM: Topo-Net Routines: X.3.2
- **ST_NewEdgeHeal** - "Heilt" zwei Kanten, indem der verbindende Knoten und beide Kanten gelöscht werden. Die beiden Kanten werden durch eine Kante ersetzt, welche dieselbe Ausrichtung wie die erste Kante hat. This method implements the SQL/MM specification. SQL-MM: Topo-Geo and Topo-Net 3: Routine Details: X.3.9
- **ST_NewEdgesSplit** - Trennt eine Kante auf, indem ein neuer Knoten entlang einer bestehenden Kante erstellt, die ursprüngliche Kante gelöscht und durch zwei neue Kanten ersetzt wird. Gibt die ID des neu erstellten Knotens aus, der die neuen Kanten verbindet. This method implements the SQL/MM specification. SQL-MM: Topo-Net Routines: X.3.8
- **ST_NumGeometries** - Wenn es sich bei der Geometrie um eine GEOMETRYCOLLECTION (oder MULTI*) handelt, wird die Anzahl der Geometrien zurückgegeben, bei Einzelgeometrien wird 1, ansonsten NULL zurückgegeben. This method implements the SQL/MM specification. SQL-MM 3: 9.1.4
- **ST_NumInteriorRings** - Gibt die Anzahl der inneren Ringe einer Polygoneometrie aus. This method implements the SQL/MM specification. SQL-MM 3: 8.2.5
- **ST_NumPatches** - Gibt die Anzahl der Maschen einer polyedrischen Oberfläche aus. Gibt NULL zurück, wenn es sich nicht um polyedrische Geometrien handelt. This method implements the SQL/MM specification. SQL-MM 3: ?
- **ST_NumPoints** - Gibt die Anzahl der Stützpunkte eines ST_LineString oder eines ST_CircularString zurück. This method implements the SQL/MM specification. SQL-MM 3: 7.2.4
- **ST_OrderingEquals** - Returns true if the given geometries represent the same geometry and points are in the same directional order. This method implements the SQL/MM specification. SQL-MM 3: 5.1.43
- **ST_Overlaps** - Returns TRUE if the Geometries share space, are of the same dimension, but are not completely contained by each other. This method implements the SQL/MM specification. SQL-MM 3: 5.1.32
- **ST_PatchN** - Gibt die auf 1-basierende n-te Geometrie (Masche) zurück, wenn es sich bei der Geometrie um ein POLYHEDRALSURFACE, oder ein POLYHEDRALSURFACEM handelt. Anderenfalls wird NULL zurückgegeben. This method implements the SQL/MM specification. SQL-MM 3: ?
- **ST_Perimeter** - Return the length measurement of the boundary of an ST_Surface or ST_MultiSurface geometry or geography. (Polygon, MultiPolygon). geometry measurement is in units of spatial reference and geography is in meters. This method implements the SQL/MM specification. SQL-MM 3: 8.1.3, 9.5.4
- **ST_Point** - Gibt einen ST_Point mit den gegebenen Koordinatenwerten aus. Ein OGC-Alias für ST_MakePoint. This method implements the SQL/MM specification. SQL-MM 3: 6.1.2
- **ST_PointFromText** - Erzeugt eine Punktgeometrie mit gegebener SRID von WKT. Wenn SRID nicht angegeben ist, wird sie standardmäßig auf 0 gesetzt. This method implements the SQL/MM specification. SQL-MM 3: 6.1.8
- **ST_PointFromWKB** - Erzeugt eine Geometrie mit gegebener SRID von WKB. This method implements the SQL/MM specification. SQL-MM 3: 6.1.9
- **ST_PointN** - Gibt den n-ten Punkt des ersten LineString's oder des kreisförmigen LineStrings's in der Geometrie zurück. Negative Werte werden rückwärts vom Ende des LineString's gezählt. Gibt NULL aus, wenn es sich bei der Geometrie nicht um einen LineString handelt. This method implements the SQL/MM specification. SQL-MM 3: 7.2.5, 7.3.5

- **ST_PointOnSurface** - Returns a POINT guaranteed to lie on the surface. This method implements the SQL/MM specification. SQL-MM 3: 8.1.5, 9.5.6. According to the specs, ST_PointOnSurface works for surface geometries (POLYGONS, MULTIPOLYGONS, CURVED POLYGONS). So PostGIS seems to be extending what the spec allows here. Most databases Oracle, DB II, ESRI SDE seem to only support this function for surfaces. SQL Server 2008 like PostGIS supports for all common geometries.
 - **ST_Polygon** - Gibt ein Polygon zurück, das aus vorgegebenen Linienzug und SRID erzeugt wurde. This method implements the SQL/MM specification. SQL-MM 3: 8.3.2
 - **ST_PolygonFromText** - Erzeugt eine Geometrie aus WKT mit der angegebenen SRID. Wenn keine SRID angegeben wird, wird diese standardmäßig auf 0 gesetzt. This method implements the SQL/MM specification. SQL-MM 3: 8.3.6
 - **ST_Relate** - Returns true if this Geometry is spatially related to another Geometry, by testing for intersections between the Interior, Boundary and Exterior of the two geometries as specified by the values in the intersectionMatrixPattern. If no intersectionMatrixPattern is passed in, then returns the maximum intersectionMatrixPattern that relates the 2 geometries. This method implements the SQL/MM specification. SQL-MM 3: 5.1.25
 - **ST_RemEdgeModFace** - Entfernt eine Kante. Falls die gelöschte Kante zwei Maschen voneinander getrennt hat, wird eine der Maschen gelöscht und die andere so geändert, dass sie den Platz der beiden ursprünglichen Maschen einnimmt. This method implements the SQL/MM specification. SQL-MM: Topo-Geo and Topo-Net 3: Routine Details: X.3.15
 - **ST_RemEdgeNewFace** - Entfernt eine Kante. Falls die gelöschte Kante zwei Maschen voneinander getrennt hat, werden die ursprünglichen Maschen gelöscht und durch einer neuen Masche ersetzt. This method implements the SQL/MM specification. SQL-MM: Topo-Geo and Topo-Net 3: Routine Details: X.3.14
 - **ST_RemoveIsoEdge** - Removes an isolated edge and returns description of action. If the edge is not isolated, then an exception is thrown. This method implements the SQL/MM specification. SQL-MM: Topo-Geo and Topo-Net 3: Routine Details: X+1.3.3
 - **ST_RemoveIsoNode** - Löscht einen isolierten Knoten und gibt eine Beschreibung der getroffenen Maßnahmen aus. Falls der Knoten nicht isoliert ist (ist der Anfangs- oder der Endpunkt einer Kante), wird eine Fehlermeldung ausgegeben. This method implements the SQL/MM specification. SQL-MM: Topo-Geo and Topo-Net 3: Routine Details: X+1.3.3
 - **ST_SRID** - Gibt den Identifikator des Koordinatenreferenzsystems, wie in der spatial_ref_sys Tabelle definiert, für die ST_Geometry aus. This method implements the SQL/MM specification. SQL-MM 3: 5.1.5
 - **ST_StartPoint** - Gibt den den Anfangspunkt einer LINESTRING Geometrie als POINT zurück. This method implements the SQL/MM specification. SQL-MM 3: 7.1.3
 - **ST_SymDifference** - Gibt eine Geometrie zurück, die jene Teile von A und B repräsentiert, die sich nicht überlagern. Wird symmetrische Differenz genannt, da $ST\_SymDifference(A,B) = ST\_SymDifference(B,A)$. This method implements the SQL/MM specification. SQL-MM 3: 5.1.21
 - **ST_Touches** - Returns TRUE if the geometries have at least one point in common, but their interiors do not intersect. This method implements the SQL/MM specification. SQL-MM 3: 5.1.28
 - **ST_Transform** - Gibt eine neue Geometrie zurück, wobei eine Koordinatentransformation in ein anderes räumliches Bezugssystem durchgeführt wird. This method implements the SQL/MM specification. SQL-MM 3: 5.1.6
 - **ST_Union** - Gibt eine Geometrie zurück, welche der mengentheoretischen Vereinigung der Geometrien entspricht. This method implements the SQL/MM specification. SQL-MM 3: 5.1.19 der Z-Index (Höhe) wenn Polygone beteiligt sind.
 - **ST_WKBToSQL** - Gibt einen geometrischen Datentyp (ST_Geometry) aus einer Well-known-Binary (WKB) Darstellung zurück. Ein Synonym für ST_GeomFromWKB, welches jedoch keine SRID annimmt. This method implements the SQL/MM specification. SQL-MM 3: 5.1.36
 - **ST_WKTToSQL** - Gibt einen spezifizierten ST_Geometry-Wert von einer Well-known-Text Darstellung (WKT) zurück. Die Bezeichnung ist ein Alias für ST_GeomFromText. This method implements the SQL/MM specification. SQL-MM 3: 5.1.34
 - **ST_Within** - Gibt TRUE zurück, wenn Geometrie A zur Gänze innerhalb von Geometrie B liegt. This method implements the SQL/MM specification. SQL-MM 3: 5.1.30
-

- **ST_X** - Gibt die X-Koordinate eines Punktes, oder NULL wenn diese nicht vorhanden ist, zurück. Die Eingabe muss ein Punkt sein. This method implements the SQL/MM specification. SQL-MM 3: 6.1.3
- **ST_Y** - Gibt die Y-Koordinate eines Punktes, oder NULL wenn diese nicht vorhanden ist, zurück. Die Eingabe muss ein Punkt sein. This method implements the SQL/MM specification. SQL-MM 3: 6.1.4
- **ST_Z** - Gibt die Z-Koordinate eines Punktes, oder NULL wenn diese nicht vorhanden ist, zurück. Die Eingabe muss ein Punkt sein. This method implements the SQL/MM specification.

14.4 PostGIS Geography Support Functions

The functions and operators given below are PostGIS functions/operators that take as input or return as output a **geography** data type object.



Note

Functions with a (T) are not native geodetic functions, and use a ST_Transform call to and from geometry to do the operation. As a result, they may not behave as expected when going over dateline, poles, and for large geometries or geometry pairs that cover more than one UTM zone. Basic transform - (favoring UTM, Lambert Azimuthal (North/South), and falling back on mercator in worst case scenario)

- **ST_Area** - Returns the area of the surface if it is a Polygon or MultiPolygon. For geometry, a 2D Cartesian area is determined with units specified by the SRID. For geography, area is determined on a curved surface with units in square meters.
- **ST_AsBinary** - Gibt die Well-known-Binary(WKB)-Darstellung der Geometrie/Geographie ohne die SRID Metadaten zurück.
- **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** - Gibt die Geometrie eines GeoJSON Elements zurück.
- **ST_AsKML** - Gibt die Geometrie als ein KML Element aus. Mehrere Varianten. Standardmäßig ist version=2 und precision=15
- **ST_AsSVG** - Gibt ein Geoobjekt als SVG-Pfadgeometrie zurück. Unterstützt den geometrischen und den geographischen Datentyp.
- **ST_AsText** - Gibt die Well-known-Text(WKT) Darstellung der Geometrie/Geographie ohne die SRID Metadaten zurück.
- **ST_Azimuth** - Returns the north-based azimuth as the angle in radians measured clockwise from the vertical on pointA to pointB.
- **ST_Buffer** - (T) Gibt eine Geometrie zurück, welche alle Punkte innerhalb einer gegebenen Entfernung von der Eingabegeometrie beinhaltet.
- **ST_Centroid** - Gibt den geometrischen Schwerpunkt der Geometrie zurück.
- **ST_CoveredBy** - Gibt 1 (TRUE) zurück, falls kein Punkt der Geometrie/Geographie A außerhalb von Geometry/Geographie B liegt
- **ST_Covers** - Gibt 1 (TRUE) zurück, falls kein Punkt der Geometrie B außerhalb von Geometry A liegt
- **ST_DWithin** - Returns true if the geometries are within the specified distance of one another. For geometry units are in those of spatial reference and For geography units are in meters and measurement is defaulted to use_spheroid=true (measure around spheroid), for faster check, use_spheroid=false to measure along sphere.
- **ST_Distance** - For geometry type Returns the 2D Cartesian distance between two geometries in projected units (based on spatial ref). For geography type defaults to return minimum geodesic distance between two geographies in meters.

- **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.
- **=** - Returns TRUE if the coordinates and coordinate order geometry/geography A are the same as the coordinates and coordinate order of geometry/geography B.
- **ST_Intersection** - (T) Gibt eine Geometrie zurück, welche den gemeinsamen Anteil von geomA und geomB repräsentiert.
- **ST_Intersects** - Returns TRUE if the Geometries/Geography "spatially intersect in 2D" - (share any portion of space) and FALSE if they don't (they are Disjoint). For geography -- tolerance is 0.00001 meters (so any points that close are considered to intersect)
- **ST_Length** - Returns the 2D length of the geometry if it is a LineString or MultiLineString. geometry are in units of spatial reference and geography are in meters (default spheroid)
- **ST_Perimeter** - Return the length measurement of the boundary of an ST_Surface or ST_MultiSurface geometry or geography. (Polygon, MultiPolygon). geometry measurement is in units of spatial reference and geography is in meters.
- **ST_Project** - Returns a POINT projected from a start point using a distance in meters and bearing (azimuth) in radians.
- **ST_Segmentize** - Gibt eine veränderte Geometrie/Geographie zurück, bei der kein Segment länger als der gegebene Abstand ist.
- **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.

14.5 PostGIS Raster Support Functions

The functions and operators given below are PostGIS functions/operators that take as input or return as output a **raster** data type object. Listed in alphabetical order.

- **Box3D** - Returns the box 3d representation of the enclosing box of the raster.
- **@** - Returns TRUE if A's bounding box is contained by B's. Uses double precision bounding box.
- **~** - Returns TRUE if A's bounding box is contains B's. Uses double precision bounding box.
- **=** - Returns TRUE if A's bounding box is the same as B's. Uses double precision bounding box.
- **&&** - Returns TRUE if A's bounding box intersects B's bounding box.
- **&<** - Returns TRUE if A's bounding box is to the left of B's.
- **&>** - Returns TRUE if A's bounding box is to the right of B's.
- **~=** - Gibt TRUE zurück wenn die Umgebungsrechtecke von "A" und "B" ident sind.
- **ST_Retile** - Gibt konfigurierte Kacheln eines beliebig gekachelten Rastercoverage aus.
- **ST_AddBand** - Gibt einen Raster mit den neu hinzugefügten Band(Bändern) aus. Der Typ , der Ausgangswert und der Index für den Speicherort des Bandes kann angegeben werden. Wenn kein Index angegeben ist, wird das Band am Ende hinzugefügt.
- **ST_AsBinary** - Gibt die Well-known-Binary (WKB) Darstellung eines Rasters zurück.

- **ST_AsGDALRaster** - Return the raster tile in the designated GDAL Raster format. Raster formats are one of those supported by your compiled library. Use ST_GDALDrivers() to get a list of formats supported by your library.
 - **ST_AsJPEG** - Return the raster tile selected bands as a single Joint Photographic Exports Group (JPEG) image (byte array). If no band is specified and 1 or more than 3 bands, then only the first band is used. If only 3 bands then all 3 bands are used and mapped to RGB.
 - **ST_AsPNG** - Return the raster tile selected bands as a single portable network graphics (PNG) image (byte array). If 1, 3, or 4 bands in raster and no bands are specified, then all bands are used. If more 2 or more than 4 bands and no bands specified, then only band 1 is used. Bands are mapped to RGB or RGBA space.
 - **ST_AsRaster** - Konvertiert den geometrischen Datentyp von PostGIS in einen PostGIS Raster.
 - **ST_AsTIFF** - Return the raster selected bands as a single TIFF image (byte array). If no band is specified, then will try to use all bands.
 - **ST_Aspect** - Returns the aspect (in degrees by default) of an elevation raster band. Useful for analyzing terrain.
 - **ST_Band** - Gibt einen oder mehrere Bänder eines bestehenden Rasters als neuen Raster aus. Nützlich um neue Raster aus bestehenden Rastern abzuleiten.
 - **ST_BandIsNoData** - Gibt TRUE aus, wenn das Band ausschließlich aus NODATA Werten besteht.
 - **ST_BandMetaData** - Gibt die grundlegenden Metadaten eines bestimmten Rasterbandes aus. Wenn der Parameter "bandnum" nicht angegeben ist, wird das 1ste Band angenommen.
 - **ST_BandNoDataValue** - Gibt den NODATA Wert des gegebenen Bandes aus. Wenn der Parameter "bandnum" nicht angegeben ist, wird das 1ste Band angenommen.
 - **ST_BandPath** - Gibt den Dateipfad aus, unter dem das Band im Dateisystem gespeichert ist. Wenn "bandnum" nicht angegeben ist, wird 1 angenommen.
 - **ST_BandPixelType** - Gibt den Pixeltyp des angegebenen Bandes aus. Wenn der Parameter "bandnum" nicht angegeben ist, wird das 1ste Band angenommen.
 - **ST_Clip** - Schneidet den Raster nach der Eingabegeometrie. Wenn die Bandnummer nicht angegeben ist, werden alle Bänder bearbeitet. Wenn crop nicht angegeben oder TRUE ist, wird der Ausgaberraster abgeschnitten.
 - **ST_ColorMap** - Erzeugt aus einem bestimmten Band des Ausgangsrasters einen neuen Raster mit bis zu vier 8BUI-Bändern (Grauwert, RGB, RGBA). Wenn kein Band angegeben ist, wird Band 1 angenommen.
 - **ST_Contains** - Return true if no points of raster rastB lie in the exterior of raster rastA and at least one point of the interior of rastB lies in the interior of rastA.
 - **ST_ContainsProperly** - Gibt TRUE zurück, wenn "rastB" das Innere von "rastA" schneidet, aber nicht die Begrenzung oder das Äußere von "rastA".
 - **ST_ConvexHull** - Return the convex hull geometry of the raster including pixel values equal to BandNoDataValue. For regular shaped and non-skewed rasters, this gives the same result as ST_Envelope so only useful for irregularly shaped or skewed rasters.
 - **ST_Count** - 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** - Returns a set of geomval (geom,val) rows, from a given raster band. If no band number is specified, band num defaults to 1.
 - **ST_DumpValues** - Gibt die Werte eines bestimmten Bandes als 2-dimensionales Feld aus.
 - **ST_Envelope** - Returns the polygon representation of the extent of the raster.
 - **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_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** - Returns the hypothetical illumination of an elevation raster band using provided azimuth, altitude, brightness and scale inputs.
 - **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_Intersection** - Gibt Geometry-PixelValue Paare, oder einen Raster aus, der durch die Schnittmenge der beiden Raster bestimmt wird, oder durch die geometrische Verschneidung einer Vektorisierung des Rasters mit einem geometrischen Datentyp.
 - **ST_Intersects** - Gibt TRUE zurück, wenn sich die Raster "rastA" und "rastB" nicht räumlich überschneiden.
 - **ST_IsEmpty** - Gibt TRUE zurück, wenn der Raster leer ist (width = 0 and height = 0). Andernfalls wird FALSE zurückgegeben.
 - **ST_MakeEmptyCoverage** - Bedeckt die georeferenzierte Fläche mit einem Gitter aus leeren Rasterkacheln.
 - **ST_MakeEmptyRaster** - Gibt einen leeren Raster (ohne Bänder), mit den gegebenen Dimensionen (width & height), upperleft X und Y, Pixelgröße, Rotation (scalex, scaley, skewx & skewy) und Koordinatenreferenzsystem (SRID), zurück. Wenn ein Raster übergeben wird, dann wird ein neuer Raster mit der selben Größe, Ausrichtung und SRID zurückgegeben. Wenn SRID nicht angegeben ist, wird das Koordinatenreferenzsystem auf "unknown" (0) gesetzt.
 - **ST_MapAlgebra** - 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** - 1 raster band version: Creates a new one band raster formed by applying a valid PostgreSQL algebraic operation on the input raster band and of pixeltype provided. Band 1 is assumed if no band is specified.
 - **ST_MapAlgebraExpr** - 2 raster band version: Creates a new one band raster formed by applying a valid PostgreSQL algebraic operation on the two input raster bands and of pixeltype provided. band 1 of each raster is assumed if no band numbers are specified. The resulting raster will be aligned (scale, skew and pixel corners) on the grid defined by the first raster and have its extent defined by the "extenttype" parameter. Values for "extenttype" can be: INTERSECTION, UNION, FIRST, SECOND.
 - **ST_MapAlgebraFct** - 1 band version - Creates a new one band raster formed by applying a valid PostgreSQL function on the input raster band and of pixeltype provided. Band 1 is assumed if no band is specified.
 - **ST_MapAlgebraFct** - 2 band version - Creates a new one band raster formed by applying a valid PostgreSQL function on the 2 input raster bands and of pixeltype provided. Band 1 is assumed if no band is specified. Extent type defaults to INTERSECTION if not specified.
 - **ST_MapAlgebraFctNgb** - 1-band version: Map Algebra Nearest Neighbor using user-defined PostgreSQL function. Return a raster which values are the result of a PLPGSQL user function involving a neighborhood of values from the input raster band.
-

- **ST_MapAlgebra** - Expression version - Returns a one-band raster given one or two input rasters, band indexes and one or more user-specified SQL expressions.
 - **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** - Return the convex hull geometry of the raster excluding NODATA pixels.
 - **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 Polygoneometrie aus, die das Pixel einer bestimmten Zeile und Spalte begrenzt.
 - **ST_PixelAsPolygons** - Gibt die umhüllende Polygoneometrie, 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** - Returns a multipolygon geometry formed by the union of pixels that have a pixel value that is not no data value. If no band number is specified, band num defaults to 1.
 - **ST_Quantile** - 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_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** - Creates a new raster composed of band types reclassified from original. The nband is the band to be changed. If nband is not specified assumed to be 1. All other bands are returned unchanged. Use case: convert a 16BUI band to a 8BUI and so forth for simpler rendering as viewable formats.
-

- **ST_Resample** - 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** - Skaliert einen Raster indem lediglich der Maßstab (oder die Pixelgröße) angepasst wird. Neue Pixelwerte werden über NearestNeighbor, bilinear, kubisch, CubicSpline oder mit dem Lanczos-Filter errechnet. Die Standardeinstellung 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** - Returns a raster with the calculated "roughness" of a DEM.
 - **ST_SRID** - Gibt den Identifikator des Koordinatenreferenzsystems des Rasters aus, das in der Tabelle "spatial_ref_sys" definiert ist.
 - **ST_SameAlignment** - Returns true if rasters have same skew, scale, spatial ref, and offset (pixels can be put on same grid without cutting into pixels) and false if they don't with notice detailing issue.
 - **ST_ScaleX** - 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_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_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_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 linken 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_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** - Returns the slope (in degrees by default) of an elevation raster band. Useful for analyzing terrain.
 - **ST_SnapToGrid** - Skaliert einen Raster durch Fangen an einem Führungsgitter. Neue Pixelwerte werden über NearestNeighbor, bilinear, kubisch, CubicSpline oder mit dem Lanczos-Filter errechnet. Die Standardeinstellung ist NearestNeighbor.
-

- **ST_Summary** - Gibt eine textliche Zusammenfassung des Rasterinhalts zurück.
- **ST_SummaryStats** - Gibt eine zusammenfassende Statistik aus, bestehend aus der Anzahl, der Summe, dem arithmetischen Mittel, der Standardabweichung, dem Minimum und dem Maximum der Werte eines Rasterbandes oder eines Rastercoverage. Wenn kein Band angegeben ist, wird Band 1 angenommen.
- **ST_SummaryStatsAgg** - Aggregatfunktion. Gibt eine zusammenfassende Statistik aus, die aus der Anzahl, der Summe, dem arithmetischen Mittel, dem Minimum und dem Maximum der Werte eines bestimmten Bandes eines Rastersatzes besteht. Wenn kein Band angegeben ist, wird Band 1 angenommen.
- **ST_TPI** - Returns a raster with the calculated Topographic Position Index.
- **ST_TRI** - Returns a raster with the calculated Terrain Ruggedness Index.
- **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 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.
- **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.

14.6 PostGIS Geometry / Geography / Raster Dump Functions

The functions given below are PostGIS functions that take as input or return as output a set of or single **geometry_dump** or **geomval** data type object.

- **ST_DumpAsPolygons** - Returns a set of geomval (geom,val) rows, from a given raster band. If no band number is specified, band num defaults to 1.

- **ST_Intersection** - 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 an geometry_dump (geom,path) Zeilen zurück, aus denen die Geometrie g1 zusammengesetzt ist.
- **ST_DumpPoints** - Gibt eine Menge von geometry_dump (geom,path) Zeilen aller Punkte zurück, aus denen die Geometrie zusammengesetzt ist.
- **ST_DumpRings** - Gibt eine Menge an geometry_dump Zeilen zurück, welche die äußeren und inneren Ringe eines Polygons darstellen.

14.7 PostGIS Box Functions

The functions given below are PostGIS functions that take as input or return as output the box* family of PostGIS spatial types. The box family of types consists of **box2d**, and **box3d**

- **Box2D** - Returns a BOX2D representing the maximum extents of the geometry.
- **Box3D** - Returns a BOX3D representing the maximum extents of the geometry.
- **Box3D** - Returns the box 3d representation of the enclosing box of the raster.
- **ST_3DExtent** - an aggregate function that returns the box3D bounding box that bounds rows of geometries.
- **ST_3DMakeBox** - Erzeugt eine BOX3D, die durch die gegebene 3D-Punktgeometrie definiert ist.
- **ST_AsMVTGeom** - Transformiert eine Geometrie in das Koordinatensystem eines Mapbox Vector Tiles.
- **ST_AsTWKB** - Gibt die Geometrie als TWKB, aka "Tiny Well-known Binary" zurück
- **ST_Box2dFromGeoHash** - Gibt die BOX2D einer GeoHash Zeichenkette zurück.
- **ST_ClipByBox2D** - Gibt jenen Teil der Geometrie zurück, der innerhalb eines Rechteckes liegt.
- **ST_EstimatedExtent** - Return the 'estimated' extent of the given spatial table. The estimated is taken from the geometry column's statistics. The current schema will be used if not specified.
- **ST_Expand** - Returns bounding box expanded in all directions from the bounding box of the input geometry. Uses double-precision
- **ST_Extent** - an aggregate function that returns the bounding box that bounds rows of geometries.
- **ST_MakeBox2D** - Erzeugt eine BOX2D, die durch die gegebene Punktgeometrie definiert ist.
- **ST_XMax** - Gibt das größte X des Umgebungsrechtecks in 2D, 3D oder als Geometrie zurück.
- **ST_XMin** - Gibt das kleinste X des Umgebungsrechtecks in 2D, 3D oder als Geometrie zurück.
- **ST_YMax** - Gibt das größte Y des Umgebungsrechtecks in 2D, 3D oder als Geometrie zurück.
- **ST_YMin** - Gibt das kleinste Y des Umgebungsrechtecks in 2D, 3D oder als Geometrie zurück.
- **ST_ZMax** - Gibt das kleinste Z des Umgebungsrechtecks in 2D, 3D oder als Geometrie zurück.
- **ST_ZMin** - Gibt das kleinste Z des Umgebungsrechtecks in 2D, 3D oder als Geometrie zurück.
- **~(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 (BOX2DF) 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.

14.8 PostGIS Functions that support 3D

The functions given below are PostGIS functions that do not throw away the Z-Index.

- **AddGeometryColumn** - Fügt eine Geometriespalte an eine bestehende Tabelle an. Standardmäßig wird Typmodifikation anstelle von Bedingungen/Constraints verwendet. Sie erreichen das alte, auf check constraints basierte Verhalten, wenn Sie use_typmod auf false setzen.
- **Box3D** - Returns a BOX3D representing the maximum extents of the geometry.
- **DropGeometryColumn** - Entfernt eine Geometriespalte aus einer räumlichen Tabelle.
- **GeometryType** - Gibt den Geometrietyp als Zeichenkette zurück. z.B.: 'LINESTRING', 'POLYGON', 'MULTIPOINT', etc.
- **ST_3DArea** - Berechnet die Fläche von 3D-Oberflächengeometrien. Gibt 0 für Solids zurück.
- **ST_3DClosestPoint** - Returns the 3-dimensional point on g1 that is closest to g2. This is the first point of the 3D shortest line.
- **ST_3DDFullyWithin** - Returns true if all of the 3D geometries are within the specified distance of one another.
- **ST_3DDWithin** - For 3d (z) geometry type Returns true if two geometries 3d distance is within number of units.
- **ST_3DDifference** - Errechnet die Differenzmenge in 3D
- **ST_3DDistance** - For geometry type Returns the 3-dimensional cartesian minimum distance (based on spatial ref) between two geometries in projected units.
- **ST_3DExtent** - an aggregate function that returns the box3D bounding box that bounds rows of geometries.
- **ST_3DIntersection** - Führt eine Verschneidung in 3D aus
- **ST_3DIntersects** - Returns TRUE if the Geometries "spatially intersect" in 3d - only for points, linestrings, polygons, polyhedral surface (area). With SFCGAL backend enabled also supports TINS
- **ST_3DLength** - Returns the 3-dimensional or 2-dimensional length of the geometry if it is a linestring or multi-linestring.
- **ST_3DLongestLine** - Returns the 3-dimensional longest line between two geometries
- **ST_3DMaxDistance** - For geometry type Returns the 3-dimensional cartesian maximum distance (based on spatial ref) between two geometries in projected units.
- **ST_3DPerimeter** - Returns the 3-dimensional perimeter of the geometry, if it is a polygon or multi-polygon.
- **ST_3DShortestLine** - Gibt die 3-dimensionale kürzeste Strecke zwischen zwei Geometrien zurück
- **ST_3DUnion** - Führt eine Vereinigung/Union in 3D aus

- **ST_Accum** - Aggregatfunktion. Erzeugt ein Feld mit Geometrien.
 - **ST_AddMeasure** - Return a derived geometry with measure elements linearly interpolated between the start and end points.
 - **ST_AddPoint** - Fügt einem Linienzug einen Punkt hinzu.
 - **ST_Affine** - Wendet eine affine 3D-Transformation auf die Geometrie an.
 - **ST_ApproximateMedialAxis** - Errechnet die genäherte Mediale Achse einer Flächengeometrie.
 - **ST_AsBinary** - Gibt die Well-known-Binary(WKB)-Darstellung der Geometrie/Geographie ohne die SRID Metadaten zurück.
 - **ST_AsEWKB** - Gibt die Well-known-Binary(WKB)-Darstellung der Geometrie mit den SRID Metadaten zurück.
 - **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** - Gibt die Geometrie eines GeoJSON Elements 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_AsKML** - Gibt die Geometrie als ein KML Element aus. Mehrere Varianten. Standardmäßig ist version=2 und precision=15
 - **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** - Returns true if the trajectories' closest points of approach are within the specified distance.
 - **ST_ClosestPointOfApproach** - Returns the measure at which points interpolated along two lines are closest.
 - **ST_Collect** - Gibt einen festgelegten ST_Geometry Wert aus einer Sammlung anderer Geometrien zurück.
 - **ST_ConvexHull** - The convex hull of a geometry represents the minimum convex geometry that encloses all geometries within the set.
 - **ST_CoordDim** - Gibt die Dimension der Koordinaten von ST_Geometry zurück.
 - **ST_CurveToLine** - Wandelt einen CIRCULARSTRING/CURVEPOLYGON in ein LINESTRING/POLYGON um
 - **ST_DelaunayTriangles** - Gibt die Delaunay-Triangulierung für gegebene Punkte zurück.
 - **ST_Difference** - Gibt eine Geometrie zurück, die jenen Teil der Geometrie A abbildet, der sich nicht mit der Geometrie B überschneidet.
 - **ST_DistanceCPA** - Returns the distance between closest points of approach in two trajectories.
 - **ST_Dump** - Gibt einen Satz an geometry_dump (geom,path) Zeilen zurück, aus denen die Geometrie g1 zusammengesetzt ist.
 - **ST_DumpPoints** - Gibt eine Menge von geometry_dump (geom,path) Zeilen aller Punkte zurück, aus denen die Geometrie zusammengesetzt ist.
 - **ST_DumpRings** - Gibt eine Menge an geometry_dump Zeilen zurück, welche die äußeren und inneren Ringe eines Polygons darstellen.
 - **ST_EndPoint** - Gibt den Endpunkt einer LINESTRING oder CIRCULARLINESTRING Geometrie als POINT zurück.
 - **ST_ExteriorRing** - 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. Funktioniert nicht mit MULTIPOLYGON
 - **ST_Extrude** - Weitet eine Oberfläche auf ein entsprechendes Volumen aus
-

- **ST_FlipCoordinates** - Gibt eine Version der gegebenen Geometrie zurück, wobei die X und Y Achse vertauscht sind. Nützlich wenn man Geoobjekte in Breite/Länge vorliegen hat und dies beheben möchte.
 - **ST_Force2D** - Die Geometrien in einen "2-dimensionalen Modus" zwingen.
 - **ST_ForceCurve** - Wandelt einen geometrischen in einen Kurven Datentyp um, soweit anwendbar.
 - **ST_ForceLHR** - Erzwingt LHR Orientierung
 - **ST_ForcePolygonCCW** - Richtet alle äußeren Ringe gegen den Uhrzeigersinn und alle inneren Ringe mit dem Uhrzeigersinn aus.
 - **ST_ForcePolygonCW** - Richtet alle äußeren Ringe im Uhrzeigersinn und alle inneren Ringe gegen den Uhrzeigersinn aus.
 - **ST_ForceRHR** - Orientiert die Knoten in einem Polygon so, dass sie der Drei-Finger-Regel folgen.
 - **ST_ForceSFS** - Erzwingt, dass Geometrien nur vom Typ SFS 1.1 sind.
 - **ST_Force_3D** - Zwingt die Geometrien in einen XYZ Modus. Dies ist ein Alias für ST_Force3DZ.
 - **ST_Force_3DZ** - Zwingt die Geometrien in einen XYZ Modus.
 - **ST_Force_4D** - Zwingt die Geometrien in einen XYZM Modus.
 - **ST_Force_Collection** - Wandelt eine Geometrie in eine GEOMETRYCOLLECTION um.
 - **ST_GeomFromEWKB** - Gibt einen geometrischen Datentyp (ST_Geometry) aus einer Well-known-Binary (WKB) Darstellung zurück.
 - **ST_GeomFromEWKT** - Gibt einen spezifizierten ST_Geometry-Wert von einer erweiterten Well-known-Text Darstellung (EWKT) zurück.
 - **ST_GeomFromGML** - Nimmt als Eingabe eine GML-Darstellung der Geometrie und gibt ein geometrisches PostGIS-Objekt aus.
 - **ST_GeomFromGeoJSON** - Nimmt als Eingabe eine GeoJSON-Darstellung der Geometrie und gibt ein geometrisches PostGIS-Objekt aus.
 - **ST_GeomFromKML** - Nimmt als Eingabe eine KML-Darstellung der Geometrie und gibt ein geometrisches PostGIS-Objekt aus.
 - **ST_GeometricMedian** - Returns the geometric median of a MultiPoint.
 - **ST_GeometryN** - Gibt die auf 1-basierende n-te Geometrie zurück, wenn es sich bei der Geometrie um eine GEOMETRYCOLLECTION, (MULTI)POINT, (MULTI)LINestring, MULTICURVE oder (MULTI)POLYGON, POLYHEDRALSURFACE handelt. Anderenfalls wird NULL zurückgegeben.
 - **ST_GeometryType** - Gibt den Geometrietyp des ST_Geometry Wertes zurück.
 - **ST_HasArc** - Returns true if a geometry or geometry collection contains a circular string
 - **ST_InteriorRingN** - Gibt den Nten innenliegenden Linienzug des Ringes der Polygoneometrie 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.
 - **ST_InterpolatePoint** - Return the value of the measure dimension of a geometry at the point closed to the provided point.
 - **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, wenn der Parameter eine Ansammlung/Kollektion von Geometrien ist (MULTI*, GEOMETRYCOLLECTION, ...)
 - **ST_IsPlanar** - Überprüft ob es sich um eine ebene Oberfläche handelt oder nicht
-

- **ST_IsPolygonCCW** - Gibt TRUE zurück, wenn alle äußeren Ringe gegen den Uhrzeigersinn orientiert sind und alle inneren Ringe im Uhrzeigersinn ausgerichtet sind.
 - **ST_IsPolygonCW** - Gibt den Wert TRUE zurück, wenn alle äußeren Ringe im Uhrzeigersinn und alle inneren Ringe gegen den Uhrzeigersinn ausgerichtet sind.
 - **ST_IsSimple** - Gibt den Wert (TRUE) zurück, wenn die Geometrie keine irregulären Stellen, wie Selbstüberschneidungen oder Selbstberührungen, aufweist.
 - **ST_IsSolid** - Überprüft ob die Geometrie ein Solid ist. Es wird keine Plausibilitätsprüfung durchgeführt.
 - **ST_IsValidTrajectory** - Returns true if the geometry is a valid trajectory.
 - **ST_Length_Spheroid** - Calculates the 2D or 3D length/perimeter of a geometry on an ellipsoid. This is useful if the coordinates of the geometry are in longitude/latitude and a length is desired without reprojection.
 - **ST_LineFromMultiPoint** - Erzeugt einen LineString aus einer MultiPoint Geometrie.
 - **ST_LineInterpolatePoint** - Fügt einen Punkt entlang einer Linie ein. 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.
 - **ST_LineSubstring** - Return a linestring being a substring of the input one starting and ending at the given fractions of total 2d length. Second and third arguments are float8 values between 0 and 1.
 - **ST_LineToCurve** - Wandelt einen LINESTRING/POLYGON in einen CIRCULARSTRING, CURVEPOLYGON um
 - **ST_LocateBetweenElevations** - Return a derived geometry (collection) value with elements that intersect the specified range of elevations inclusively. Only 3D, 4D LINESTRINGS and MULTILINESTRINGS are supported.
 - **ST_M** - Gibt die M-Koordinate eines Punktes, oder NULL, wenn diese nicht existiert, zurück. Bei der Eingabe muss es sich um eine Punktgeometrie handeln.
 - **ST_MakeLine** - Erzeugt einen Linienzug aus einer Punkt-, Mehrfachpunkt- oder Liniengeometrie.
 - **ST_MakePoint** - Erzeugt eine 2D-, 3D- oder 4D-Punktgeometrie.
 - **ST_MakePolygon** - Erzeugt ein Polygon, das durch die gegebene Hülle gebildet wird. Die Eingabegeometrie muss aus geschlossenen Linienzügen bestehen.
 - **ST_MakeSolid** - Wandelt die Geometrie in ein Solid um. Es wird keine Überprüfung durchgeführt. Um ein gültiges Solid zu erhalten muss die eingegebene Geometrie entweder eine geschlossene polyedrische Oberfläche oder ein geschlossenes TIN sein.
 - **ST_MakeValid** - Versucht eine ungültige Geometrie, ohne den Verlust an Knoten zu bereinigen.
 - **ST_MemSize** - Returns the amount of space (in bytes) the geometry takes.
 - **ST_MemUnion** - Das gleiche wie ST_Union, nur freundlicher zum Arbeitsspeicher (verwendet weniger Arbeitsspeicher und mehr Rechnerzeit).
 - **ST_NDims** - Gibt die Koordinatendimension der Geometrie als Small Int zurück. Der Wertebereich ist 2, 3 oder 4.
 - **ST_NPoints** - Gibt die Anzahl der Punkte (Knoten) einer Geometrie zurück.
 - **ST_NRings** - Wenn es sich bei der Geometrie um ein Polygon oder um ein MultiPolygon handelt, wird die Anzahl der Ringe zurückgegeben.
 - **ST_Node** - Knotenberechnung für eine Menge von Linienzügen.
 - **ST_NumGeometries** - Wenn es sich bei der Geometrie um eine GEOMETRYCOLLECTION (oder MULTI*) handelt, wird die Anzahl der Geometrien zurückgegeben, bei Einzelgeometrien wird 1, ansonsten NULL zurückgegeben.
 - **ST_NumPatches** - Gibt die Anzahl der Maschen einer polyedrischen Oberfläche aus. Gibt NULL zurück, wenn es sich nicht um polyedrische Geometrien handelt.
 - **ST_Orientation** - Bestimmt die Ausrichtung der Fläche
-

- **ST_PatchN** - Gibt die auf 1-basierende n-te Geometrie (Masche) zurück, wenn es sich bei der Geometrie um ein POLYHEDRALSURFACE, oder ein POLYHEDRALSURFACEM handelt. Anderenfalls wird NULL zurückgegeben.
 - **ST_PointFromWKB** - Erzeugt eine Geometrie mit gegebener SRID von WKB.
 - **ST_PointN** - Gibt den n-ten Punkt des ersten LineString's oder des kreisförmigen LineStrings's in der Geometrie zurück. Negative Werte werden rückwärts vom Ende des LineString's gezählt. Gibt NULL aus, wenn es sich bei der Geometrie nicht um einen LineString handelt.
 - **ST_PointOnSurface** - Returns a POINT guaranteed to lie on the surface.
 - **ST_Points** - Gibt einen MultiPoint zurück, welcher alle Koordinaten einer Geometrie enthält.
 - **ST_Polygon** - Gibt ein Polygon zurück, das aus vorgegebenen Linienzug und SRID erzeugt wurde.
 - **ST_RemovePoint** - Entfernt einen Punkt aus einem Linienzug.
 - **ST_RemoveRepeatedPoints** - Gibt eine Version der Eingabegeometrie zurück, wobei duplizierte Punkte entfernt werden.
 - **ST_Reverse** - Gibt die Geometrie in umgekehrter Knotenreihenfolge zurück.
 - **ST_Rotate** - Dreht die Geometrie um rotRadians gegen den Uhrzeigersinn um ein Zentrum.
 - **ST_RotateX** - Dreht eine Geometrie um rotRadians um die X-Achse.
 - **ST_RotateY** - Dreht eine Geometrie um rotRadians um die Y-Achse.
 - **ST_RotateZ** - Dreht eine Geometrie um rotRadians um die Z-Achse.
 - **ST_Scale** - Skaliert eine Geometrie anhand der gegebenen Faktoren.
 - **ST_SetPoint** - Einen Punkt eines Linienzuges durch einen gegebenen Punkt ersetzen.
 - **ST_Shift_Longitude** - Schaltet geometrische Koordinaten zwischen den Bereichen -180..180 und 0..360 um.
 - **ST_SnapToGrid** - Fängt alle Punkte der Eingabegeometrie auf einem regelmäßigen Gitter.
 - **ST_StartPoint** - Gibt den den Anfangspunkt einer LINESTRING Geometrie als POINT zurück.
 - **ST_StraightSkeleton** - Berechnet aus einer Geometrie ein "Gerippe" aus Geraden.
 - **ST_SwapOrdinates** - Gibt eine Version der Ausgangsgeometrie zurück, in der die angegebenen Ordinatenwerte ausgetauscht werden.
 - **ST_SymDifference** - Gibt eine Geometrie zurück, die jene Teile von A und B repräsentiert, die sich nicht überlagern. Wird symmetrische Differenz genannt, da $ST\_SymDifference(A,B) = ST\_SymDifference(B,A)$.
 - **ST_Tessellate** - Erzeugt ein Oberflächen-Mosaik aus einem Polygon oder einer polyedrischen Oberfläche und gibt dieses als TIN oder als TIN-Kollektion zurück
 - **ST_TransScale** - Umwandlung einer Geometrie entsprechend den gegebenen Skalierungsfaktoren und Versätzen.
 - **ST_Translate** - Verschiebt eine Geometrie um die angegebenen Versätze.
 - **ST_UnaryUnion** - Wie ST_Union, arbeitet aber auf der Ebene der Geometriebestandteile.
 - **ST_Volume** - Berechnet das Volumen eines 3D-Solids. Auf Oberflächengeometrien (auch auf geschlossene) angewandt wird 0 zurückgegeben.
 - **ST_WrapX** - Versammelt eine Geometrie um einen X-Wert
 - **ST_X** - Gibt die X-Koordinate eines Punktes, oder NULL wenn diese nicht vorhanden ist, zurück. Die Eingabe muss ein Punkt sein.
 - **ST_XMax** - Gibt das größte X des Umgebungsrechtecks in 2D, 3D oder als Geometrie zurück.
 - **ST_XMin** - Gibt das kleinste X des Umgebungsrechtecks in 2D, 3D oder als Geometrie zurück.
-

- **ST_Y** - Gibt die Y-Koordinate eines Punktes, oder NULL wenn diese nicht vorhanden ist, zurück. Die Eingabe muss ein Punkt sein.
- **ST_YMax** - Gibt das größte Y des Umgebungsrechtecks in 2D, 3D oder als Geometrie zurück.
- **ST_YMin** - Gibt das kleinste Y des Umgebungsrechtecks in 2D, 3D oder als Geometrie zurück.
- **ST_Z** - Gibt die Z-Koordinate eines Punktes, oder NULL wenn diese nicht vorhanden ist, zurück. Die Eingabe muss ein Punkt sein.
- **ST_ZMax** - Gibt das kleinste Z des Umgebungsrechtecks in 2D, 3D oder als Geometrie zurück.
- **ST_ZMin** - Gibt das kleinste Z des Umgebungsrechtecks in 2D, 3D oder als Geometrie zurück.
- **ST_Zmflag** - Gibt die ZM (semantische Dimension) Flag der Geometrie als Small Int zurück. Die Werte sind: 0=2D, 1=3DM, 2=3DZ, 3=4D.
- **TG_Equals** - Gibt TRUE zurück, wenn zwei TopoGeometry Objekte aus denselben topologischen Elementarstrukturen bestehen.
- **TG_Intersects** - Gibt TRUE zurück, wenn sich kein beliebiges Paar von Elementarstrukturen zweier TopoGeometry Objekte überschneidet.
- **UpdateGeometrySRID** - Erneuert die SRID aller Features einer Geometriespalte, die Metadaten und die SRID in "geometry_columns". Wenn Constraints erzwungen wurden, werden die Constraints mit der neuen SRID versehen. Falls die Definition über den Datentyp erfolgte, wird die Datentypdefinition geändert.
- **geometry_overlaps_nd** - Gibt TRUE zurück, wenn A's n-D bounding box B's n-D bounding box schneidet.
- **overlaps_nd_geometry_gidx** - Gibt TRUE zurück, wenn die (cached) n-D bounding box einer Geometrie eine n-D float precision bounding box (GIDX) schneidet.
- **overlaps_nd_gidx_geometry** - Gibt TRUE zurück, wenn eine n-D float precision bounding box (GIDX) eine (cached) n-D bounding box einer Geometrie schneidet.
- **overlaps_nd_gidx_gidx** - Gibt TRUE zurück, wenn sich zwei n-D float precision bounding boxes (GIDX) gegenseitig überschneiden.
- **postgis_sfcgal_version** - Gibt die verwendete Version von SFCGAL aus

14.9 PostGIS Curved Geometry Support Functions

The functions given below are PostGIS functions that can use CIRCULARSTRING, CURVEPOLYGON, and other curved geometry types

- **AddGeometryColumn** - Fügt eine Geometriespalte an eine bestehende Tabelle an. Standardmäßig wird Typmodifikation anstelle von Bedingungen/Constraints verwendet. Sie erreichen das alte, auf check constraints basierte Verhalten, wenn Sie use_typmod auf false setzen.
- **Box2D** - Returns a BOX2D representing the maximum extents of the geometry.
- **Box3D** - Returns a BOX3D representing the maximum extents of the geometry.
- **DropGeometryColumn** - Entfernt eine Geometriespalte aus einer räumlichen Tabelle.
- **GeometryType** - Gibt den Geometrietyp als Zeichenkette zurück. z.B.: 'LINESTRING', 'POLYGON', 'MULTIPOINT', etc.
- **PostGIS_AddBBox** - Fügt der Geometrie ein Umgebungsrechteck hinzu.
- **PostGIS_DropBBox** - Löscht das zwischengespeicherte Umgebungsrechteck der Geometrie.
- **PostGIS_HasBBox** - Gibt TRUE zurück, wenn die BBox der Geometrie zwischengespeichert ist, andernfalls wird FALSE zurückgegeben.

- **ST_3DExtent** - an aggregate function that returns the box3D bounding box that bounds rows of geometries.
 - **ST_Accum** - Aggregatfunktion. Erzeugt ein Feld mit Geometrien.
 - **ST_Affine** - Wendet eine affine 3D-Transformation auf die Geometrie an.
 - **ST_AsBinary** - Gibt die Well-known-Binary(WKB)-Darstellung der Geometrie/Geographie ohne die SRID Metadaten zurück.
 - **ST_AsEWKB** - Gibt die Well-known-Binary(WKB)-Darstellung der Geometrie mit den SRID Metadaten zurück.
 - **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_AsText** - Gibt die Well-known-Text(WKT) Darstellung der Geometrie/Geographie ohne die SRID Metadaten zurück.
 - **ST_Collect** - Gibt einen festgelegten ST_Geometry Wert aus einer Sammlung anderer Geometrien zurück.
 - **ST_CoordDim** - Gibt die Dimension der Koordinaten von ST_Geometry zurück.
 - **ST_CurveToLine** - Wandelt einen CIRCULARSTRING/CURVEPOLYGON in ein LINestring/POLYGON um
 - **ST_Distance** - For geometry type Returns the 2D Cartesian distance between two geometries in projected units (based on spatial ref). For geography type defaults to return minimum geodesic distance between two geographies in meters.
 - **ST_Dump** - Gibt einen Satz an geometry_dump (geom,path) Zeilen zurück, aus denen die Geometrie g1 zusammengesetzt ist.
 - **ST_DumpPoints** - Gibt eine Menge von geometry_dump (geom,path) Zeilen aller Punkte zurück, aus denen die Geometrie zusammengesetzt ist.
 - **ST_EndPoint** - Gibt den Endpunkt einer LINestring oder CIRCULARLINestring Geometrie als POINT zurück.
 - **ST_EstimatedExtent** - Return the 'estimated' extent of the given spatial table. The estimated is taken from the geometry column's statistics. The current schema will be used if not specified.
 - **ST_FlipCoordinates** - Gibt eine Version der gegebenen Geometrie zurück, wobei die X und Y Achse vertauscht sind. Nützlich wenn man Geoobjekte in Breite/Länge vorliegen hat und dies beheben möchte.
 - **ST_Force2D** - Die Geometrien in einen "2-dimensionalen Modus" zwingen.
 - **ST_ForceCurve** - Wandelt einen geometrischen in einen Kurven Datentyp um, soweit anwendbar.
 - **ST_ForceSFS** - Erzwingt, dass Geometrien nur vom Typ SFS 1.1 sind.
 - **ST_Force3D** - Zwingt die Geometrien in einen XYZ Modus. Dies ist ein Alias für ST_Force3DZ.
 - **ST_Force3DM** - Zwingt die Geometrien in einen XYM Modus.
 - **ST_Force3DZ** - Zwingt die Geometrien in einen XYZ Modus.
 - **ST_Force4D** - Zwingt die Geometrien in einen XYZM Modus.
 - **ST_ForceCollection** - Wandelt eine Geometrie in eine GEOMETRYCOLLECTION um.
 - **ST_GeoHash** - Gibt die Geometrie in der GeoHash Darstellung aus.
 - **ST_GeogFromWKB** - Erzeugt ein geographisches Objekt aus der Well-known-Binary (WKB) oder der erweiterten Well-known-Binary (EWKB) Darstellung.
 - **ST_GeomFromEWKB** - Gibt einen geometrischen Datentyp (ST_Geometry) aus einer Well-known-Binary (WKB) Darstellung zurück.
 - **ST_GeomFromEWKT** - Gibt einen spezifizierten ST_Geometry-Wert von einer erweiterten Well-known-Text Darstellung (EWKT) zurück.
 - **ST_GeomFromText** - Gibt einen spezifizierten ST_Geometry Wert aus einer Well-known-Text Darstellung (WKT) zurück.
-

- **ST_GeomFromWKB** - Erzeugt ein geometrisches Objekt aus der Well-known-Binary (WKB) Darstellung und einer optionalen SRID.
 - **ST_GeometryN** - Gibt die auf 1-basierende n-te Geometrie zurück, wenn es sich bei der Geometrie um eine GEOMETRYCOLLECTION, (MULTI)POINT, (MULTI)LINestring, MULTICURVE oder (MULTI)POLYGON, POLYHEDRALSURFACE handelt. Anderenfalls wird NULL zurückgegeben.
 - **=** - Returns TRUE if the coordinates and coordinate order geometry/geography A are the same as the coordinates and coordinate order of geometry/geography B.
 - **&<|** - Gibt TRUE zurück, wenn die bounding box von A jene von B überlagert oder unterhalb liegt.
 - **ST_HasArc** - Returns true if a geometry or geometry collection contains a circular string
 - **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, wenn der Parameter eine Ansammlung/Kollektion von Geometrien ist (MULTI*, GEOMETRYCOLLECTION, ...)
 - **ST_IsEmpty** - Gibt den Wert TRUE zurück, falls es sich bei der Geometrie um eine leere GeometryCollection, Polygon, Point etc. handelt.
 - **ST_LineToCurve** - Wandelt einen LINestring/POLYGON in einen CIRCULARSTRING, CURVEPOLYGON um
 - **ST_MemSize** - Returns the amount of space (in bytes) the geometry takes.
 - **ST_NPoints** - Gibt die Anzahl der Punkte (Knoten) einer Geometrie zurück.
 - **ST_NRings** - Wenn es sich bei der Geometrie um ein Polygon oder um ein MultiPolygon handelt, wird die Anzahl der Ringe zurückgegeben.
 - **ST_PointFromWKB** - Erzeugt eine Geometrie mit gegebener SRID von WKB.
 - **ST_PointN** - Gibt den n-ten Punkt des ersten LineString's oder des kreisförmigen LineStrings's in der Geometrie zurück. Negative Werte werden rückwärts vom Ende des LineString's gezählt. Gibt NULL aus, wenn es sich bei der Geometrie nicht um einen LineString handelt.
 - **ST_Points** - Gibt einen MultiPoint zurück, welcher alle Koordinaten einer Geometrie enthält.
 - **ST_Rotate** - Dreht die Geometrie um rotRadians gegen den Uhrzeigersinn um ein Zentrum.
 - **ST_RotateZ** - Dreht eine Geometrie um rotRadians um die Z-Achse.
 - **ST_SRID** - Gibt den Identifikator des Koordinatenreferenzsystems, wie in der spatial_ref_sys Tabelle definiert, für die ST_Geometry aus.
 - **ST_Scale** - Skaliert eine Geometrie anhand der gegebenen Faktoren.
 - **ST_SetSRID** - Weist der SRID einer Geometrie einen bestimmten Ganzzahlwert zu.
 - **ST_StartPoint** - Gibt den den Anfangspunkt einer LINestring Geometrie als POINT zurück.
 - **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** - Umwandlung einer Geometrie entsprechend den gegebenen Skalierungsfaktoren und Versätzen.
 - **ST_Transform** - Gibt eine neue Geometrie zurück, wobei eine Koordinatentransformation in ein anderes räumliches Bezugssystem durchgeführt wird.
 - **ST_Translate** - Verschiebt eine Geometrie um die angegebenen Versätze.
 - **ST_XMax** - Gibt das größte X des Umgebungsrechtecks in 2D, 3D oder als Geometrie zurück.
-

- **ST_XMin** - Gibt das kleinste X des Umgebungsrechtecks in 2D, 3D oder als Geometrie zurück.
- **ST_YMax** - Gibt das größte Y des Umgebungsrechtecks in 2D, 3D oder als Geometrie zurück.
- **ST_YMin** - Gibt das kleinste Y des Umgebungsrechtecks in 2D, 3D oder als Geometrie zurück.
- **ST_ZMax** - Gibt das kleinste Z des Umgebungsrechtecks in 2D, 3D oder als Geometrie zurück.
- **ST_ZMin** - Gibt das kleinste Z des Umgebungsrechtecks in 2D, 3D oder als Geometrie zurück.
- **ST_Zmflag** - Gibt die ZM (semantische Dimension) Flag der Geometrie als Small Int zurück. Die Werte sind: 0=2D, 1=3DM, 2=3DZ, 3=4D.
- **UpdateGeometrySRID** - Erneuert die SRID aller Features einer Geometriespalte, die Metadaten und die SRID in "geometry_columns". Wenn Constraints erzwungen wurden, werden die Constraints mit der neuen SRID versehen. Falls die Definition über den Datentyp erfolgte, wird die Datentypdefinition geändert.
- **~(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.

14.10 PostGIS Polyhedral Surface Support Functions

The functions given below are PostGIS functions that can use POLYHEDRALSURFACE, POLYHEDRALSURFACEM geometries

- **Box2D** - Returns a BOX2D representing the maximum extents of the geometry.

- **Box3D** - Returns a BOX3D representing the maximum extents of the geometry.
 - **GeometryType** - Gibt den Geometrietyp als Zeichenkette zurück. z.B.: 'LINESTRING', 'POLYGON', 'MULTIPOINT', etc.
 - **ST_3DArea** - Berechnet die Fläche von 3D-Oberflächengeometrien. Gibt 0 für Solids zurück.
 - **ST_3DClosestPoint** - Returns the 3-dimensional point on g1 that is closest to g2. This is the first point of the 3D shortest line.
 - **ST_3DDFullyWithin** - Returns true if all of the 3D geometries are within the specified distance of one another.
 - **ST_3DDWithin** - For 3d (z) geometry type Returns true if two geometries 3d distance is within number of units.
 - **ST_3DDifference** - Errechnet die Differenzmenge in 3D
 - **ST_3DDistance** - For geometry type Returns the 3-dimensional cartesian minimum distance (based on spatial ref) between two geometries in projected units.
 - **ST_3DExtent** - an aggregate function that returns the box3D bounding box that bounds rows of geometries.
 - **ST_3DIntersection** - Führt eine Verschneidung in 3D aus
 - **ST_3DIntersects** - Returns TRUE if the Geometries "spatially intersect" in 3d - only for points, linestrings, polygons, polyhedral surface (area). With SFCGAL backend enabled also supports TINS
 - **ST_3DLongestLine** - Returns the 3-dimensional longest line between two geometries
 - **ST_3DMaxDistance** - For geometry type Returns the 3-dimensional cartesian maximum distance (based on spatial ref) between two geometries in projected units.
 - **ST_3DShortestLine** - Gibt die 3-dimensionale kürzeste Strecke zwischen zwei Geometrien zurück
 - **ST_3DUnion** - Führt eine Vereinigung/Union in 3D aus
 - **ST_Accum** - Aggregatfunktion. Erzeugt ein Feld mit Geometrien.
 - **ST_Affine** - Wendet eine affine 3D-Transformation auf die Geometrie an.
 - **ST_ApproximateMedialAxis** - Errechnet die genäherte Mediale Achse einer Flächengeometrie.
 - **ST_Area** - Returns the area of the surface if it is a Polygon or MultiPolygon. For geometry, a 2D Cartesian area is determined with units specified by the SRID. For geography, area is determined on a curved surface with units in square meters.
 - **ST_AsBinary** - Gibt die Well-known-Binary(WKB)-Darstellung der Geometrie/Geographie ohne die SRID Metadaten zurück.
 - **ST_AsEWKB** - Gibt die Well-known-Binary(WKB)-Darstellung der Geometrie mit den SRID Metadaten zurück.
 - **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 von ST_Geometry zurück.
 - **ST_Dimension** - Die inhärente Dimension des geometrischen Objekts muss niedriger oder gleich der Dimension der Koordinaten sein.
 - **ST_Dump** - Gibt einen Satz an geometry_dump (geom,path) Zeilen zurück, aus denen die Geometrie g1 zusammengesetzt ist.
 - **ST_DumpPoints** - Gibt eine Menge von geometry_dump (geom,path) Zeilen aller Punkte zurück, aus denen die Geometrie zusammengesetzt ist.
 - **ST_Expand** - Returns bounding box expanded in all directions from the bounding box of the input geometry. Uses double-precision
 - **ST_Extent** - an aggregate function that returns the bounding box that bounds rows of geometries.
-

- **ST_Extrude** - Weitet eine Oberfläche auf ein entsprechendes Volumen aus
 - **ST_FlipCoordinates** - Gibt eine Version der gegebenen Geometrie zurück, wobei die X und Y Achse vertauscht sind. Nützlich wenn man Geoobjekte in Breite/Länge vorliegen hat und dies beheben möchte.
 - **ST_Force2D** - Die Geometrien in einen "2-dimensionalen Modus" zwingen.
 - **ST_ForceLHR** - Erzwingt LHR Orientierung
 - **ST_ForceRHR** - Orientiert die Knoten in einem Polygon so, dass sie der Drei-Finger-Regel folgen.
 - **ST_ForceSFS** - Erzwingt, dass Geometrien nur vom Typ SFS 1.1 sind.
 - **ST_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 die auf 1-basierende n-te Geometrie zurück, wenn es sich bei der Geometrie um eine GEOMETRYCOLLECTION, (MULTI)POINT, (MULTI)LINestring, MULTICURVE oder (MULTI)POLYGON, POLYHEDRALSURFACE handelt. Anderenfalls wird NULL zurückgegeben.
 - **ST_GeometryType** - Gibt den Geometrietyp des ST_Geometry Wertes zurück.
 - **=** - Returns TRUE if the coordinates and coordinate order geometry/geography A are the same as the coordinates and coordinate order of geometry/geography B.
 - **&<|** - 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_IsPlanar** - Überprüft ob es sich um eine ebene Oberfläche handelt oder nicht
 - **ST_IsSolid** - Überprüft ob die Geometrie ein Solid ist. Es wird keine Plausibilitätsprüfung durchgeführt.
 - **ST_MakeSolid** - Wandelt die Geometrie in ein Solid um. Es wird keine Überprüfung durchgeführt. Um ein gültiges Solid zu erhalten muss die eingegebene Geometrie entweder eine geschlossene polyedrische Oberfläche oder ein geschlossenes TIN sein.
 - **ST_MemSize** - Returns the amount of space (in bytes) the geometry takes.
 - **ST_NPoints** - Gibt die Anzahl der Punkte (Knoten) einer Geometrie zurück.
 - **ST_NumGeometries** - Wenn es sich bei der Geometrie um eine GEOMETRYCOLLECTION (oder MULTI*) handelt, wird die Anzahl der Geometrien zurückgegeben, bei Einzelgeometrien wird 1, ansonsten NULL zurückgegeben.
 - **ST_NumPatches** - Gibt die Anzahl der Maschen einer polyedrischen Oberfläche aus. Gibt NULL zurück, wenn es sich nicht um polyedrische Geometrien handelt.
 - **ST_PatchN** - Gibt die auf 1-basierende n-te Geometrie (Masche) zurück, wenn es sich bei der Geometrie um ein POLYHEDRALSURFACE, oder ein POLYHEDRALSURFACEM handelt. Anderenfalls wird NULL zurückgegeben.
 - **ST_RemoveRepeatedPoints** - Gibt eine Version der Eingabegeometrie zurück, wobei duplizierte Punkte entfernt werden.
-

- **ST_Reverse** - Gibt die Geometrie in umgekehrter Knotenreihenfolge zurück.
- **ST_Rotate** - Dreht die Geometrie um rotRadians gegen den Uhrzeigersinn um ein Zentrum.
- **ST_RotateX** - Dreht eine Geometrie um rotRadians um die X-Achse.
- **ST_RotateY** - Dreht eine Geometrie um rotRadians um die Y-Achse.
- **ST_RotateZ** - Dreht eine Geometrie um rotRadians um die Z-Achse.
- **ST_Scale** - Skaliert eine Geometrie anhand der gegebenen Faktoren.
- **ST_ShiftLongitude** - Schaltet geometrische Koordinaten zwischen den Bereichen -180..180 und 0..360 um.
- **ST_StraightSkeleton** - Berechnet aus einer Geometrie ein "Gerippe" aus Geraden.
- **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_Tessellate** - Erzeugt ein Oberflächen-Mosaik aus einem Polygon oder einer polyedrischen Oberfläche und gibt dieses als TIN oder als TIN-Kollektion zurück
- **ST_Transform** - Gibt eine neue Geometrie zurück, wobei eine Koordinatentransformation in ein anderes räumliches Bezugssystem durchgeführt wird.
- **ST_Volume** - Berechnet das Volumen eines 3D-Solids. Auf Oberflächengeometrien (auch auf geschlossene) angewandt wird 0 zurückgegeben.
- **~(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.
- **postgis_sfcgal_version** - Gibt die verwendete Version von SFCGAL aus

14.11 PostGIS Function Support Matrix

Below is an alphabetical listing of spatial specific functions in PostGIS and the kinds of spatial types they work with or OGC/SQL compliance they try to conform to.

- A  means the function works with the type or subtype natively.
- A  means it works but with a transform cast built-in using cast to geometry, transform to a "best srid" spatial ref and then cast back. Results may not be as expected for large areas or areas at poles and may accumulate floating point junk.
- A  means the function works with the type because of a auto-cast to another such as to box3d rather than direct type support.
- A  means the function only available if PostGIS compiled with SFCGAL support.
- A  means the function support is provided by SFCGAL if PostGIS compiled with SFCGAL support, otherwise GEOS/built-in support.
- geom - Basic 2D geometry support (x,y).
- geog - Basic 2D geography support (x,y).
- 2.5D - basic 2D geometries in 3 D/4D space (has Z or M coord).
- PS - Polyhedral surfaces
- T - Triangles and Triangulated Irregular Network surfaces (TIN)

Function	geom	geog	2.5D	Curves	SQL MM	PS	T
Box2D	✓			✓		✓	✓
Box3D	✓		✓	✓		✓	✓
Find_SRID							
GeometryType	✓		✓	✓		✓	✓
ST_3DArea							
ST_3DClosestPoint	✓		✓			✓	
ST_3DDFullyWithi	✓		✓			✓	
ST_3DDWithin	✓		✓		✓	✓	
ST_3DDifference							
ST_3DDistance	✓						
ST_3DExtent	✓		✓	✓		✓	✓
ST_3DIntersection							
ST_3DIntersects	✓						

Function	geom	geog	2.5D	Curves	SQL MM	PS	T
ST_3DLength	✓		✓				
ST_3DLongestLine	✓		✓			✓	
ST_3DMakeBox	✓						
ST_3DMaxDistance	✓		✓			✓	
ST_3DPerimeter	✓		✓				
ST_3DShortestLine	✓		✓			✓	
ST_3DUnion							
ST_Accum	✓		✓	✓		✓	✓
ST_AddMeasure	✓		✓				
ST_AddPoint	✓		✓				
ST_Affine	✓		✓	✓		✓	✓
ST_ApproximateM  Axis							
ST_Area	✓	✓					
ST_AsBinary	✓	✓	✓	✓	✓	✓	✓
ST_AsEWKB	✓		✓	✓		✓	✓
ST_AsEWKT	✓	✓	✓	✓		✓	✓
ST_AsEncodedPoly	✓						
ST_AsGML	✓	✓	✓			✓	✓
ST_AsGeoJSON	✓	✓	✓				
ST_AsGeobuf							
ST_AsHEXEWKB	✓		✓	✓			
ST_AsKML	✓	✓	✓				
ST_AsLatLonText	✓						
ST_AsMVT							
ST_AsMVTGeom	✓						
ST_AsSVG	✓	✓					
ST_AsTWKB	✓						
ST_AsText	✓	✓		✓	✓		
ST_AsX3D	✓		✓			✓	✓
ST_Azimuth	✓	✓					
ST_BdMPolyFromT	✓						
ST_BdPolyFromTe	✓						
ST_Boundary	✓		✓		✓		

Function	geom	geog	2.5D	Curves	SQL MM	PS	T
ST_BoundingDiagon	✓		✓				
ST_Box2dFromGeo	✗h						
ST_Buffer	✓	😄			✓		
ST_BuildArea	✓						
ST_CPAWithin	✓		✓				
ST_Centroid	✓	✓			✓		
ST_ClipByBox2D	✓						
ST_ClosestPoint	✓						
ST_ClosestPointOf	✓roach		✓				
ST_ClusterDBSCAN	✓						
ST_ClusterIntersect	✓						
ST_ClusterKMeans	✓						
ST_ClusterWithin	✓						
ST_Collect	✓		✓	✓			
ST_CollectionExtra	✓						
ST_CollectionHom	✓ize						
ST_ConcaveHull	✓						
ST_Contains	✓				✓		
ST_ContainsProperl	✓						
ST_ConvexHull	✓		✓		✓		
ST_CoordDim	✓		✓	✓	✓	✓	✓
ST_CoveredBy	✓	✓					
ST_Covers	✓	✓					
ST_Crosses	✓				✓		
ST_CurveToLine	✓		✓	✓	✓		
ST_DFullyWithin	✓						
ST_DWithin	✓	✓					
ST_DelaunayTriang	✓		✓				✓
ST_Difference	✓		✓		✓		
ST_Dimension	✓				✓	✓	✓
ST_Disjoint	✓				✓		
ST_Distance	✓	✓		✓	📊		
ST_DistanceCPA	✓		✓				

Function	geom	geog	2.5D	Curves	SQL MM	PS	T
ST_DistanceSphere	✓						
ST_DistanceSpheroid	✓						
ST_Dump	✓		✓	✓		✓	✓
ST_DumpPoints	✓		✓	✓		✓	✓
ST_DumpRings	✓		✓				
ST_EndPoint	✓		✓	✓	✓		
ST_Envelope	✓				✓		
ST_Equals	✓				✓		
ST_EstimatedExtent				✓			
ST_Expand	✓					✓	✓
ST_Extent	✓					✓	✓
ST_ExteriorRing	✓		✓		✓		
ST_Extrude							
ST_FlipCoordinates	✓		✓	✓		✓	✓
ST_Force2D	✓		✓	✓		✓	
ST_ForceCurve	✓		✓	✓			
ST_ForceLHR							
ST_ForcePolygonCCW	✓		✓				
ST_ForcePolygonCW	✓		✓				
ST_ForceRHR	✓		✓			✓	
ST_ForceSFS	✓		✓	✓		✓	✓
ST_Force3D	✓		✓	✓		✓	
ST_Force3DMM	✓			✓			
ST_Force3DZ	✓		✓	✓		✓	
ST_Force4D	✓		✓	✓			
ST_ForceCollection	✓		✓	✓		✓	
ST_FrechetDistance	✓						
ST_GMLToSQL	✓				✓		
ST_GeneratePoints	✓						
ST_GeoHash	✓			✓			
ST_GeogFromText		✓					
ST_GeogFromWKB		✓		✓			
ST_GeographyFromText		✓					

Function	geom	geog	2.5D	Curves	SQL MM	PS	T
ST_GeomCollFromText	✓				✓		
ST_GeomFromEWKB	✓		✓	✓		✓	✓
ST_GeomFromEWKB	✓		✓	✓		✓	✓
ST_GeomFromGML	✓		✓			✓	✓
ST_GeomFromGeoJSON	✓						
ST_GeomFromGeoJSON	✓	N	✓				
ST_GeomFromKML	✓		✓				
ST_GeomFromTWKB	✓						
ST_GeomFromText	✓			✓	✓		
ST_GeomFromWK	✓			✓	✓		
ST_GeometricMedian	✓		✓				
ST_GeometryFromText	✓				✓		
ST_GeometryN	✓		✓	✓	✓	✓	✓
ST_GeometryType	✓		✓		✓	✓	
>>	✓						
<<	✓						
~	✓						
@	✓						
=	✓	✓		✓		✓	
<<	✓						
&>	✓						
&<	✓			✓		✓	
&<	✓						
&>	✓						
>>	✓						
~=	✓					✓	
ST_HasArc	✓		✓	✓			
ST_HausdorffDistance	✓						
ST_InteriorRingN	✓		✓		✓		
ST_InterpolatePoint	✓		✓				
ST_Intersection	✓	😄			📄		
ST_Intersects	✓	✓			📄		
ST_IsClosed	✓		✓	✓	✓	✓	

Function	geom	geog	2.5D	Curves	SQL MM	PS	T
ST_IsCollection	✓		✓	✓			
ST_IsEmpty	✓			✓	✓		
ST_IsPlanar							
ST_IsPolygonCCW	✓		✓				
ST_IsPolygonCW	✓		✓				
ST_IsRing	✓				✓		
ST_IsSimple	✓		✓		✓		
ST_IsSolid							
ST_IsValid	✓				✓		
ST_IsValidDetail	✓						
ST_IsValidReason	✓						
ST_IsValidTrajectory	✓		✓				
ST_Length	✓	✓					
ST_Length2D	✓						
ST_Length2D_Spheroid	✓						
ST_LengthSpheroid	✓		✓				
ST_LineCrossingDirection	✓						
ST_LineFromEncodedPolyline	✓						
ST_LineFromMultiPoint	✓		✓				
ST_LineFromText	✓				✓		
ST_LineFromWKB	✓				✓		
ST_LineInterpolatePoint	✓		✓				
ST_LineLocatePoint	✓						
ST_LineMerge	✓						
ST_LineSubstring	✓		✓				
ST_LineToCurve	✓		✓	✓			
ST_LinestringFromWKB	✓				✓		
ST_LocateAlong	✓						
ST_LocateBetween	✓						
ST_LocateBetweenPoints	✓		✓				
ST_LongestLine	✓						
ST_M	✓		✓		✓		
ST_MLineFromText	✓				✓		

Function	geom	geog	2.5D	Curves	SQL MM	PS	T
ST_MPointFromText	✓				✓		
ST_MPolyFromText	✓				✓		
ST_MakeBox2D	✓						
ST_MakeEnvelope	✓						
ST_MakeLine	✓		✓				
ST_MakePoint	✓		✓				
ST_MakePointM	✓						
ST_MakePolygon	✓		✓				
ST_MakeSolid							
ST_MakeValid	✓		✓				
ST_MaxDistance	✓						
ST_MemSize	✓		✓	✓		✓	✓
ST_MemUnion	✓		✓				
ST_MinimumBoundingCircle	✓						
ST_MinimumBoundingRadius	✓						
ST_MinimumClearance	✓						
ST_MinimumClearanceLine	✓						
ST_MinkowskiSum							
ST_Multi	✓						
ST_NDims	✓		✓				
ST_NPoints	✓		✓	✓		✓	
ST_NRings	✓		✓	✓			
ST_Node	✓		✓				
ST_Normalize	✓						
ST_NumGeometries	✓		✓		✓	✓	✓
ST_NumInteriorRings	✓						
ST_NumInteriorRing	✓				✓		
ST_NumPatches	✓		✓		✓	✓	
ST_NumPoints	✓				✓		
ST_OffsetCurve	✓						
ST_OrderingEquals	✓				✓		
ST_Orientation							
ST_Overlaps	✓				✓		

Function	geom	geog	2.5D	Curves	SQL MM	PS	T
ST_PatchN	✓		✓		✓	✓	
ST_Perimeter	✓	✓			✓		
ST_Perimeter2D	✓						
ST_Point	✓				✓		
ST_PointFromGeoHash							
ST_PointFromText	✓				✓		
ST_PointFromWKB	✓		✓	✓	✓		
ST_PointN	✓		✓	✓	✓		
ST_PointOnSurface	✓		✓		✓		
ST_PointInsideCircle	✓						
ST_Points	✓		✓	✓			
ST_Polygon	✓		✓		✓		
ST_PolygonFromText	✓				✓		
ST_Polygonize	✓						
ST_Project		✓					
ST_Relate	✓				✓		
ST_RelateMatch							
ST_RemovePoint	✓		✓				
ST_RemoveRepeatedPoints	✓		✓			✓	
ST_Reverse	✓		✓			✓	
ST_Rotate	✓		✓	✓		✓	✓
ST_RotateX	✓		✓			✓	✓
ST_RotateY	✓		✓			✓	✓
ST_RotateZ	✓		✓	✓		✓	✓
ST_SRID	✓			✓	✓		
ST_Scale	✓		✓	✓		✓	✓
ST_Segmentize	✓	✓					
ST_SetEffectiveArea	✓						
ST_SetPoint	✓		✓				
ST_SetSRID	✓			✓			
ST_SharedPaths	✓						
ST_ShiftLongitude	✓		✓			✓	✓
ST_ShortestLine	✓						
ST_Simplify	✓						
ST_SimplifyPreserveTopology	✓						

Function	geom	geog	2.5D	Curves	SQL MM	PS	T
ST_SimplifyVW	✓						
ST_Snap	✓						
ST_SnapToGrid	✓		✓				
ST_Split	✓						
ST_StartPoint	✓		✓	✓	✓		
ST_StraightSkeleton							
ST_Subdivide	✓						
ST_Summary	✓	✓		✓		✓	✓
ST_SwapOrdinates	✓		✓	✓		✓	✓
ST_SymDifference	✓		✓		✓		
ST_Tessellate							
ST_Touches	✓				✓		
ST_TransScale	✓		✓	✓			
ST_Transform	✓			✓	✓	✓	
ST_Translate	✓		✓	✓			
ST_UnaryUnion	✓		✓				
ST_Union	✓				✓		
ST_Volume							
ST_VoronoiLines	✓						
ST_VoronoiPolygon	✓						
ST_WKBToSQL	✓				✓		
ST_WKTToSQL	✓				✓		
ST_Within	✓				✓		
ST_WrapX	✓		✓				
ST_X	✓		✓		✓		
ST_XMax			✓	✓			
ST_XMin			✓	✓			
ST_Y	✓		✓		✓		
ST_YMax			✓	✓			
ST_YMin			✓	✓			
ST_Z	✓		✓		✓		
ST_ZMax			✓	✓			
ST_ZMin			✓	✓			

Function	geom	geog	2.5D	Curves	SQL MM	PS	T
<code>ST_Zmflag</code>	✓		✓	✓			
<code>~(box2df,box2df)</code>				✓		✓	
<code>~(box2df,geometry)</code>	✓			✓		✓	
<code>~(geometry,box2df)</code>	✓			✓		✓	
<code><#></code>	✓						
<code><<#>></code>	✓						
<code><<->></code>	✓						
<code> =</code>	✓						
<code><-></code>	✓	✓					
<code>&&</code>	✓	✓		✓		✓	
<code>&&&</code>	✓		✓	✓		✓	✓
<code>@(box2df,box2df)</code>				✓		✓	
<code>@(box2df,geometry)</code>	✓			✓		✓	
<code>@(geometry,box2df)</code>	✓			✓		✓	
<code>&&(box2df,box2df)</code>				✓		✓	
<code>&&(box2df,geometry)</code>	✓			✓		✓	
<code>&&(geometry,box2df)</code>	✓			✓		✓	
<code>&&&(geometry,gid)</code>	✓		✓	✓		✓	✓
<code>&&&(gid,geometry)</code>	✓		✓	✓		✓	✓
<code>&&&(gid,gid)</code>			✓	✓		✓	✓
<code>postgis.backend</code>							
<code>postgis.enable_outdb_rasters</code>							
<code>postgis.gdal_datapath</code>							
<code>postgis.gdal_enabled_drivers</code>							
<code>postgis_sfcgal_version</code>							

14.12 New, Enhanced or changed PostGIS Functions

14.12.1 PostGIS Functions new or enhanced in 2.4

The functions given below are PostGIS functions that were added or enhanced.

Functions new in PostGIS 2.4

- **ST_Centroid** - Availability: 2.4.0 support for geography was introduced. Gibt den geometrischen Schwerpunkt der Geometrie zurück.
- **ST_FrechetDistance** - Availability: 2.4.0 - requires GEOS >= 3.7.0 Returns the Fréchet distance between two geometries. This is a measure of similarity between curves that takes into account the location and ordering of the points along the curves. Units are in the units of the spatial reference system of the geometries.

Functions enhanced in PostGIS 2.4

All aggregates now marked as parallel safe which should allow them to be used in plans that can employ parallelism.

- **Normalize_Address** - Enhanced: 2.4.0 norm_addy object includes additional fields zip4 and address_alphanumeric. Given a textual street address, returns a composite norm_addy type that has road suffix, prefix and type standardized, street, streetname etc. broken into separate fields. This function will work with just the lookup data packaged with the tiger_geocoder (no need for tiger census data).
- **Page_Normalize_Address** - Enhanced: 2.4.0 norm_addy object includes additional fields zip4 and address_alphanumeric. Given a textual street address, returns a composite norm_addy type that has road suffix, prefix and type standardized, street, streetname etc. broken into separate fields. This function will work with just the lookup data packaged with the tiger_geocoder (no need for tiger census data). Requires address_standardizer extension.
- **ST_AsTWKB** - Enhanced: 2.4.0 memory and speed improvements. Gibt die Geometrie als TWKB, aka "Tiny Well-known Binary" zurück
- **ST_Covers** - Enhanced: 2.4.0 Support for polygon in polygon and line in polygon added for geography type Gibt 1 (TRUE) zurück, falls kein Punkt der Geometrie B außerhalb von Geometry A liegt
- **ST_Project** - Enhanced: 2.4.0 Allow negative distance and non-normalized azimuth. Returns a POINT projected from a start point using a distance in meters and bearing (azimuth) in radians.

Functions changed in PostGIS 2.4

All PostGIS aggregates now marked as parallel safe. This will force a drop and recreate of aggregates during upgrade which may fail if any user views or sql functions rely on PostGIS aggregates.

- **=** - Changed: 2.4.0, in prior versions this was bounding box equality not a geometric equality. If you need bounding box equality, use `ST_Equals` instead. Returns TRUE if the coordinates and coordinate order geometry/geography A are the same as the coordinates and coordinate order of geometry/geography B.

14.12.2 PostGIS Functions new or enhanced in 2.3

The functions given below are PostGIS functions that were added or enhanced.



Note

PostGIS 2.3.0: PostgreSQL 9.6+ support for parallel queries.



Note

PostGIS 2.3.0: PostGIS extension, all functions schema qualified to reduce issues in database restore.



Note

PostGIS 2.3.0: PostgreSQL 9.4+ support for BRIN indexes. Refer to Section [4.6.2](#).



Note

PostGIS 2.3.0: Tiger Geocoder upgraded to work with TIGER 2016 data.

The functions given below are PostGIS functions that are enhanced in PostGIS 2.3.

- **ST_Contains** - Enhanced: 2.3.0 Enhancement to PIP short-circuit extended to support MultiPoints with few points. Prior versions only supported point in polygon.
- **ST_Covers** - Enhanced: 2.3.0 Enhancement to PIP short-circuit for geometry extended to support MultiPoints with few points. Prior versions only supported point in polygon.
- **ST_Expand** - Enhanced: 2.3.0 support was added to expand a box by different amounts in different dimensions.
- **ST_Intersects** - Enhanced: 2.3.0 Enhancement to PIP short-circuit extended to support MultiPoints with few points. Prior versions only supported point in polygon.
- **ST_Within** - Enhanced: 2.3.0 Enhancement to PIP short-circuit for geometry extended to support MultiPoints with few points. Prior versions only supported point in polygon.

14.12.3 PostGIS Functions new or enhanced in 2.2

The functions given below are PostGIS functions that were added or enhanced.

**Note**

postgis_sfcgal now can be installed as an extension using `CREATE EXTENSION postgis_sfcgal;`

**Note**

PostGIS 2.2.0: Tiger Geocoder upgraded to work with TIGER 2015 data.

**Note**

address_standardizer, address_standardizer_data_us extensions for standardizing address data refer to [Chapter 12](#) for details.

**Note**

Many functions in topology rewritten as C functions for increased performance.

The functions given below are PostGIS functions that are enhanced in PostGIS 2.2.

- **ST_Area** - Enhanced: 2.2.0 - measurement on spheroid performed with GeographicLib for improved accuracy and robustness. Requires Proj $\geq$ 4.9.0 to take advantage of the new feature.
- **ST_Azimuth** - Enhanced: 2.2.0 measurement on spheroid performed with GeographicLib for improved accuracy and robustness. Requires Proj $\geq$ 4.9.0 to take advantage of the new feature.
- **ST_Distance** - Enhanced: 2.2.0 - measurement on spheroid performed with GeographicLib for improved accuracy and robustness. Requires Proj $\geq$ 4.9.0 to take advantage of the new feature.

14.12.4 PostGIS functions breaking changes in 2.2

The functions given below are PostGIS functions that have possibly breaking changes in PostGIS 2.2. If you use any of these, you may need to check your existing code.

- **Get_Geocode_Setting** - Changed: 2.2.0 : default settings are now kept in a table called `geocode_settings_default`. Use customized settingsa are in `geocode_settings` and only contain those that have been set by user.
- **ST_3DClosestPoint** - Changed: 2.2.0 - if 2 2D geometries are input, a 2D point is returned (instead of old behavior assuming 0 for missing Z). In case of 2D and 3D, Z is no longer assumed to be 0 for missing Z.
- **ST_3DDistance** - Changed: 2.2.0 - In case of 2D and 3D, Z is no longer assumed to be 0 for missing Z.
- **ST_3DLongestLine** - Changed: 2.2.0 - if 2 2D geometries are input, a 2D point is returned (instead of old behavior assuming 0 for missing Z). In case of 2D and 3D, Z is no longer assumed to be 0 for missing Z.
- **ST_3DMaxDistance** - Changed: 2.2.0 - In case of 2D and 3D, Z is no longer assumed to be 0 for missing Z.
- **ST_3DShortestLine** - Changed: 2.2.0 - if 2 2D geometries are input, a 2D point is returned (instead of old behavior assuming 0 for missing Z). In case of 2D and 3D, Z is no longer assumed to be 0 for missing Z.
- **ST_DistanceSphere** - Changed: 2.2.0 In prior versions this used to be called `ST_Distance_Sphere`
- **ST_DistanceSpheroid** - Changed: 2.2.0 In prior versions this used to be called `ST_Distance_Spheroid`
- **ST_Equals** - Changed: 2.2.0 Returns true even for invalid geometries if they are binary equal
- **ST_LengthSpheroid** - Changed: 2.2.0 In prior versions this used to be called `ST_Length_Spheroid` and used to have a `ST_3DLength_Spheroid` alias
- **ST_MemSize** - Changed: 2.2.0 name changed to `ST_MemSize` to follow naming convention. In prior versions this function was called `ST_Mem_Size`, old name deprecated though still available.
- **ST_PointInsideCircle** - Changed: 2.2.0 In prior versions this used to be called `ST_Point_Inside_Circle`

14.12.5 PostGIS Functions new or enhanced in 2.1

The functions given below are PostGIS functions that were added or enhanced.



Note

More Topology performance Improvements. Please refer to Chapter 11 for more details.



Note

Bug fixes (particularly with handling of out-of-band rasters), many new functions (often shortening code you have to write to accomplish a common task) and massive speed improvements to raster functionality. Refer to Chapter 9 for more details.



Note

PostGIS 2.1.0: Tiger Geocoder upgraded to work with TIGER 2012 census data. `geocode_settings` added for debugging and tweaking rating preferences, loader made less greedy, now only downloads tables to be loaded. PostGIS 2.1.1: Tiger Geocoder upgraded to work with TIGER 2013 data. Please refer to Section 13.1 for more details.

The functions given below are PostGIS functions that are enhanced in PostGIS 2.1.

- **ST_Aspect** - Enhanced: 2.1.0 Uses ST_MapAlgebra() and added optional interpolate_nodata function parameter
- **ST_HillShade** - Enhanced: 2.1.0 Uses ST_MapAlgebra() and added optional interpolate_nodata function parameter
- **ST_Polygon** - Enhanced: 2.1.0 Improved Speed (fully C-Based) and the returning multipolygon is ensured to be valid.
- **ST_Slope** - Enhanced: 2.1.0 Uses ST_MapAlgebra() and added optional units, scale, interpolate_nodata function parameters
- **ST_DWithin** - Enhanced: 2.1.0 improved speed for geography. See Making Geography faster for details.
- **ST_DWithin** - Enhanced: 2.1.0 support for curved geometries was introduced.
- **ST_Distance** - Enhanced: 2.1.0 improved speed for geography. See Making Geography faster for details.
- **ST_Distance** - Enhanced: 2.1.0 - support for curved geometries was introduced.

14.12.6 PostGIS functions breaking changes in 2.1

The functions given below are PostGIS functions that have possibly breaking changes in PostGIS 2.1. If you use any of these, you may need to check your existing code.

- **ST_Aspect** - Changed: 2.1.0 In prior versions, return values were in radians. Now, return values default to degrees
- **ST_HillShade** - Changed: 2.1.0 In prior versions, azimuth and altitude were expressed in radians. Now, azimuth and altitude are expressed in degrees
- **ST_Polygon** - Changed: 2.1.0 In prior versions would sometimes return a polygon, changed to always return multipolygon.
- **ST_Slope** - Changed: 2.1.0 In prior versions, return values were in radians. Now, return values default to degrees
- **ST_EstimatedExtent** - Changed: 2.1.0. Up to 2.0.x this was called ST_Estimated_Extent.
- **ST_LineLocatePoint** - Changed: 2.1.0. Up to 2.0.x this was called ST_Line_Locate_Point.
- **ST_LineSubstring** - Changed: 2.1.0. Up to 2.0.x this was called ST_Line_Substring.

14.12.7 PostGIS Functions new, behavior changed, or enhanced in 2.0

The functions given below are PostGIS functions that were added, enhanced, or have Section 14.12.8 breaking changes in 2.0 releases.

New geometry types: TIN and Polyhedral surfaces was introduced in 2.0



Note

Greatly improved support for Topology. Please refer to Chapter 11 for more details.



Note

In PostGIS 2.0, raster type and raster functionality has been integrated. There are way too many new raster functions to list here and all are new so please refer to Chapter 9 for more details of the raster functions available. Earlier pre-2.0 versions had raster_columns/raster_overviews as real tables. These were changed to views before release. Functions such as ST_AddRasterColumn were removed and replaced with AddRasterConstraints, DropRasterConstraints as a result some apps that created raster tables may need changing.



Note

Tiger Geocoder upgraded to work with TIGER 2010 census data and now included in the core PostGIS documentation. A reverse geocoder function was also added. Please refer to Section 13.1 for more details.

- **@** - Availability: 2.0.5 geometry @ raster introduced Returns TRUE if A's bounding box is contained by B's. Uses double precision bounding box.
- **Loader_Generate_Script** - Availability: 2.0.0 to support Tiger 2010 structured data and load census tract (tract), block groups (bg), and blocks (tabblocks) tables . Generates a shell script for the specified platform for the specified states that will download Tiger data, stage and load into tiger_data schema. Each state script is returned as a separate record. Latest version supports Tiger 2010 structural changes and also loads census tract, block groups, and blocks tables.

The functions given below are PostGIS functions that are enhanced in PostGIS 2.0.

- **Box2D** - Enhanced: 2.0.0 support for Polyhedral surfaces, Triangles and TIN was introduced.
- **Box3D** - Enhanced: 2.0.0 support for Polyhedral surfaces, Triangles and TIN was introduced.
- **Geocode** - Enhanced: 2.0.0 to support Tiger 2010 structured data and revised some logic to improve speed, accuracy of geocoding, and to offset point from centerline to side of street address is located on. The new parameter max_results useful for specifying number of best results or just returning the best result.
- **ST_Intersection** - Enhanced: 2.0.0 - Verschneidungsoperation im Rasterraum eingeführt. In Vorgängerversionen von 2.0.0 wurde lediglich die Verschneidung im Vektorraum unterstützt.
- **ST_3DExtent** - Enhanced: 2.0.0 support for Polyhedral surfaces, Triangles and TIN was introduced.
- **ST_Accum** - Enhanced: 2.0.0 support for Polyhedral surfaces, Triangles and TIN was introduced.
- **ST_Area** - Enhanced: 2.0.0 - support for 2D polyhedral surfaces was introduced.
- **ST_Azimuth** - Enhanced: 2.0.0 support for geography was introduced.
- **ST_Expand** - Enhanced: 2.0.0 support for Polyhedral surfaces, Triangles and TIN was introduced.
- **ST_Extent** - Enhanced: 2.0.0 support for Polyhedral surfaces, Triangles and TIN was introduced.
- **ST_Relate** - Enhanced: 2.0.0 - added support for specifying boundary node rule (requires GEOS >= 3.0).

14.12.8 PostGIS Functions changed behavior in 2.0

The functions given below are PostGIS functions that have changed behavior in PostGIS 2.0 and may require application changes.



Note

Most deprecated functions have been removed. These are functions that haven't been documented since 1.2 or some internal functions that were never documented. If you are using a function that you don't see documented, it's probably deprecated, about to be deprecated, or internal and should be avoided. If you have applications or tools that rely on deprecated functions, please refer to [?qandaentry] for more details.



Note

Bounding boxes of geometries have been changed from float4 to double precision (float8). This has an impact on answers you get using bounding box operators and casting of bounding boxes to geometries. E.g ST_SetSRID(abbox) will often return a different more accurate answer in PostGIS 2.0+ than it did in prior versions which may very well slightly change answers to view port queries.



Note

The arguments hasno data was replaced with exclude_nodata_value which has the same meaning as the older hasno data but clearer in purpose.

- **Box3D** - Changed: 2.0.0 In pre-2.0 versions, there used to be a box2d instead of box3d. Since box2d is a deprecated type, this was changed to box3d.
- **ST_3DExtent** - Changed: 2.0.0 In prior versions this used to be called ST_Extent3D
- **ST_3DLength** - Changed: 2.0.0 In prior versions this used to be called ST_Length3D
- **ST_3DPerimeter** - Changed: 2.0.0 In prior versions this used to be called ST_Perimeter3D
- **ST_Length** - Changed: 2.0.0 Breaking change -- in prior versions applying this to a MULTI/POLYGON of type geography would give you the perimeter of the POLYGON/MULTIPOLYGON. In 2.0.0 this was changed to return 0 to be in line with geometry behavior. Please use ST_Perimeter if you want the perimeter of a polygon
- **ST_LocateAlong** - Changed: 2.0.0 in prior versions this used to be called ST_Locate_Along_Measure. The old name has been deprecated and will be removed in the future but is still available.
- **ST_LocateBetween** - Changed: 2.0.0 - in prior versions this used to be called ST_Locate_Between_Measures. The old name has been deprecated and will be removed in the future but is still available for backward compatibility.

14.12.9 PostGIS Functions new, behavior changed, or enhanced in 1.5

The functions given below are PostGIS functions that were introduced or enhanced in this minor release.

- **ST_Distance** - Availability: 1.5.0 geography support was introduced in 1.5. Speed improvements for planar to better handle large or many vertex geometries For geometry type Returns the 2D Cartesian distance between two geometries in projected units (based on spatial ref). For geography type defaults to return minimum geodesic distance between two geographies in meters.
- **ST_DistanceSphere** - Availability: 1.5 - support for other geometry types besides points was introduced. Prior versions only work with points. Returns minimum distance in meters between two lon/lat geometries. Uses a spherical earth and radius derived from the spheroid defined by the SRID. Faster than ST_DistanceSpheroid , but less accurate. PostGIS versions prior to 1.5 only implemented for points.
- **ST_DistanceSpheroid** - Availability: 1.5 - support for other geometry types besides points was introduced. Prior versions only work with points. Returns the minimum distance between two lon/lat geometries given a particular spheroid. PostGIS versions prior to 1.5 only support points.

14.12.10 PostGIS Functions new, behavior changed, or enhanced in 1.4

The functions given below are PostGIS functions that were introduced or enhanced in the 1.4 release.

14.12.11 PostGIS Functions new in 1.3

The functions given below are PostGIS functions that were introduced in the 1.3 release.

Chapter 15

Meldung von Problemen

15.1 Software Bugs melden

Effektive Fehlerberichte stellen eine wesentliche Methode dar, um an der Entwicklung von PostGIS mitzuwirken. Der wirksamste Fehlerbericht ist jener, der es den PostGIS-Entwicklern ermöglicht ihn nachzuvollziehen. Idealerweise enthält er ein Skript das den Fehler auslöst und jegliche Information bezüglich der Umgebung in der er entdeckt wurde. Ausreichende Information kann über `SELECT postgis_full_version()` [für PostGIS] und `SELECT version()` [für PostgreSQL] gewonnen werden.

Falls Sie nicht die letzte Version verwenden, sollten Sie zuerst unter [release changelog](#) nachsehen, ob Ihr Bug 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.

15.2 Probleme mit der Dokumentation melden

Die Dokumentation sollte die Eigenschaften und das Verhalten der Software exakt widerspiegeln. Wenn das nicht der Fall ist, so kann entweder ein Softwarebug oder eine fehlerhafte bzw. unzulängliche Dokumentation daran Schuld sein.

Probleme in der Dokumentation können unter [PostGIS bug tracker](#) gemeldet werden.

Wenn die Überarbeitung trivial ist, können Sie diese in einem neuen Bug Tracker Issue beschreiben. Geben Sie bitte die exakte Stelle in der Dokumentation an.

Wenn es sich um umfangreichere Änderungen handelt, ist ein Subversion Patch zweifellos die bessere Wahl. Dabei handelt es sich um einen vierstufigen Vorgang unter Unix (angenommen, Sie haben [Subversion](#) bereits installiert):

1. Eine Kopie des PostGIS Subversion Trunks auschecken. Eingabe unter Unix:
`svn checkout http://svn.osgeo.org/postgis/trunk/`
Dies wird im Verzeichnis `./trunk` gespeichert
2. Erledigen Sie die Änderungen an der Dokumentation mit Ihrem Lieblingstexteditor. Auf Unix, tippen Sie (zum Beispiel:-):
`vim trunk/doc/postgis.xml`
Bedenken Sie bitte, dass die Dokumentation in DocBook XML und nicht in HTML geschrieben ist. Falls Sie damit nicht vertraut sind, so folgen Sie bitte dem Beispiel in der restlichen Dokumentation.

3. Erzeugung einer Patchdatei, welche die Unterschiede zur Master-Kopie der Dokumentation enthält. Unter Unix tippen Sie bitte:

svn diff trunk/doc/postgis.xml > doc.patch

4. Fügen Sie den Patch einem neuen Thema/Issue im Bug Tracker bei.

Appendix A

Anhang

A.1 Release 2.4.0

Release date: 2017/09/30

A.1.1 Neue Funktionalität

#3822, `postgis_full_version()` geändert, so dass jetzt auch die Version von PostgreSQL angezeigt und überprüft wird, an der die Skripte ausgeführt wurden (Sandro Santilli)

#2411, Unterstützung für Kurven in `ST_Reverse` (Sandro Santilli)

#2951, `ST_Centroid` für den geographischen Datentyp (Danny Götte)

#3788, Allow `postgis_restore.pl` to work on directory-style (-Fd) dumps (Roger Crew)

#3772, Direction agnostic `ST_CurveToLine` output (Sandro Santilli / KKGeo)

#2464, `ST_CurveToLine` mit Toleranz "MaxError" (Sandro Santilli / KKGeo)

#3599, Geobuf Ausgabe via `ST_AsGeobuf` (Björn Harrtell)

#3661, Ausgabe von Mapbox Vektorkacheln via `ST_AsMVT` (Björn Harrtell / CartoDB)

#3689, Add orientation checking and forcing functions (Dan Baston)

#3753, Gist penalty speed improvements for 2D and ND points (Darafei Praliaskouski, Andrey Borodin)

#3677, `ST_FrechetDistance` (Shinichi Sugiyama)

Most aggregates (raster and geometry), and all stable / immutable (raster and geometry) marked as parallel safe

#2249, `ST_MakeEmptyCoverage` für Raster (David Zwarg, ainomieli)

#3709, Allow signed distance for `ST_Project` (Darafei Praliaskouski)

#524, Covers support for polygon on polygon, line on line, point on line for geography (Danny Götte)

A.1.2 Verbesserungen und Fehlerbehebung

Many corrections to docs and several translations almost complete. Andreas Schild who provided many corrections to core docs. PostGIS Japanese translation team first to reach completion of translation.

Unterstützung von PostgreSQL 10

Vorbereitende Unterstützung von PostgreSQL 11

#3645, Avoid loading logically deleted records from shapefiles

- #3747, der Datentyp "norm_addy tiger_geocoder" wurde um die Attribute "zip4" und "address_alphanumeric" erweitert.
- #3748, address_standardizer Lookup-Tabellen aktualisiert, damit "pagc_normalize_address" die Abkürzungen besser normiert
- #3647, verbesserte Knotenberechnung in ST_Node durch die Verwendung von GEOSNode (Wouter Geraedts)
- #3684, Aktualisierung auf das EPSG Register v9 (Even Rouault)
- #3830, Initialisierung des inkompatiblen Datentyps (>=9.6) "address_standardizer" fixiert
- #3662, shp2pgsql ergänzt, so dass im Modus "debug" Fehlermeldungen an stderr ausgegeben werden
- #3405, Speicherleck in "lwgeom_to_points" fixiert
- #3832, "shp2pgsql" unterstützt jetzt breite Ganzzahlfelder mit dem Datentyp "int8"
- #3841, Deterministic sorting support for empty geometries in btree geography
- #3844, Make = operator a strict equality test, and < > to rough "spatial sorting"
- #3855, ST_AsTWKB memory and speed improvements

A.1.3 Wichtige Änderungen

Unterstützung für PostgreSQL 9.2. eingestellt.

- #3810, GEOS 3.4.0 oder höher zum kompilieren erforderlich

Most aggregates now marked as parallel safe, which means most aggs have to be dropped / recreated. If you have views that utilize PostGIS aggs, you'll need to drop before upgrade and recreate after upgrade

- #3578, ST_NumInteriorRings(POLYGON EMPTY) gibt nun 0 anstatt NULL zurück

_ST_DumpPoints entfernt, da es ab PostGIS 2.1.0 - wo ST_DumpPoints in C neu implementiert wurde - nicht länger benötigt wird

B-Tree index operators < = > changed to provide better spatial locality on sorting and have expected behavior on GROUP BY. If you have btree index for geometry or geography, you need to REINDEX it, or review if it was created by accident and needs to be replaced with GiST index. If your code relies on old left-to-right box compare ordering, update it to use << >> operators.

A.2 Release 2.3.3

Freigabedatum: 2017/07/01

Es handelt sich um eine Release für Bugfixes und zur Verbesserung der Rechenleistung.

A.2.1 Fehlerbehebung und Verbesserungen

- #3777, Unregelmäßigkeiten mit einer leeren Geometrie bei GROUP BY
- #3711, Azimuth error upon adding 2.5D edges to topology
- #3726, PDF manual from dblatex renders fancy quotes for programlisting (Mike Toews)
- #3738, Raster: die Verwendung von -s ohne -Y in "raster2pgsql" transformiert den Raster anstatt die SRID zu setzen
- #3744, ST_Subdivide verliert Komponenten der invertierten Geometrie (Darafei Praliaskouski Komzpa)
- #3750, @ and ~ operator not always schema qualified in geometry and raster functions. Causes restore issues. (Shane StClair of Axiom Data Science)
- #3682, Eigenartige Feldlänge der booleschen Variablen im Ergebnis von pgsqll2shp
- #3701, Behebung der Probleme mit doppelten Anführungszeichen in pgsqll2shp
- #3704, ST_AsX3D stürzt mit einer leeren Geometrie ab
- #3730, Änderung von "Error" auf "Notice", wenn ST_Clip ein Band nicht berechnen kann

A.3 Release 2.3.2

Freigabedatum: 2017/01/31

Es handelt sich um eine Release für Bugfixes und zur Verbesserung der Rechenleistung.

A.3.1 Fehlerbehebung und Verbesserungen

#3418, Erneute Überprüfung von KNN in 9.5+ scheitert mit einem falsch sortierten Index für die zurückgegebenen Tuple

#3675, Funktionen für Lagevergleiche nutzen in einigen Fällen den Index nicht

#3680, Bei den PostGIS Upgrade-Skripts fehlt GRANT für die Views

#3683, PostGIS kann nach einem PostgreSQL "pg_upgrade" von < 9.5 to pg > 9.4 nicht mehr aktualisiert werden

#3688, ST_AsLatLonText: rundet Minuten

A.4 Release 2.3.1

Freigabedatum: 2006/11/28

Es handelt sich um eine Release für Bugfixes und zur Verbesserung der Rechenleistung.

A.4.1 Fehlerbehebung und Verbesserungen

#1973, st_concavehull() returns sometimes empty geometry collection Fix from gde

#3501, add raster constraint max extent exceeds array size limit for large tables

#3643, PostGIS kompiliert auf neuesten OSX XCode nicht

#3644, Deadlock bei "interrupt_relate"

#3650, ST_Extent, ST_3DExtent and ST_Mem* Aggregatfunktionen als "parallel safe" gekennzeichnet und dadurch parallele Verarbeitung ermöglicht

#3652, Collection(MultiCurve()) stürzt ab

#3656, Upgrade von Aggregaten in 2.2 oder niedrigeren Versionen fixiert

#3659, Crash caused by raster GUC define after CREATE EXTENSION using wrong memory context. (manaeem)

#3665, beschädigter Index and Speicherleck bei BRIN Indizes; Patch von Julien Rouhaud (Dalibo)

#3667, Programmfehler in ST_Segmentize mit geographischem Datentyp; Patch von Hugo Mercier (Oslandia)

A.5 Release 2.3.0

Freigabedatum: 2016/09/26

Dies ist eine neue Feature-Release, mit neuen Funktionen, verbesserter Rechenleistung, allen behobenen Bugs von PostGIS 2.2.3 und anderen nützlichen Dingen.

A.5.1 Wichtige Änderungen

#3466, Typumwandlung von Box3D in den geometrischen Datentyp gibt nun eine 3D-Geometrie zurück (Julien Rouhaud of Dalibo)

#3396, ST_EstimatedExtent, gab eine WARNUNG anstatt einer FEHLERMELDUNG aus (Regina Obe)

A.5.2 Neue Funktionalität

Add support for custom TOC in `postgis_restore.pl` (Christoph Moench-Tegeder)

Unterstützung von negativen Indizes in `ST_PointN` und `ST_SetPoint` (Rémi Cura)

Parameter zu `ST_Buffer` mit dem geographischen Datentyp hinzugefügt (Thomas Bonfort)

`TopoGeom_addElement`, `TopoGeom_remElement` (Sandro Santilli)

`populate_topology_layer` (Sandro Santilli)

#454, `ST_WrapX` und `lwgeom_wrapx` (Sandro Santilli)

#1758, `ST_Normalize` (Sandro Santilli)

#2236, `shp2pgsql -d` sendet nun "DROP TABLE IF EXISTS"

#2259, `ST_VoronoiPolygons` und `ST_VoronoiLines` (Dan Baston)

#2841 und #2996, `ST_MinimumBoundingRadius` und die neue Implementierung von `ST_MinimumBoundingCircle` verwendet den Algorithmus von Welzl (Dan Baston)

#2991, Enable `ST_Transform` to use PROJ.4 text (Mike Toews)

#3059, Allow passing per-dimension parameters in `ST_Expand` (Dan Baston)

#3339, `ST_GeneratePoints` (Paul Ramsey)

#3362, `ST_ClusterDBSCAN` (Dan Baston)

#3364, `ST_GeometricMedian` (Dan Baston)

#3391, `ST_EstimatedExtent` unterstützt Tabellenvererbung (Alessandro Pasotti)

#3424, `ST_MinimumClearance` (Dan Baston)

#3428, `ST_Points` (Dan Baston)

#3465, `ST_ClusterKMeans` (Paul Ramsey)

#3469, `ST_MakeLine` mit MULTIPOINTS (Paul Norman)

#3549, Unterstützung für parallele Abfragen in PostgreSQL 9.6, so weit wie möglich (Paul Ramsey, Regina Obe)

#3557, Geometry function costs based on query stats (Paul Norman)

#3591, Unterstützung für BRIN Indizes hinzugefügt. PostgreSQL 9.4+ notwendig. (Giuseppe Broccolo von 2nd Quadrant, Julien Rouhaud und Ronan Dunklau von Dalibo)

#3496, Make `postgis` non-relocateable for extension install, schema qualify calls in functions (Regina Obe) Should resolve once and for all for extensions #3494, #3486, #3076

#3547, Aktualisierung des Tiger-Geokodierer um TIGER 2016 und sowohl http als auch ftp zu unterstützen.

#3613, Segmentize geography using equal length segments (Hugo Mercier of Oslandia)

A.5.3 Bugfixes

Alle relevanten Bugfixes von PostGIS 2.2.3

#2841, `ST_MinimumBoundingCircle` deckt das Original nicht ab

#3604, `pgcommon/Makefile.in` orders CFLAGS incorrectly leading to wrong `liblwgeom.h` (Greg Troxel)

A.5.4 Verbesserung der Rechenleistung

#75, Verbesserung von PIP Kurzschlussauswertung (Dan Baston)

#3383, Avoid deserializing small geometries during index operations (Dan Baston)

#3400, Minor optimization of PIP routines (Dan Baston)

Make adding a line to topology interruptible (Sandro Santilli)

Aktualisierung der Dokumentation durch Mike Toews

A.6 Release 2.2.2

Freigabedatum: 2016/03/22

Es handelt sich um eine Release für Bugfixes und zur Verbesserung der Rechenleistung.

A.6.1 Neue Funktionalität

#3463, Fix crash on face-collapsing edge change

#3422, Improve ST_Split robustness on standard precision double systems (arm64, ppc64el, s390c, powerpc, ...)

#3427, Update spatial_ref_sys to EPSG version 8.8

#3433, ST_ClusterIntersecting fehlerhaft bei MultiPoints

#3435, ST_AsX3D fix rendering of concave geometries

#3436, memory handling mistake in parray_clone_deep

#3437, ST_Intersects fehlerhaft bei MultiPoints

#3461, ST_GeomFromKML crashes Postgres when there are innerBoundaryIs and no outerBoundaryIs

#3429, upgrading to 2.3 or from 2.1 can cause loop/hang on some platforms

#3460, ST_ClusterWithin gibt nach Upgrade den Fehler 'Tolerance not defined' aus

#3490, Raster data restore issues, materialized views. Scripts postgis_proc_set_search_path.sql, rpostgis_proc_set_search_path.sql refer to http://postgis.net/docs/manual-2.2/RT_FAQ.html#faq_raster_data_not_restore

#3426, failing POINT EMPTY tests on fun architectures

A.7 Release 2.2.1

Freigabedatum: 2016/01/06

Es handelt sich um eine Release für Bugfixes und zur Verbesserung der Rechenleistung.

A.7.1 Neue Funktionalität

#2232, vermeiden der Anhäufung von Fehlern beim Runden in in SVG

#3321, Fixieren von Einbußen an Rechenleistung beim Laden von Topologie

#3329, Fixieren von Einbußen bei der Stabilität von "TopoGeo_addPoint".

#3349, Fixierung des Installationspfads der postgis_topology-Skripts

#3351, Isolation der Endknoten in ST_RemoveIsoEdge (und lwt_RemIsoEdge)

#3355, "Geography": geometrische BBox für ST_Segmentize.

#3359, Verlust von Geometrien in der TopoGeometry Definition durch toTopoGeom bereinigt

#3360, _raster_constraint_info_scale ungültiger Eingabesyntax.

#3375, Absturz bei wiederholtem Entfernen von Punkten aus collection(point).

#3378, Fixierung der Handhabung von hierarchischen "TopoGeometries" mit mehreren Topologien.

#3380, #3402, Anzahl der Linien beim Laden in die Topologie verringert

#3388, #3410, fehlende Endpunkte in ST_Removepoints bereinigt

#3389, Pufferüberlauf in lwgeom_to_geojson

#3390, Kompilation unter Alpine Linux 3.2 meldet einen Fehler, wenn PostGIS und die postgis_topology Extension kompiliert wird.

#3393, Bei einigen Polygonen ergibt ST_Area NaN

#3401, Verbesserung der Stabilität von "ST_Split" auf 32bit Systemen

#3404, ST_ClusterWithin bringt das Back-end zum Absturz

#3407, Absturz beim Teilen einer Masche oder einer Kante, die an mehreren TopoGeometry Objekten beteiligt sind, beseitigt

#3411, Cluster-Funktionen verwenden den räumlichen Index nicht

#3412, Verbesserung der Stabilität beim "Snapping"-Schritt in TopoGeo_addLinestring

#3415, OSX 10.9 Kompilation unter "pkgsrc" bereinigt

Speicherleck in lwt_ChangeEdgeGeom [liblwgeom] behoben

A.8 Release 2.2.0

Freigabedatum: 2015/10/07

Dies ist eine neue Feature-Release, mit neuen Funktionen, verbesserter Rechenleistung und anderen nützlichen Dingen.

A.8.1 Neue Funktionalität

Topologie API in "liblwgeom" (Sandro Santilli / Regione Toscana - SITA)

Neue lwgeom_unaryunion Methode in liblwgeom

Neue lwgeom_linemerge Methode in liblwgeom

Neue lwgeom_is_simple Methode in liblwgeom

#3169, SFCGAL 1.1 Unterstützung: hinzufügen von ST_3DDifference, ST_3DUnion, ST_Volume, ST_MakeSolid, ST_IsSolid (Vincent Mora / Oslandia)

#3169, ST_ApproximateMedialAxis (Sandro Santilli)

ST_CPAWithin (Sandro Santilli / Boundless)

Ab PostgreSQL 9.5, der |= Operator mit CPA Semantik und KNN Unterstützung (Sandro Santilli / Boundless)

#3131, KNN Unterstützung für den geographischen Datentyp (Paul Ramsey / CartoDB)

#3023, ST_ClusterIntersecting / ST_ClusterWithin (Dan Baston)

#2703, Exakte KNN Ergebnisse für alle geometrischen Datentypen, aka "KNN re-check" (Paul Ramsey / CartoDB)

#1137, Einen Toleranzwert in ST_RemoveRepeatedPoints ermöglichen (Paul Ramsey / CartoDB)

#3062, Die Übergabe eines M Faktors für ST_Scale ermöglichen (Sandro Santilli / Boundless)

#3139, ST_BoundingDiagonal (Sandro Santilli / Boundless)

#3129, ST_IsValidTrajectory (Sandro Santilli / Boundless)

#3128, ST_ClosestPointOfApproach (Sandro Santilli / Boundless)

#3152, ST_DistanceCPA (Sandro Santilli / Boundless)

Canonical output for index key types

ST_SwapOrdinates (Sandro Santilli / Boundless)

#2918, GeographicLib - geodätische Funktionen (Mike Toews)

#3074, ST_Subdivide zum zerlegen großer Geometrien (Paul Ramsey / CartoDB)

#3040, geometrischer Schwerpunkt KNN-GiST-Index basiert (<<->>) n-D Distanzoperatoren (Sandro Santilli / Boundless)

API zur Unterbrechungsbehandlung für liblwgeom (Sandro Santilli / CartoDB)

#2939, ST_ClipByBox2D (Sandro Santilli / CartoDB)

#2247, ST_Retile und ST_CreateOverview: bei Erzeugung von Rasterübersichten in der Datenbank (Sandro Santilli / Vizzuality)

#899, Zuordnung der Attributnamen über die "-m" Option für shp2pgsql (Regina Obe / Sandro Santilli)

#1678, GUC Variable "postgis.gdal_datapath" für die GDAL Konfigurationsvariable "GDAL_DATA" eingeführt

#2843, Koordinatentransformation beim Raster Import (Sandro Santilli / Vizzuality)

#2349, Ein- und Ausgabe für encoded_polyline (Kashif Rasul)

#2159, postgis_full_version() meldet libjson Version

#2770, ST_MemSize(raster)

postgis_noop(raster) hinzugefügt

Fehlende Varianten von ST_TPI(), ST_TRI() und ST_Roughness() ergänzt

GUC Variable "postgis.gdal_enabled_drivers" für die GDAL Konfigurationsvariable "GDAL_SKIP" eingeführt

GUC Variable "postgis.enable_outdb_rasters" für den Zugriff auf Raster mit out-db Bändern eingeführt

#2387, die Extension "address_standardizer" ist jetzt Teil von PostGIS (Stephen Woodbridge / imaptools.com, Walter Sinclair, Regina Obe)

#2816, die "address_standardizer_data_us" Extension liefert die Verweise für "lex", "gaz"; "rules" für den "address_standardizer" (Stephen Woodbridge / imaptools.com, Walter Sinclair, Regina Obe)

#2341, Neuer Parameter für Masken in ST_MapAlgebra

#2397, der Shapefile-Lader liest die Zeichencodierung des Shapefiles automatisch aus

#2430, ST_ForceCurve

#2565, ST_SummaryStatsAgg()

#2567, ST_CountAgg()

#2632, ST_AsGML() Unterstützung für gekrümmte Features

#2652, --upgrade-path Option zu run_test.pl hinzugefügt

#2754, sfcgal als Extension

#2227, Generalisierung mit dem Visvalingam-Whyatt Algorithmus ST_SimplifyVW, ST_SetEffectiveArea (Nicklas Avén)

Funktionen zum codieren und decodieren von TWKB, ST_AsTWKB, ST_GeomFromTWKB (Paul Ramsey / Nicklas Avén / CartoDB)

A.8.2 Verbesserungen

#3223, memcmp Kurzschlussauswertung zu ST_Equals hinzugefügt (Daniel Baston)

#3227, Upgrade für den Tiger Geokodierer zur Unterstützung von Tiger 2015 census

#2278, liblwgeom zwischen "Minor Releases" kompatibel machen

#897, ST_AsX3D Unterstützung für GeoKoordinaten und die Systeme "GD" und "WE"; kann X- und Y-Achsen vertauschen (Option = 2, 3)

ST_Split: das Auftrennen von Linien durch MultiLines, MultiPoints und (Multi)Polygongrenzen ermöglichen

#3070, Vereinfachung des Constraint für den geometrischen Datentyp

#2839, Implement selectivity estimator for functional indexes, speeding up spatial queries on raster tables. (Sandro Santilli / Vizzuality)

- #2361, die Spalte "spatial_index" zum View "raster_columns" hinzugefügt
- #2390, Testsuite für pgsql2shp
- #2527, die Flag "-k" für raster2pgsql hinzugefügt, um die Überprüfung des Bandes auf NODATA zu überspringen
- #2616, Verringerung der Typumwandlungen in Text, während des Aufbaus und des Exports von Topologie
- #2717, ST_numPoints, ST_PointN, ST_startPoint und ST_endPoint für CompundCurve
- #2747, Unterstützung von GDAL 2.0
- #2754, SFCGAL kann nun mit CREATE EXTENSION (Vincent Mora @ Oslandia) installiert werden
- #2828, ST_Envelope(raster) von SQL nach C portieren
- #2829, Shortcut ST_Clip(raster) if geometry fully contains the raster and no NODATA specified
- #2906, Upgrade des Tiger Geokodierers für die Tiger 2014 Daten
- #3048, Generalisierung von Geometrien beschleunigt (J.Santana @ CartoDB)
- #3092, Langsamer View "geometry_columns" bei vielen Geometrietabellen

A.9 Release 2.1.8

Freigabedatum: 2015-07-07

Dies ist eine wichtige Bugfix-Release.

A.9.1 Bugfixes

- #3159, Zwischenspeicherung einer BBox durch ST_Affine nicht erzwingen - falls vorher nicht vorhanden
 - #3018, "GROUP BY Geography" gibt manchmal duplizierte Datensätze zurück
 - #3084, shp2pgsql - illegal number format when specific system locale set
 - #3094, Fehlerhafte Eingabe von GeoJSON bringt Back-end zum Absturz
 - #3104, st_asgml schleust ein zufälliges Zeichen im ID-Attribut ein
 - #3155, Mit "make uninstall" liblwgeom.h entfernen
 - #3177, gserialized_is_empty kann verschachtelte, leere Geometrien nicht behandeln
- Absturz von ST_LineLocatePoint bereinigt

A.10 Release 2.1.7

Freigabedatum: 2015-03-30

Dies ist eine wichtige Bugfix-Release.

A.10.1 Bugfixes

- #3086, ST_DumpValues() crashes backend on cleanup with invalid band indexes
- #3088, "strcasestr" in "liblwgeom.h" nicht (erneut) definieren
- #3094, Fehlerhafte Eingabe von GeoJSON bringt Back-end zum Absturz

A.11 Release 2.1.6

Freigabedatum: 2015-03-20

Es handelt sich um eine Release für Bugfixes und zur Verbesserung der Rechenleistung.

A.11.1 Verbesserungen

#3000, Ensure edge splitting and healing algorithms use indexes

#3048, Geschwindigkeitsverbesserung bei der Generalisierung von Geometrien (J.Santana @ CartoDB)

#3050, Speed up geometry type reading (J.Santana @ CartoDB)

A.11.2 Bugfixes

#2941, der geographische Datentyp darf eine andere SRID als 4326 aufweisen

#3069, small objects getting inappropriately fluffed up w/ boxes

#3068, "postgis_typmod_dims" gibt bei Dimensionen, die nicht durch ein Constraint festgelegt sind, NULL zurück

#3061, Duplizierte Punkte in JSON, GML, GML ST_GeomFrom* Funktionen erlauben

#3058, ND-GiST picksplit Methode fixiert, indem Teilungsoperation auf der bestgeeigneten Ebene durchgeführt wird

#3052, Die Operatoren <-> and <#> für PostgreSQL < 9.1 zur Verfügung stellen

#3045, Verwirrung mit der Dimensionalität im &&& Operator behoben

#3016, Deregistrierung von Layer mit beschädigten Topologien zulassen

#3015, Fehler bei der Ausführung von "TopologySummary" vermeiden

#3020, ST_AddBand out-db Bug; als Wert für die Höhe wird die Breite benutzt

#3031, Das Wiederherstellen von Geometry(Point) Tabellen die mit Leerfeldern "gedumped" wurden erlauben

A.12 Release 2.1.5

Freigabedatum: 2014-12-18

Es handelt sich um eine Release für Bugfixes und zur Verbesserung der Rechenleistung.

A.12.1 Verbesserungen

#2933, Geschwindigkeitssteigerung beim Erstellen von großen Sammelgeometrien

A.12.2 Bugfixes

#2947, Speicherleck in "lwgeom_make_valid" bereinigt

#2949, Speicherleck in lwgeom_mindistance2d beseitigt

#2931, die Bezeichnung für BOX ist case-sensitive

#2942, PostgreSQL 9.5 support

#2953, 2D Statistik wird nicht erstellt, wenn die Werte Z/M extrem sind

#3009, die Typumwandlung in Geography kann den Basiswert eines Tupels ändern

A.13 Release 2.1.4

Freigabedatum: 2014-09-10

Es handelt sich um eine Release für Bugfixes und zur Verbesserung der Rechenleistung.

A.13.1 Verbesserungen

#2745, Geschwindigkeitsverbesserung beim Aufruf von ST_Simplify mit Punkten

#2747, Unterstützung von GDAL 2.0

#2749, rtpostgis_upgrade_20_21.sql ACID gemacht

#2811, Namen der Indizes beim Laden von Shapefiles/Rastern nicht festlegen

#2829, Shortcut ST_Clip(raster) if geometry fully contains the raster and no NODATA specified

#2895, Verbesserung des Auswertungsplan durch Senkung der Kosten von ST_ConvexHull(raster) auf 300

A.13.2 Bugfixes

#2605, Armel: _ST_Covers() gibt für einem Punkt in einer Aussparung TRUE zurück

#2911, Ausgabemasstab von Rastern in ST_Rescale/ST_Resample/ST_Resize bei Massstab 1/-1 und einem Versatz von 0/0 bereinigt.

Absturz von ST_Union(raster) bereinigt

#2704, ST_GeomFromGML() funktioniert nicht einwandfrei mit einem Feld aus "gml:pos" (Even Roualt)

#2708, updategeometriesrid doesn't update srid check when schema not specified. Patch from Marc Jansen

#2720, lwpoly_add_ring should update maxrings after realloc

#2759, Fix postgis_restore.pl handling of multiline object comments embedding sql comments

#2774, fix undefined behavior in parray_calculate_gbox_geodetic

Speicherzugriffsverletzung in ST_MakeValid behoben

#2784, Falscher Übergabewert --with-sfcgal bereinigt

#2772, Premature memory free in RASTER_getBandPath (ST_BandPath)

#2755, Regressionstests für alle Versionen von SFCGAL fixiert

#2775, Speicherleck in "lwline_from_lwmpoint"

#2802, ST_MapAlgebra überprüft die von der Rückruffunktion zurückgegebenen Werte auf Validität

#2803, ST_MapAlgebra handles no userarg and STRICT callback function

#2834, ST_Estimated_Extent and mixedCase table names (regression bug)

#2845, ST_AddPoint erzeugt mangelhafte Geometrie

#2870, Binary insert into geography column results geometry being inserted

#2872, make install builds documentation (Greg Troxell)

#2819, find isfinite or replacement on Centos5 / Solaris

#2899, geocode limit 1 not returning best answer (tiger geocoder)

#2903, Kann auf FreeBSD nicht kompiliert werden

#2927 reverse_geocode not filling in direction prefix (tiger geocoder) get rid of deprecated ST_Line_Locate_Point called

A.14 Release 2.1.3

Freigabedatum: 2014/05/13

Dies ist eine Bugfix- und Sicherheits-Release.

A.14.1 Wichtige Änderungen

Beginnend mit dieser Version sind der Zugriff auf Offline-Raster und die GDAL-Treiber standardmäßig deaktiviert.

Einführung der Umgebungsvariable "POSTGIS_GDAL_ENABLED_DRIVERS" zum Aktivieren bestimmter GDAL-Treiber. Standardmäßig sind alle GDAL-Treiber deaktiviert

Einführung der Umgebungsvariable "POSTGIS_GDAL_ENABLED_RASTERS" um out-db Rasterbänder zu ermöglichen. Standardmäßig sind out-db Raster deaktiviert

The environment variables must be set for the PostgreSQL process, and determines the behavior of the whole cluster.

A.14.2 Bugfixes

#2697, Eingabe eines ungültigen GeoJSON Polygons führt zu Absturz des Server-Prozesses

#2700, Fix dumping of higher-dimension datasets with null rows

#2706, ST_DumpPoints von LEEREN Geometrien führt zu Serverabsturz

A.15 Release 2.1.2

Freigabedatum: 2014/03/31

Dies ist eine Bugfix-Release, die sich mit Problemen befasst die seit Release 2.1.1 gemeldet wurden.

A.15.1 Bugfixes

#2666, Error out at configure time if no SQL preprocessor can be found

#2534, ST_Distance gibt falsche Ergebnisse für große Geographien zurück

#2539, Die Anwesenheit/Brauchbarkeit von json-c/json.h vor der Kompilation überprüfen

#2543, invalid join selectivity error from simple query

#2546, GeoJSON with string coordinates parses incorrectly

#2547, ST_Simplify(TopoGeometry) für hierarchische TopoGeometrien bereinigt

#2552, Handhabung von NULL-Rastern in ST_AsPNG, ST_AsTIFF und ST_AsJPEG fixiert

#2555, Fix parsing issue of range arguments of ST_Reclass

#2556, Das Ergebnis von ST_Intersects hängt beim geographischen Datentyp von der Reihenfolge der Eingabe ab

#2580, Doppelinstallationen von PostGIS in der selben Datenbank verbieten

#2589, Remove use of unnecessary void pointers

#2607, Innerhalb einer Datenbankverbindung können maximal 1024 out-db Dateien geöffnet werden

#2610, Ensure face splitting algorithm uses the edge index

#2615, EstimatedExtent (and hence, underlying stats) gathering wrong bbox

#2619, Empty rings array in GeoJSON polygon causes crash

- #2634, regression in sphere distance code
- #2638, die geographische Entfernungsberechnung ist bei M-Geometrien manchmal falsch
- #2648, #2653, Lösung für topologische Funktionen, wenn das Schema "topology" nicht im Suchpfad/"search_path" ist
- #2654, Überholte Aufrufe von "topology" eingestellt
- #2655, Zulassen, dass Anwender die keine Berechtigung auf das Schema "topology" haben, postgis_full_version() aufrufen können
- #2674, Fix missing operator = and hash_raster_ops opclass on raster
- #2675, #2534, #2636, #2634, #2638, Probleme bei der geographischen Entfernungsberechnung

A.15.2 Verbesserungen

- #2494, Arbeitsspeicherkopien im GiST Index vermeiden (hayamiz)
- #2560, Soft-Upgrade: Das Löschen/Neuerstellen von unveränderten Aggregaten vermeiden

A.16 Release 2.1.1

Freigabedatum: 2013/11/06

Dies ist eine Bugfix-Release, die sich mit Problemen befasst die seit Release 2.1.0 gemeldet wurden.

A.16.1 Wichtige Änderungen

- #2514, Änderung der Lizenz für Raster von GPL v3+ auf v2+; ermöglicht die Verbreitung der PostGIS-Extension als GPLv2.

A.16.2 Bugfixes

- #2396, Make regression tests more endian-agnostic
- #2434, Fix ST_Intersection(geog,geog) regression in rare cases
- #2454, Verhaltensweise der Funktionen "ST_PixelAsXXX" in Bezug auf den Parameter "exclude_nodata_value" fixiert
- #2489, Fix upgrades from 2.0 leaving stale function signatures
- #2525, Handhabung von SRID bei verschachtelten Kollektionen gelöst
- #2449, Beseitigung einer potentiell unendlichen Schleife beim Aufbau eines Index
- #2493, Fixierung der Verhaltensweise von "ST_DumpValues" wenn ein leerer Raster angegeben wird
- #2502, Installationsschema für postgis_topology_scripts_installed() fixiert
- #2504, Schutzverletzung bei falschem Aufruf von pgsql2shp beseitigt
- #2512, Unterstützung von Fremdtabellen und materialisierten Views in "raster_columns" und "raster_overviews"

A.16.3 Verbesserungen

- #2478, Unterstützung für tiger 2013
 - #2463, genaue Längenberechnung für Bogengeometrien
-

A.17 Release 2.1.0

Freigabedatum: 2013/08/17

This is a minor release addressing both bug fixes and performance and functionality enhancements addressing issues since 2.0.3 release. If you are upgrading from 2.0+, only a soft upgrade is required. If you are upgrading from 1.5 or earlier, a hard upgrade is required.

A.17.1 Wichtige Änderungen

#1653, Removed srid parameter from ST_Resample(raster) and variants with reference raster no longer apply reference raster's SRID.

#1962 ST_Segmentize - As a result of the introduction of geography support, The construct: `SELECT ST_Segmentize('LINESTRING(1 2, 3 4)', 0.5);` will result in ambiguous function error

#2026, ST_Union(raster) vereinigt nun alle Bänder von allen Rastern

#2089, liblwgeom: lwgeom_set_handlers ersetzt lwgeom_init_allocators.

#2150, regular_blocking is no longer a constraint. column of same name in raster_columns now checks for existence of spatially_unique and coverage_tile constraints

ST_Intersects(raster, geometry) verhält sich genau so wie ST_Intersects(geometry, raster).

Bei der Punktvariante von ST_SetValue(raster) wurde die SRID der eingegebenen Geometrien und Raster bisher nicht überprüft.

Die Parameter Azimut und Höhe von ST_Hillshade werden nun in Grad anstatt in Bogenmaß angegeben.

ST_Slope und ST_Aspect geben die Zellwerte nun in Grad statt in Bogenmaß aus.

#2104, ST_World2RasterCoord, ST_World2RasterCoordX und ST_World2RasterCoordY umbenannt in ST_WorldToRasterCoord, ST_WorldToRasterCoordX und ST_WorldToRasterCoordY. ST_Raster2WorldCoord, ST_Raster2WorldCoordX und ST_Raster2WorldCoordY umbenannt in ST_RasterToWorldCoord, ST_RasterToWorldCoordX und ST_RasterToWorldCoordY

ST_Estimated_Extent in ST_EstimatedExtent umbenannt

ST_Line_Interpolate_Point in ST_LineInterpolatePoint umbenannt

ST_Line_Substring in ST_LineSubstring umbenannt

ST_Line_Locate_Point in ST_LineLocatePoint umbenannt

ST_Force_XXX in ST_ForceXXX umbenannt

ST_MapAlgebraFctNgb and 1 and 2 raster variants of ST_MapAlgebraFct. Use ST_MapAlgebra instead

1 and 2 raster variants of ST_MapAlgebraExpr. Use expression variants of ST_MapAlgebra instead

A.17.2 Neue Funktionalität

- Siehe http://postgis.net/docs/manual-2.1/PostGIS_Special_Functions_Index.html#NewFunctions_2_1 für eine vollständige Liste der neuen Funktionen

#310, ST_DumpPoints nun als C-Funktion (Nathan Wagner) und viel schneller

#739, UpdateRasterSRID()

#945, improved join selectivity, N-D selectivity calculations, user accessible selectivity and stats reader functions for testing (Paul Ramsey / OpenGeo)

toTopoGeom with TopoGeometry sink (Sandro Santilli / Vizzuality)

clearTopoGeom (Sandro Santilli / Vizzuality)

ST_Segmentize(geography) (Paul Ramsey / OpenGeo)

ST_DelaunayTriangles (Sandro Santilli / Vizzuality)

ST_NearestValue, ST_Neighborhood (Bborie Park / UC Davis)

ST_PixelAsPoint, ST_PixelAsPoints (Bborie Park / UC Davis)

ST_PixelAsCentroid, ST_PixelAsCentroids (Bborie Park / UC Davis)

ST_Raster2WorldCoord, ST_World2RasterCoord (Bborie Park / UC Davis)

Zusätzliche Funktionen für räumliche Beziehungen von Raster/Raster (ST_Contains, ST_ContainsProperly, ST_Covers, ST_CoveredBy, ST_Disjoint, ST_Overlaps, ST_Touches, ST_Within, ST_DWithin, ST_DFullyWithin) (Bborie Park / UC Davis)

Added array variants of ST_SetValues() to set many pixel values of a band in one call (Bborie Park / UC Davis)

#1293, ST_Resize(raster) um die Rastergröße über width/height ändern zu können

#1627, "tiger_geocoder" Paket als PostgreSQL EXTENSION

#1643, #2076, Upgrade des Tiger Geokodierers für die Tiger Daten von 2011 und 2012 (Regina Obe / Paragon Corporation)
Finanziert von der Hunter Systems Group

GEOMETRYCOLLECTION Unterstützung für ST_MakeValid (Sandro Santilli / Vizzuality)

#1709, ST_NotSameAlignmentReason(raster, raster)

#1818, ST_GeomFromGeoHash und Freunde (Jason Smith (darkpanda))

#1856, reverse geocoder rating setting for prefer numbered highway name

ST_PixelOfValue (Bborie Park / UC Davis)

Typumwandlung für PostgreSQL geotypes (POINT/PATH/POLYGON).

Added geomval array variant of ST_SetValues() to set many pixel values of a band using a set of geometries and corresponding values in one call (Bborie Park / UC Davis)

ST_Tile(raster), um einen Raster in Kacheln zu zerlegen (Bborie Park / UC Davis)

#1895, neuer Algorithmus zum Auftrennen von Knoten im R-Baum (Alex Korotkov)

#2011, ST_DumpValues um Raster als ein Feld auszugeben (Bborie Park / UC Davis)

#2018, ST_Distance für CircularString, CurvePolygon, MultiCurve, MultiSurface, CompoundCurve

#2030, n-raster (und n-band) ST_MapAlgebra (Bborie Park / UC Davis)

#2193, Utilize PAGC parser as drop in replacement for tiger normalizer (Steve Woodbridge, Regina Obe)

#2210, ST_MinConvexHull(raster)

lwgeom_from_geojson in liblwgeom (Sandro Santilli / Vizzuality)

#1687, ST_Simplify für TopoGeometry (Sandro Santilli / Vizzuality)

#2228, TopoJSON Ausgabe für TopoGeometry (Sandro Santilli / Vizzuality)

#2123, ST_FromGDALRaster

#613, der Parametertyp für ST_SetGeoReference ist nun numerisch anstatt Text

#2276, ST_AddBand(raster) Variante für out-db Rasterbänder

#2280, ST_Summary(raster)

#2163, ST_TPI für Raster (Nathaniel Clay)

#2164, ST_TRI für Raster (Nathaniel Clay)

#2302, ST_Roughness für Raster (Nathaniel Clay)

#2290, ST_ColorMap(raster) zum Erzeugen von RGBA Rasterbändern

#2254, Add SFCGAL backend support. (Backend selection through postgis.backend var) Functions available both through GEOS or SFCGAL: ST_Intersects, ST_3DIntersects, ST_Intersection, ST_Area, ST_Distance, ST_3DDistance New functions available only with SFCGAL backend: ST_3DIntersection, ST_Tessellate, ST_3DArea, ST_Extrude, ST_ForceLHR ST_Orientation, ST_Minkowski, ST_StraightSkeleton postgis_sfcgal_version New function available in PostGIS: ST_ForceSFS (Olivier Courtin and Hugo Mercier / Oslandia)

A.17.3 Verbesserungen

Für Einzelheiten zu den neuen Funktionen, siehe Section 14.12.5.

Viel schnelleres ST_Union und ST_Clip, sowie viele neue Funktionen für Raster

For geometry/geography better planner selectivity and a lot more functions.

#823, tiger geocoder: Make loader_generate_script download portion less greedy

#826, raster2pgsql no longer defaults to padding tiles. Flag -P can be used to pad tiles

#1363, ST_AddBand(raster, ...) array Version nach C portiert

#1364, ST_Union(raster, ...) Aggregierungsfunktion nach C portiert

#1655, Additional default values for parameters of ST_Slope

#1661, Add aggregate variant of ST_SameAlignment

#1719, Add support for Point and GeometryCollection ST_MakeValid inputs

#1780, Unterstützung von ST_GeoHash für den geographischen Datentyp

#1796, Big performance boost for distance calculations in geography

#1802, improved function interruptibility.

#1823, add parameter in ST_AsGML to use id column for GML 3 output (become mandatory since GML 3.2.1)

#1856, tiger geocoder: reverse geocoder rating setting for prefer numbered highway name

#1938, Refactor basic ST_AddBand to add multiple new bands in one call

#1978, wrong answer when calculating length of a closed circular arc (circle)

#1989, Preprocess input geometry to just intersection with raster to be clipped

#2021, Added multi-band support to ST_Union(raster, ...) aggregate function

#2006, better support of ST_Area(geography) over poles and dateline

#2065, ST_Clip(raster, ...) ist nun eine C-Funktion

#2069, Added parameters to ST_Tile(raster) to control padding of tiles

#2078, New variants of ST_Slope, ST_Aspect and ST_HillShade to provide solution to handling tiles in a coverage

#2097, Added RANGE uniontype option for ST_Union(raster)

#2105, Added ST_Transform(raster) variant for aligning output to reference raster

#2119, Rasters passed to ST_Resample(), ST_Rescale(), ST_Reskew(), and ST_SnapToGrid() no longer require an SRID

#2141, More verbose output when constraints fail to be added to a raster column

#2143, Changed blocksize constraint of raster to allow multiple values

#2148, Constraint "coverage_tile" für Raster hinzugefügt

#2149, Constraint "spatially_unique" für Raster hinzugefügt

TopologySummary output now includes unregistered layers and a count of missing TopoGeometry objects from their natural layer.

Neuer, optionaler Parameter für ST_HillShade(), ST_Aspect() und ST_Slope() zur Interpolation von NODATA Pixel vor der Operation.

Point variant of ST_SetValue(raster) is now a wrapper around geomval variant of ST_SetValues(rast).

Proper support for raster band's isnodata flag in core API and loader.

Zusätzliche Standardwerte für die Parameter von ST_Aspect und ST_HillShade

#2178, ST_Summary now advertises presence of known srid with an [S] flag

- #2202, libjson-c ist jetzt optional (--without-json configure switch)
- #2213, Unterstützung für libjson-c 0.10+
- #2231, raster2pgsql supports user naming of filename column with -n
- #2200, ST_Union(raster, uniontype) vereinigt alle Bänder aller Raster
- #2264, postgis_restore.pl support for restoring into databases with postgis in a custom schema
- #2244, emit warning when changing raster's georeference if raster has out-db bands
- #2222, add parameter OutAsIn to flag whether ST_AsBinary should return out-db bands as in-db bands

A.17.4 Bugfixes

- #1839, handling of subdatasets in GeoTIFF in raster2pgsql.
 - #1840, fix logic of when to compute # of tiles in raster2pgsql.
 - #1870, align the docs and actual behavior of raster's ST_Intersects
 - #1872, ST_ApproxSummarystats fixiert, indem die Division durch Null verhindert wird
 - #1875, ST_SummaryStats returns NULL for all parameters except count when count is zero
 - #1932, fix raster2pgsql of syntax for index tablespaces
 - #1936, ST_GeomFromGML on CurvePolygon causes server crash
 - #1939, remove custom data types: summarystats, histogram, quantile, valuecount
 - #1951, remove crash on zero-length linestrings
 - #1957, ST_Distance to a one-point LineString returns NULL
 - #1976, Geography point-in-ring code overhauled for more reliability
 - #1981, cleanup of unused variables causing warnings with gcc 4.6+
 - #1996, support POINT EMPTY in GeoJSON output
 - #2062, improve performance of distance calculations
 - #2057, Fixed linking issue for raster2psql to libpq
 - #2077, Fixed incorrect values returning from ST_Hillshade()
 - #2019, ST_FlipCoordinates does not update bbox
 - #2100, ST_AsRaster may not return raster with specified pixel type
 - #2126, Better handling of empty rasters from ST_ConvexHull()
 - #2165, ST_NumPoints regression failure with CircularString
 - #2168, ST_Distance ist nicht immer kommutativ
 - #2182, Fix issue with outdb rasters with no SRID and ST_Resize
 - #2188, Fix function parameter value overflow that caused problems when copying data from a GDAL dataset
 - #2198, Fix incorrect dimensions used when generating bands of out-db rasters in ST_Tile()
 - #2201, ST_GeoHash wrong on boundaries
 - #2203, Changed how rasters with unknown SRID and default geotransform are handled when passing to GDAL Warp API
 - #2215, Fixed raster exclusion constraint for conflicting name of implicit index
 - #2251, Fix bad dimensions when rescaling rasters with default geotransform matrix
 - #2133, Fix performance regression in expression variant of ST_MapAlgebra
-

- #2257, GBOX variables not initialized when testing with empty geometries
- #2271, Prevent parallel make of raster
- #2282, Fix call to undefined function nd_stats_to_grid() in debug mode
- #2307, ST_MakeValid gibt ungültige Geometrien aus
- #2309, Remove confusing INFO message when trying to get SRS info
- #2336, FIPS 20 (KS) causes wildcard expansion to wget all files
- #2348, Provide raster upgrade path for 2.0 to 2.1
- #2351, st_distance between geographies wrong
- #2359, Fix handling of schema name when adding overview constraints
- #2371, Support GEOS versions with more than 1 digit in micro
- #2383, Remove unsafe use of \ from raster warning message
- #2384, Incorrect variable datatypes for ST_Neighborhood

A.17.5 Known Issues

- #2111, Raster bands can only reference the first 256 bands of out-db rasters

A.18 Release 2.0.5

Freigabedatum: 2014/03/31

This is a bug fix release, addressing issues that have been filed since the 2.0.4 release. If you are using PostGIS 2.0+ a soft upgrade is required. For users of PostGIS 1.5 or below, a hard upgrade is required.

A.18.1 Bugfixes

- #2494, memcpy in GIST Index vermeiden
- #2502, Installationsschema für postgis_topology_scripts_installed() fixiert
- #2504, Schutzverletzung bei falschem Aufruf von pgsql2shp beseitigt
- #2528, Fix memory leak in ST_Split / lwline_split_by_line
- #2532, Add missing raster/geometry commutator operators
- #2533, Remove duplicated signatures
- #2552, Handhabung von NULL-Rastern in ST_AsPNG, ST_AsTIFF und ST_AsJPEG fixiert
- #2555, Fix parsing issue of range arguments of ST_Reclass
- #2589, Remove use of unnecessary void pointers
- #2607, Cannot open more than 1024 out-db files in process
- #2610, Ensure face splitting algorithm uses the edge index
- #2619, Empty ring array in GeoJSON polygon causes crash
- #2638, die geographische Entfernungsberechnung ist bei M-Geometrien manchmal falsch

A.18.2 Wichtige Änderungen

- ##2514, Change raster license from GPL v3+ to v2+, allowing distribution of PostGIS Extension as GPLv2.

A.19 Release 2.0.4

Freigabedatum: 2013/09/06

This is a bug fix release, addressing issues that have been filed since the 2.0.3 release. If you are using PostGIS 2.0+ a soft upgrade is required. For users of PostGIS 1.5 or below, a hard upgrade is required.

A.19.1 Bugfixes

#2110, Equality operator between EMPTY and point on origin

Allow adding points at precision distance with TopoGeo_addPoint

#1968, Fix missing edge from toTopoGeom return

#2165, ST_NumPoints regression failure with CircularString

#2168, ST_Distance ist nicht immer kommutativ

#2186, gui progress bar updates too frequent

#2201, ST_GeoHash wrong on boundaries

#2257, GBOX variables not initialized when testing with empty geometries

#2271, Prevent parallel make of raster

#2267, Server crash from analyze table

#2277, potential segfault removed

#2307, ST_MakeValid gibt ungültige Geometrien aus

#2351, st_distance between geographies wrong

#2359, Incorrect handling of schema for overview constraints

#2371, Support GEOS versions with more than 1 digit in micro

#2372, Cannot parse space-padded KML coordinates

Fix build with systemwide liblwgeom installed

#2383, Fix unsafe use of \ in warning message

#2410, Fix segmentize of collinear curve

#2412, ST_LineToCurve support for lines with less than 4 vertices

#2415, ST_Multi Unterstützung für COMPOUNDCURVE und CURVEPOLYGON

#2420, ST_LineToCurve: require at least 8 edges to define a full circle

#2423, ST_LineToCurve: require all arc edges to form the same angle

#2424, ST_CurveToLine: add support for COMPOUNDCURVE in MULTICURVE

#2427, Make sure to retain first point of curves on ST_CurveToLine

A.19.2 Verbesserungen

#2269, Avoid uselessly detoasting full geometries on ANALYZE

A.19.3 Known Issues

#2111, Raster bands can only reference the first 256 bands of out-db rasters

A.20 Release 2.0.3

Freigabedatum: 2013/03/01

This is a bug fix release, addressing issues that have been filed since the 2.0.2 release. If you are using PostGIS 2.0+ a soft upgrade is required. For users of PostGIS 1.5 or below, a hard upgrade is required.

A.20.1 Bugfixes

#2126, Better handling of empty rasters from ST_ConvexHull()

#2134, Make sure to process SRS before passing it off to GDAL functions

Verschiedene Speicherlecks in liblwgeom fixiert

#2173, Fix robustness issue in splitting a line with own vertex also affecting topology building (#2172)

#2174, Fix usage of wrong function lwpoly_free()

#2176, Fix robustness issue with ST_ChangeEdgeGeom

#2184, Properly copy topologies with Z value

postgis_restore.pl support for mixed case geometry column name in dumps

#2188, Fix function parameter value overflow that caused problems when copying data from a GDAL dataset

#2216, More memory errors in MultiPolygon GeoJSON parsing (with holes)

Speicherleck im GeoJSON Parser fixiert

A.20.2 Verbesserungen

#2141, More verbose output when constraints fail to be added to a raster column

ST_ChangeEdgeGeom beschleunigt

A.21 Release 2.0.2

Freigabedatum: 2012/12/03

Dies ist eine Bugfix-Release, die sich mit Problemen befasst die seit Release 2.0.1 gemeldet wurden.

A.21.1 Bugfixes

#1287, Drop of "gist_geometry_ops" broke a few clients package of legacy_gist.sql for these cases

#1391, Errors during upgrade from 1.5

#1828, Poor selectivity estimate on ST_DWithin

#1838, error importing tiger/line data

#1869, ST_AsBinary is not unique added to legacy_minor/legacy.sql scripts

#1885, Missing field from tabblock table in tiger2010 census_loader.sql

#1891, Use LDFLAGS environment when building liblwgeom

#1900, Fix pgsql2shp for big-endian systems

#1932, Fix raster2pgsql for invalid syntax for setting index tablespace

#1936, ST_GeomFromGML on CurvePolygon causes server crash

#1955, ST_ModEdgeHeal and ST_NewEdgeHeal for doubly connected edges

#1957, ST_Distance to a one-point LineString returns NULL

#1976, Geography point-in-ring code overhauled for more reliability

#1978, wrong answer calculating length of closed circular arc (circle)

#1981, Remove unused but set variables as found with gcc 4.6+

#1987, Restore 1.5.x behaviour of ST_Simplify

#1989, Preprocess input geometry to just intersection with raster to be clipped

#1991, geocode really slow on PostgreSQL 9.2

#1996, support POINT EMPTY in GeoJSON output

#1998, Fix ST_{Mod,New}EdgeHeal joining edges sharing both endpoints

#2001, ST_CurveToLine has no effect if the geometry doesn't actually contain an arc

#2015, ST_IsEmpty('POLYGON(EMPTY)') gibt False zurück

#2019, ST_FlipCoordinates does not update bbox

#2025, Fix side location conflict at TopoGeo_AddLineString

#2026, improve performance of distance calculations

#2033, Fix adding a splitting point into a 2.5d topology

#2051, Fix excess of precision in ST_AsGeoJSON output

#2052, Fix buffer overflow in lwgeom_to_geojson

#2056, Fixed lack of SRID check of raster and geometry in ST_SetValue()

#2057, Fixed linking issue for raster2psql to libpq

#2060, Fix "dimension" check violation by GetTopoGeomElementArray

#2072, Removed outdated checks preventing ST_Intersects(raster) from working on out-db bands

#2077, Fixed incorrect answers from ST_Hillshade(raster)

#2092, Namespace issue with ST_GeomFromKML, ST_GeomFromGML for libxml 2.8+

#2099, Fix double free on exception in ST_OffsetCurve

#2100, ST_AsRaster() may not return raster with specified pixel type

#2108, Sicherstellen, dass ST_Line_Interpolate_Point immer einen POINT zurückgibt

#2109, Sicherstellen, dass ST_Centroid immer einen POINT zurückgibt

#2117, Sicherstellen, dass ST_PointOnSurface immer einen POINT zurückgibt

#2129, Fix SRID in ST_Homogenize output with collection input

#2130, Fix memory error in MultiPolygon GeoJson parsing

Update der URL von Maven jar

A.21.2 Verbesserungen

#1581, ST_Clip(raster, ...) no longer imposes NODATA on a band if the corresponding band from the source raster did not have NODATA

#1928, Accept array properties in GML input multi-geom input (Kashif Rasul and Shoaib Burq / SpacialDB)

#2082, Add indices on start_node and end_node of topology edge tables

#2087, Speedup topology.GetRingEdges using a recursive CTE

A.22 Release 2.0.1

Freigabedatum: 2012/06/22

Dies ist eine Bugfix-Release, die sich mit Problemen befasst die seit Release 2.0.0 gemeldet wurden.

A.22.1 Bugfixes

- #1264, fix `st_dwithin`(geog, geog, 0).
 - #1468 `shp2pgsql-gui` table column schema get shifted
 - #1694, fix building with clang. (vince)
 - #1708, improve restore of pre-PostGIS 2.0 backups.
 - #1714, more robust handling of high topology tolerance.
 - #1755, `ST_GeographyFromText` Unterstützung für höhere Dimensionen.
 - #1759, loading transformed shapefiles in raster enabled db.
 - #1761, handling of subdatasets in NetCDF, HDF4 and HDF5 in `raster2pgsql`.
 - #1763, `topology.toTopoGeom` use with custom `search_path`.
 - #1766, don't let `ST_RemEdge*` destroy peripheral `TopoGeometry` objects.
 - #1774, Clearer error on setting an edge geometry to an invalid one.
 - #1775, `ST_ChangeEdgeGeom` collision detection with 2-vertex target.
 - #1776, fix `ST_SymDifference`(empty, geom) to return geom.
 - #1779, install SQL comment files.
 - #1782, fix spatial reference string handling in raster.
 - #1789, fix false edge-node crossing report in `ValidateTopology`.
 - #1790, fix `toTopoGeom` handling of duplicated primitives.
 - #1791, fix `ST_Azimuth` with very close but distinct points.
 - #1797, fix `(ValidateTopology(xxx)).*` syntax calls.
 - #1805, den 900913 SRID Eintrag wieder übernehmen.
 - #1813, Only show readable relations in metadata tables.
 - #1819, fix floating point issues with `ST_World2RasterCoord` and `ST_Raster2WorldCoord` variants.
 - #1820 compilation on 9.2beta1.
 - #1822, topology load on PostgreSQL 9.2beta1.
 - #1825, fix prepared geometry cache lookup
 - #1829, fix uninitialized read in GeoJSON parser
 - #1834, revise postgis extension to only backup user specified `spatial_ref_sys`
 - #1839, handling of subdatasets in GeoTIFF in `raster2pgsql`.
 - #1840, fix logic of when to compute # of tiles in `raster2pgsql`.
 - #1851, fix `spatial_ref_system` parameters for EPSG:3844
 - #1857, fix failure to detect endpoint mismatch in `ST_AddEdge*Face*`
 - #1865, data loss in `postgis_restore.pl` when data rows have leading dashes.
 - #1867, catch invalid topology name passed to `topogeo_add*`
-

- #1872, ST_ApproxSummarystats fixiert, indem die Division durch Null verhindert wird
- #1873, fix ptarray_locate_point to return interpolated Z/M values for on-the-line case
- #1875, ST_SummaryStats returns NULL for all parameters except count when count is zero
- #1881, shp2pgsql-gui -- editing a field sometimes triggers removing row
- #1883, Geocoder install fails trying to run create_census_base_tables() (Brian Panulla)

A.22.2 Verbesserungen

More detailed exception message from topology editing functions.

- #1786, improved build dependencies
- #1806, speedup of ST_BuildArea, ST_MakeValid and ST_GetFaceGeometry.
- #1812, Add lwgeom_normalize in LIBLWGEOM for more stable testing.

A.23 Release 2.0.0

Freigabedatum: 2012/04/03

This is a major release. A hard upgrade is required. Yes this means a full dump reload and some special preparations if you are using obsolete functions. Refer to Section 2.10.2 for details on upgrading. Refer to Section 14.12.7 for more details and changed/new functions.

A.23.1 Tester - Unsere heimlichen Helden

We are most indebted to the numerous members in the PostGIS community who were brave enough to test out the new features in this release. No major release can be successful without these folk.

Below are those who have been most valiant, provided very detailed and thorough bug reports, and detailed analysis.

Andrea Peri - massenweise Topologie-Tests, Überprüfung auf Richtigkeit
 Andreas Forø Tollefsen - Raster-Tests
 Chris English - topology stress testing loader functions
 Salvatore Larosa - Stabilitätstest bezüglich Topologie
 Brian Hamlin - Benchmarking (also experimental experimental branches before they are folded into core) , general testing of various
 Mike Pease - Tests des Tiger Geokodierer - sehr detaillierte Problemlberichte
 Tom van Tilburg - Rastertest

A.23.2 Wichtige Änderungen

- #722, #302, Most deprecated functions removed (over 250 functions) (Regina Obe, Paul Ramsey)

"Unknown" SRID von -1 auf 0 geändert. (Paul Ramsey)

-- (most deprecated in 1.2) removed non-ST variants buffer, length, intersects (and internal functions renamed) etc.

-- If you have been using deprecated functions CHANGE your apps or suffer the consequences. If you don't see a function documented -- it ain't supported or it is an internal function. Some constraints in older tables were built with deprecated functions. If you restore you may need to rebuild table constraints with populate_geometry_columns(). If you have applications or tools that rely on deprecated functions, please refer to [?qandaentry] for more details.

- #944 geometry_columns is now a view instead of a table (Paul Ramsey, Regina Obe) for tables created the old way reads (srid, type, dims) constraints for geometry columns created with type modifiers reads from column definition

- #1081, #1082, #1084, #1088 - Management functions support typmod geometry column creation functions now default to typmod creation (Regina Obe)

#1083 probe_geometry_columns(), rename_geometry_table_constraints(), fix_geometry_columns(); removed - now obsolete with geometry_column view (Regina Obe)

#817 Umbenennung alter 3D-Funktionen entsprechend der Vereinbarung "ST_3D" (Nicklas Avén)

#548 (sorta), ST_NumGeometries, ST_GeometryN now returns 1 (or the geometry) instead of null for single geometries (Sandro Santilli, Maxime van Noppen)

A.23.3 Neue Funktionalität

KNN Gist index based centroid (<->) and box (<#>) distance operators (Paul Ramsey / funded by Vizzuality)

Support for TIN and PolyHedralSurface and enhancement of many functions to support 3D (Olivier Courtin / Oslandia)

Raster support integrated and documented (Pierre Racine, Jorge Arévalo, Mateusz Loskot, Sandro Santilli, David Zwarg, Regina Obe, Bborie Park) (Company developer and funding: University Laval, Deimos Space, CadCorp, Michigan Tech Research Institute, Azavea, Paragon Corporation, UC Davis Center for Vectorborne Diseases)

Räumliche Indizes 3D-Fähig machen - in Arbeit (Paul Ramsey, Mark Cave-Ayland)

Topology support improved (more functions), documented, testing (Sandro Santilli / Faunalia for RT-SIGTA), Andrea Peri, Regina Obe, Jose Carlos Martinez Llari

3D relationship and measurement support functions (Nicklas Avén)

ST_3DDistance, ST_3DClosestPoint, ST_3DIntersects, ST_3DShortestLine und mehr ...

N-dimensionale räumliche Indizes (Paul Ramsey / OpenGeo)

ST_Split (Sandro Santilli / Faunalia für RT-SIGTA)

ST_IsValidDetail (Sandro Santilli / Faunalia für RT-SIGTA)

ST_MakeValid (Sandro Santilli / Faunalia für RT-SIGTA)

ST_RemoveRepeatedPoints (Sandro Santilli / Faunalia für RT-SIGTA)

ST_GeometryN and ST_NumGeometries support for non-collections (Sandro Santilli)

ST_IsCollection (Sandro Santilli, Maxime van Noppen)

ST_SharedPaths (Sandro Santilli / Faunalia für RT-SIGTA)

ST_Snap (Sandro Santilli)

ST_RelateMatch (Sandro Santilli / Faunalia für RT-SIGTA)

ST_ConcaveHull (Regina Obe und Leo Hsu / Paragon Corporation)

ST_UnaryUnion (Sandro Santilli / Faunalia für RT-SIGTA)

ST_AsX3D (Regina Obe / Arrival 3D Finanzierung)

ST_OffsetCurve (Sandro Santilli, Rafal Magda)

ST_GeomFromGeoJSON (Kashif Rasul, Paul Ramsey / Vizzuality Funding)

A.23.4 Verbesserungen

Made shape file loader tolerant of truncated multibyte values found in some free worldwide shapefiles (Sandro Santilli)

Lots of bug fixes and enhancements to shp2pgsql Beefing up regression tests for loaders Reproject support for both geometry and geography during import (Jeff Adams / Azavea, Mark Cave-Ayland)

pgsql2shp conversion from predefined list (Loic Dachary / Mark Cave-Ayland)

Shp-pgsql GUI Lader - unterstützt das Laden von mehreren Dateien gleichzeitig. (Mark Leslie)

Extras - upgraded tiger_geocoder from using old TIGER format to use new TIGER shp and file structure format (Stephen Frost)

Extras - revised tiger_geocoder to work with TIGER census 2010 data, addition of reverse geocoder function, various bug fixes, accuracy enhancements, limit max result return, speed improvements, loading routines. (Regina Obe, Leo Hsu / Paragon Corporation / funding provided by Hunter Systems Group)

Overall Documentation proofreading and corrections. (Kasif Rasul)

Cleanup PostGIS JDBC classes, revise to use Maven build. (Maria Arias de Reyna, Sandro Santilli)

A.23.5 Bugfixes

#1335 ST_AddPoint returns incorrect result on Linux (Even Rouault)

A.23.6 Release specific credits

We thank [U.S Department of State Human Information Unit \(HIU\)](#) and [Vizzuality](#) for general monetary support to get PostGIS 2.0 out the door.

A.24 Release 1.5.4

Freigabedatum: 2012/05/07

Dies ist eine Bugfix-Release, die sich mit Problemen befasst die seit Release 1.5.3 gemeldet wurden.

A.24.1 Bugfixes

#547, Speicherprobleme bei ST_Contains (Sandro Santilli)

#621, Problem finding intersections with geography (Paul Ramsey)

#627, PostGIS/PostgreSQL Prozess stürzt bei ungültigen Geometrien ab (Paul Ramsey)

#810, Genauere Flächenberechnung (Paul Ramsey)

#852, improve spatial predicates robustness (Sandro Santilli, Nicklas Avén)

#877, ST_Estimated_Extent gibt NULL für leere Tabellen zurück (Sandro Santilli)

#1028, Wenn ST_AsSVG fehlschlägt, schiesst es den Postgres Server ab (Paul Ramsey)

#1056, Fix boxes of arcs and circle stroking code (Paul Ramsey)

#1121, populate_geometry_columns using deprecated functions (Regin Obe, Paul Ramsey)

#1135, improve testsuite predictability (Andreas 'ads' Scherbaum)

#1146, images generator crashes (bronaugh)

#1170, North Pole intersection fails (Paul Ramsey)

#1179, ST_AsText crash with bad value (kjurka)

#1184, honour DESTDIR in documentation Makefile (Bryce L Nordgren)

#1227, Serverabsturz bei ungültigem GML

#1252, SRID appearing in WKT (Paul Ramsey)

#1264, st_dwithin(g, g, 0) funktioniert nicht (Paul Ramsey)

#1344, Export von Tabellen mit ungültigen Geometrien erlauben (Sandro Santilli)

#1389, falscher proj4text für SRID 31300 und 31370 (Paul Ramsey)

#1406, shp2pgsql crashes when loading into geography (Sandro Santilli)

- #1595, fixed SRID redundancy in ST_Line_SubString (Sandro Santilli)
- #1596, Überprüfung von SRID in UpdateGeometrySRID (Mike Toews, Sandro Santilli)
- #1602, fix ST_Polygonize to retain Z (Sandro Santilli)
- #1697, fix crash with EMPTY entries in GiST index (Paul Ramsey)
- #1772, fix ST_Line_Locate_Point with collapsed input (Sandro Santilli)
- #1799, Protect ST_Segmentize from max_length=0 (Sandro Santilli)
- Änderung der Parameterreihenfolge in 900913 (Paul Ramsey)
- Die Kompilation mittels "gmake" wird jetzt unterstützt (Greg Troxel)

A.25 Release 1.5.3

Freigabedatum: 2011/06/25

Dies ist eine Bugfix-Release, die sich mit Problemen befasst die seit Release 1.5.2 gemeldet wurden. Falls Sie PostGIS 1.3+ am Laufen haben ist ein "Soft Upgrade" ausreichend, ansonsten empfehlen wir ein "Hard Upgrade"

A.25.1 Bugfixes

- #1056, erzeugt korrekte Bboxes für Bogengeometrien, bereinigt Indexfehler (Paul Ramsey)
- #1007, ST_IsValid crash fix requires GEOS 3.3.0+ or 3.2.3+ (Sandro Santilli, reported by Birgit Laggner)
- #940, Unterstützung für PostgreSQL 9.1 beta 1 (Regina Obe, Paul Ramsey, Patch von stl)
- #845, ST_Intersects Präzisionsfehler (Sandro Santilli, Nicklas Avén) Gemeldet von cdestigter
- #884, Unsichere Ergebnisse mit ST_Within, ST_Intersects (Chris Hodgson)
- #779, shp2pgsql -S option seems to fail on points (Jeff Adams)
- #666, ST_DumpPoints is not null safe (Regina Obe)
- #631, Update NZ projections for grid transformation support (jpalmer)
- #630, Peculiar Null treatment in arrays in ST_Collect (Chris Hodgson) Reported by David Bitner
- #624, Speicherleck in ST_GeogFromText (ryang, Paul Ramsey)
- #609, Bad source code in manual section 5.2 Java Clients (simoc, Regina Obe)
- #604, shp2pgsql usage touchups (Mike Toews, Paul Ramsey)
- #573 ST_Union versagt bei einer Gruppe von Linienzügen. Kein PostGIS Bug, in GEOS 3.3.0 bereinigt
- #457 ST_CollectionExtract returns non-requested type (Nicklas Avén, Paul Ramsey)
- #441 ST_AsGeoJson Bbox on GeometryCollection error (Olivier Courtin)
- #411 Ability to backup invalid geometries (Sando Santilli) Reported by Regione Toscana
- #409 ST_AsSVG - degraded (Olivier Courtin) Reported by Sdikiy
- #373 Documentation syntax error in hard upgrade (Paul Ramsey) Reported by psvensso

A.26 Release 1.5.2

Freigabedatum: 2010/09/27

Dies ist eine Bugfix-Release, die sich mit Problemen befasst die seit Release 1.5.1 gemeldet wurden. Falls Sie PostGIS 1.3+ am Laufen haben ist ein "Soft Upgrade" ausreichend, ansonsten empfehlen wir ein "Hard Upgrade"

A.26.1 Bugfixes

Loader: Die Handhabung von leeren (0-Knoten) Geometrien in Shapefiles bereinigt. (Sandro Santilli)

#536, Geography ST_Intersects, ST_Covers, ST_CoveredBy und Geometry ST_Equals verwenden keinen räumlichen Index (Regina Obe, Nicklas Aven)

#573, Improvement to ST_Contains geography (Paul Ramsey)

Loader: Add support for command-q shutdown in Mac GTK build (Paul Ramsey)

#393, Loader: Add temporary patch for large DBF files (Maxime Guillaud, Paul Ramsey)

#507, Fix wrong OGC URN in GeoJSON and GML output (Olivier Courtin)

spatial_ref_sys.sql Datumsumwandlung für das Koordinatenreferenzsystem SRID 3021 hinzugefügt (Paul Ramsey)

Geography - remove crash for case when all geographies are out of the estimate (Paul Ramsey)

#469, array_aggregation Fehler beseitigt (Greg Stark, Paul Ramsey)

#532, Temporary geography tables showing up in other user sessions (Paul Ramsey)

#562, ST_Dwithin Fehler bei großer "Geography" (Paul Ramsey)

#513, shape loading GUI tries to make spatial index when loading DBF only mode (Paul Ramsey)

#527, shape loading GUI should always append log messages (Mark Cave-Ayland)

#504, shp2pgsql sollte xmin/xmax Attribute umbenennen (Sandro Santilli)

#458, postgis_comments being installed in contrib instead of version folder (Mark Cave-Ayland)

#474, Analyze auf eine Tabelle mit geographischen Spalten führt zu Serverabsturz (Paul Ramsey)

#581, LWGEOM-expand produces inconsistent results (Mark Cave-Ayland)

#513, Einen Filter für dbf zur shp2pgsql-gui hinzugefügt und das Hochladen nur von dbf ermöglicht (Paul Ramsey)

Weitere Probleme bei der Kompilation mit PostgreSQL 9.0 fixiert (Mark Cave-Ayland)

#572, Password whitespace for Shape File (Mark Cave-Ayland)

#603, shp2pgsql: "-w" erzeugt ungültiges WKT für MULTI*-Objekte. (Mark Cave-Ayland)

A.27 Release 1.5.1

Freigabedatum: 2010/03/11

This is a bug fix release, addressing issues that have been filed since the 1.4.1 release. If you are running PostGIS 1.3+, a soft upgrade is sufficient otherwise a hard upgrade is recommended.

A.27.1 Bugfixes

#410, update embedded bbox when applying ST_SetPoint, ST_AddPoint ST_RemovePoint to a linestring (Paul Ramsey)

#411, allow dumping tables with invalid geometries (Sandro Santilli, for Regione Toscana-SIGTA)

#414, include geography_columns view when running upgrade scripts (Paul Ramsey)

#419, allow support for multilinestring in ST_Line_Substring (Paul Ramsey, for Lidwala Consulting Engineers)

#421, fix computed string length in ST_AsGML() (Olivier Courtin)

#441, fix GML generation with heterogeneous collections (Olivier Courtin)

#443, incorrect coordinate reversal in GML 3 generation (Olivier Courtin)

#450, #451, wrong area calculation for geography features that cross the date line (Paul Ramsey)

Unterstützung für die bevorstehenden 9.0 PgSQL Release sichergestellt (Paul Ramsey)

A.28 Release 1.5.0

Freigabedatum: 2010/02/04

This release provides support for geographic coordinates (lat/lon) via a new GEOGRAPHY type. Also performance enhancements, new input format support (GML,KML) and general upkeep.

A.28.1 API Stabilität

The public API of PostGIS will not change during minor (0.0.X) releases.

Der `~=` Operator überprüft nun auf Gleichheit des Umgebungsrechtecks anstelle von Gleichheit der tatsächlichen Geometrien.

A.28.2 Kompatibilität

GEOS, Proj4, und LibXML2 sind nun verbindliche Abhängigkeiten

Untere Bibliotheksversionen stellen die *Mindestanforderung* für PostGIS 1.5

PostgreSQL 8.3 und höher auf allen Plattformen

Nur GEOS 3.1 und höher (GEOS 3.2+ um alle Funktionen zu nutzen)

LibXML2 2.5+ in Bezug auf die neue ST_GeomFromGML/KML Funktionalität

Proj4 - nur 4.5 und höher

A.28.3 Neue Funktionalität

Section [14.12.9](#)

Berechnungen mittels Hausdorff-Metrik hinzugefügt ([#209](#)) (Vincent Picavet)

Added parameters argument to ST_Buffer operation to support one-sided buffering and other buffering styles (Sandro Santilli)

Addition of other Distance related visualization and analysis functions (Nicklas Aven)

- ST_ClosestPoint
- ST_DFullyWithin
- ST_LongestLine
- ST_MaxDistance
- ST_ShortestLine

ST_DumpPoints (Maxime van Noppen)

KML, GML Eingabe über ST_GeomFromGML und ST_GeomFromKML (Olivier Courtin)

Extract homogeneous collection with ST_CollectionExtract (Paul Ramsey)

Add measure values to an existing linestring with ST_AddMeasure (Paul Ramsey)

History table implementation in utils (George Silva)

Geographischer Datentyp und unterstützende Funktionen

- Sphärische Algorithmen (Dave Skea)
- Objekt/Index Implementierung (Paul Ramsey)
- Selectivity implementation (Mark Cave-Ayland)
- Serialisierung von KML, GML und JSON (Olivier Courtin)
- ST_Area, ST_Distance, ST_DWithin, ST_GeogFromText, ST_GeogFromWKB, ST_Intersects, ST_Covers, ST_Buffer (Paul Ramsey)

A.28.4 Verbesserungen

Verbesserung der Rechenleistung von ST_Distance (Nicklas Aven)

Update der Dokumentation und Verbesserungen (Regina Obe, Kevin Neufeld)

Tests und Qualitätskontrolle (Regina Obe)

PostGIS 1.5 support PostgreSQL 8.5 trunk (Guillaume Lelarge)

Win32 support and improvement of core shp2pgsql-gui (Mark Cave-Ayland)

In situ 'make check' Unterstützung (Paul Ramsey)

A.28.5 Bugfixes

<http://trac.osgeo.org/postgis/query?status=closed&milestone=PostGIS+1.5.0&order=priority>

A.29 Release 1.4.0

Freigabedatum: 2009/07/24

This release provides performance enhancements, improved internal structures and testing, new features, and upgraded documentation. If you are running PostGIS 1.1+, a soft upgrade is sufficient otherwise a hard upgrade is recommended.

A.29.1 API Stabilität

Ab der 1.4 Release Serie keine Änderung der öffentlichen API bei Unterversionen/"Minor Releases".

A.29.2 Kompatibilität

Untere Versionen sind die *Mindestanforderungen* für PostGIS 1.4

PostgreSQL 8.2 und höher; auf allen Plattformen

Nur GEOS 3.0 und höher

Nur PROJ4 4.5 und höher

A.29.3 Neue Funktionalität

ST_Union() uses high-speed cascaded union when compiled against GEOS 3.1+ (Paul Ramsey)

ST_ContainsProperly() benötigt GEOS 3.1+

ST_Intersects(), ST_Contains(), ST_Within() use high-speed cached prepared geometry against GEOS 3.1+ (Paul Ramsey / funded by Zonar Systems)

Vastly improved documentation and reference manual (Regina Obe & Kevin Neufeld)

Figures and diagram examples in the reference manual (Kevin Neufeld)

ST_IsValidReason() returns readable explanations for validity failures (Paul Ramsey)

ST_GeoHash() returns a geohash.org signature for geometries (Paul Ramsey)

GTK+ multi-platform GUI for shape file loading (Paul Ramsey)

ST_LineCrossingDirection() returns crossing directions (Paul Ramsey)

ST_LocateBetweenElevations() gibt eine Teilzeichenfolge, basierend auf der Z-Ordinate zurück. (Paul Ramsey)

Geometry parser returns explicit error message about location of syntax errors (Mark Cave-Ayland)

ST_AsGeoJSON() gibt eine als JSON formatierte Geometrie zurück (Olivier Courtin)

Populate_Geometry_Columns() -- fügt Datensätze für Tabellen und Views automatisch zu geometry_columns hinzu (Kevin Neufeld)

ST_MinimumBoundingCircle() -- gibt das kleinste Kreispolygon zurück, welches die Geometrie umfasst (Bruce Rindahl)

A.29.4 Verbesserungen

Core geometry system moved into independent library, liblwgeom. (Mark Cave-Ayland)

Neues Build-System nutzt den PostgreSQL "pgxs" Bootstrap. (Mark Cave-Ayland)

Debugging framework formalized and simplified. (Mark Cave-Ayland)

All build-time #defines generated at configure time and placed in headers for easier cross-platform support (Mark Cave-Ayland)

Logging framework formalized and simplified (Mark Cave-Ayland)

Expanded and more stable support for CIRCULARSTRING, COMPOUNDCURVE and CURVEPOLYGON, better parsing, wider support in functions (Mark Leslie & Mark Cave-Ayland)

Improved support for OpenSolaris builds (Paul Ramsey)

Improved support for MSVC builds (Mateusz Loskot)

Update der KML Unterstützung (Olivier Courtin)

Framework für den Komponententest von liblwgeom (Paul Ramsey)

Neues Framework für umfassende Tests an allen PostGIS Funktionen (Regine Obe)

Verbesserung der Rechenleistung bei allen geometrischen Aggregatfunktionen (Paul Ramsey)

Unterstützung für das kommende PostgreSQL 8.4 (Mark Cave-Ayland, Talha Bin Rizwan)

Shp2pgsql und pgsq2shp re-worked to depend on the common parsing/unparsing code in liblwgeom (Mark Cave-Ayland)

Use of PDF DbLatex to build PDF docs and preliminary instructions for build (Jean David Techer)

Automated User documentation build (PDF and HTML) and Developer Doxygen Documentation (Kevin Neufeld)

Automated build of document images using ImageMagick from WKT geometry text files (Kevin Neufeld)

CSS für eine attraktivere HTML-Dokumentation (Dane Springmeyer)

A.29.5 Bugfixes

<http://trac.osgeo.org/postgis/query?status=closed&milestone=PostGIS+1.4.0&order=priority>

A.30 Release 1.3.6

Freigabedatum: 2009/05/04

If you are running PostGIS 1.1+, a soft upgrade is sufficient otherwise a hard upgrade is recommended. This release adds support for PostgreSQL 8.4, exporting prj files from the database with shape data, some crash fixes for shp2pgsql, and several small bug fixes in the handling of "curve" types, logical error importing dbf only files, improved error handling of AddGeometryColumns.

A.31 Release 1.3.5

Freigabedatum: 2008/12/15

If you are running PostGIS 1.1+, a soft upgrade is sufficient otherwise a hard upgrade is recommended. This release is a bug fix release to address a failure in ST_Force_Collection and related functions that critically affects using MapServer with LINE layers.

A.32 Release 1.3.4

Freigabedatum: 2008/11/24

This release adds support for GeoJSON output, building with PostgreSQL 8.4, improves documentation quality and output aesthetics, adds function-level SQL documentation, and improves performance for some spatial predicates (point-in-polygon tests).

Bug fixes include removal of crashers in handling circular strings for many functions, some memory leaks removed, a linear referencing failure for measures on vertices, and more. See the NEWS file for details.

A.33 Release 1.3.3

Freigabedatum: 2008/04/12

This release fixes bugs shp2pgsql, adds enhancements to SVG and KML support, adds a ST_SimplifyPreserveTopology function, makes the build more sensitive to GEOS versions, and fixes a handful of severe but rare failure cases.

A.34 Release 1.3.2

Freigabedatum: 2007/12/01

This release fixes bugs in ST_EndPoint() and ST_Envelope, improves support for JDBC building and OS/X, and adds better support for GML output with ST_AsGML(), including GML3 output.

A.35 Release 1.3.1

Freigabedatum: 2007/08/13

This release fixes some oversights in the previous release around version numbering, documentation, and tagging.

A.36 Release 1.3.0

Freigabedatum: 2007/08/09

This release provides performance enhancements to the relational functions, adds new relational functions and begins the migration of our function names to the SQL-MM convention, using the spatial type (SP) prefix.

A.36.1 Zusätzliche Funktionalität

JDBC: Hibernate Dialekt hinzugefügt (herzlichen Dank an Norman Barker)

Added ST_Covers and ST_CoveredBy relational functions. Description and justification of these functions can be found at <http://lin-ear-th-inking.blogspot.com/2007/06/subtleties-of-ogc-covers-spatial.html>

Die relationale Funktion "ST_DWithin" hinzugefügt.

A.36.2 Verbesserung der Rechenleistung

Added cached and indexed point-in-polygon short-circuits for the functions ST_Contains, ST_Intersects, ST_Within and ST_Disjoint

Added inline index support for relational functions (except ST_Disjoint)

A.36.3 Sonstige Änderungen

Extended curved geometry support into the geometry accessor and some processing functions

Migration der Funktionen entsprechend der SQL-MM Benennungsregel; Verwendung des Präfix (ST) für den räumlichen Datentyp.

Initiale Unterstützung von PostgreSQL 8.3 hinzugefügt

A.37 Release 1.2.1

Freigabedatum: 2007/01/11

Diese Release liefert Bugfixes in der PostgreSQL 8.2 Unterstützung und einige kleine Verbesserungen bei der Rechenleistung.

A.37.1 Änderungen

Fixed point-in-polygon shortcut bug in Within().

PostgreSQL 8.2 Handhabung von NULL fixiert.

Updated RPM spec files.

Added short-circuit for Transform() in no-op case.

JDBC: Fixed JTS handling for multi-dimensional geometries (thanks to Thomas Marti for hint and partial patch). Additionally, now JavaDoc is compiled and packaged. Fixed classpath problems with GCJ. Fixed pgjdbc 8.2 compatibility, losing support for jdk 1.3 and older.

A.38 Release 1.2.0

Freigabedatum: 2006/12/08

Diese Release liefert Definitionen für Datentypen gemeinsam mit Möglichkeiten zur Serialisierung/Deserialisierung SQL-MM definierter gekrümmter Geometrien sowie Verbesserungen bei der Rechenleistung.

A.38.1 Änderungen

Serialisierung/Deserialisierung für gekrümmte geometrische Datentypen

Added point-in-polygon shortcircuit to the Contains and Within functions to improve performance for these cases.

A.39 Release 1.1.6

Freigabedatum: 2006/11/02

This is a bugfix release, in particular fixing a critical error with GEOS interface in 64bit systems. Includes an updated of the SRS parameters and an improvement in reprojections (take Z in consideration). Upgrade is *encouraged*.

A.39.1 Upgraden

Wenn Sie von Version 1.0.3 oder höher aus upgraden, folgen Sie bitte dem **soft upgrade** Prozedere.

If you are upgrading from a release *between 1.0.0RC6 and 1.0.2* (inclusive) and really want a live upgrade read the **upgrade section** of the 1.0.3 release notes chapter.

Ein Upgrade einer Vorgängerversion von 1.0.0RC6 benötigt ein **hard upgrade**.

A.39.2 Bugfixes

C-API Wechsel für 64-bit Architekturen fixiert

loader/dumper: fixed regression tests and usage output

Bugfix von setSRID() in JDBC, thanks to Thomas Marti

A.39.3 Sonstige Änderungen

Verwendung der Z Ordinate bei Koordinatentransformationen

spatial_ref_sys.sql auf EPSG 6.11.1 erneuert

Simplified Version.config infrastructure to use a single pack of version variables for everything.

Include the Version.config in loader/dumper USAGE messages

Replace hand-made, fragile JDBC version parser with Properties

A.40 Release 1.1.5

Freigabedatum: 2006/10/13

This is an bugfix release, including a critical segfault on win32. Upgrade is *encouraged*.

A.40.1 Upgraden

Wenn Sie von Version 1.0.3 oder höher aus upgraden, folgen Sie bitte dem **soft upgrade** Prozedere.

If you are upgrading from a release *between 1.0.0RC6 and 1.0.2* (inclusive) and really want a live upgrade read the **upgrade section** of the 1.0.3 release notes chapter.

Ein Upgrade einer Vorgängerversion von 1.0.0RC6 benötigt ein **hard upgrade**.

A.40.2 Bugfixes

Fixed MingW link error that was causing pgsql2shp to segfault on Win32 when compiled for PostgreSQL 8.2

fixed nullpointer Exception in Geometry.equals() method in Java

Added EJB3Spatial.odt to fulfill the GPL requirement of distributing the "preferred form of modification"

Überholte Synchronisation aus dem JDBC JTS Code entfernt.

Updated heavily outdated README files for shp2pgsql/pgsql2shp by merging them with the manpages.

Fixed version tag in jdbc code that still said "1.1.3" in the "1.1.4" release.

A.40.3 Neue Funktionalität

Added -S option for non-multi geometries to shp2pgsql

A.41 Release 1.1.4

Freigabedatum: 2006/09/27

Dies ist eine Bugfix Release mit einigen Verbesserungen in der Java Schnittstelle. Ein Upgrade wird *empfohlen*.

A.41.1 Upgraden

Wenn Sie von Version 1.0.3 oder höher aus upgraden, folgen Sie bitte dem **soft upgrade** Prozedere.

If you are upgrading from a release *between 1.0.0RC6 and 1.0.2* (inclusive) and really want a live upgrade read the **upgrade section** of the 1.0.3 release notes chapter.

Ein Upgrade einer Vorgängerversion von 1.0.0RC6 benötigt ein **hard upgrade**.

A.41.2 Bugfixes

Unterstützung für PostgreSQL 8.2

Fixed bug in collect() function discarding SRID of input

SRID Prüfkriterien in MakeBox2d und MakeBox3d hinzugefügt

Regressionstests fixiert, damit GEOS-3.0.0 diesen auch bestehen kann

Verbesserung der Nebenläufigkeit von pgsq2shp.

A.41.3 Java Änderungen

reworked JTS support to reflect new upstream JTS developers' attitude to SRID handling. Simplifies code and drops build depend on GNU trove.

EJB2 Unterstützung wurde von "Geodetix s.r.l. Company" großzügig unterstützt

EJB3 Anleitung / Beispiele von Norman Barker <nbarker@ittvis.com>

Kleine Reorganisation des Java Verzeichnisses.

A.42 Release 1.1.3

Freigabedatum: 2006/06/30

This is an bugfix release including also some new functionalities (most notably long transaction support) and portability enhancements. Upgrade is *encouraged*.

A.42.1 Upgraden

Wenn Sie von Version 1.0.3 oder höher aus upgraden, folgen Sie bitte dem **soft upgrade** Prozedere.

If you are upgrading from a release *between 1.0.0RC6 and 1.0.2* (inclusive) and really want a live upgrade read the **upgrade section** of the 1.0.3 release notes chapter.

Ein Upgrade einer Vorgängerversion von 1.0.0RC6 benötigt ein **hard upgrade**.

A.42.2 Bugfixes / Fehlerfreiheit

Bugfix: distance(poly,poly) gab falsche Ergebnisse.

BUGFIX in pgsq2shp successful return code.

Bugfix in shp2pgsql bei der Handhabung von MultiLine-WKT.

BUGFIX in affine() failing to update bounding box.

WKT parser: forbidden construction of multigeometries with EMPTY elements (still supported for GEOMETRYCOLLECTION).

A.42.3 Neue Funktionalität

NEW Long Transactions support.

NEW DumpRings() function.

NEW AsHEXEWKB(geom, XDR|NDR) function.

A.42.4 JDBC Änderungen

Verbesserte Regressionstests: MultiPoint und wissenschaftliche Ordinaten

Einige kleine Bugs im JDBC Code bereinigt

Added proper accessor functions for all fields in preparation of making those fields private later

A.42.5 Sonstige Änderungen

NEU Regressionstest für Loader/Dumper

Optionen --with-proj-libdir und --with-geos-libdir hinzugefügt.

Kompilation für Tru64.

Jade zur Erzeugung der Dokumentation

pgsql2shp nur mit den notwendigen Bibliotheken binden

Initiale Unterstützung von PostgreSQL 8.2.

A.43 Release 1.1.2

Freigabedatum: 2006/03/30

This is an bugfix release including some new functions and portability enhancements. Upgrade is *encouraged*.

A.43.1 Upgraden

Wenn Sie von Version 1.0.3 oder höher aus upgraden, folgen Sie bitte dem **soft upgrade** Prozedere.

If you are upgrading from a release *between 1.0.0RC6 and 1.0.2* (inclusive) and really want a live upgrade read the **upgrade section** of the 1.0.3 release notes chapter.

Ein Upgrade einer Vorgängerversion von 1.0.0RC6 benötigt ein **hard upgrade**.

A.43.2 Bugfixes

Bugfix in der SnapToGrid() Berechnung des Umgebungsrechtecks

BUGFIX in EnforceRHR()

jdbc2 SRID handling fixes in JTS code

Unterstützung für 64bit Architekturen

A.43.3 Neue Funktionalität

Regressionstest können nun **vor** der PostGIS Installation ausgeführt werden

New affine() matrix transformation functions

Neue rotate{,X,Y,Z}() Funktion

Alte Übersetzungs- und Skalierungsfunktionen benutzen nun intern affine()

Embedded access control in estimated_extent() for builds against postgresql >= 8.0.0

A.43.4 Sonstige Änderungen

Portableres `./configure` Skript

Changed `./run_test` script to have more sane default behaviour

A.44 Release 1.1.1

Freigabedatum: 2006/01/23

This is an important Bugfix release, upgrade is *highly recommended*. Previous version contained a bug in `postgis_restore.pl` preventing **hard upgrade** procedure to complete and a bug in GEOS-2.2+ connector preventing GeometryCollection objects to be used in topological operations.

A.44.1 Upgraden

Wenn Sie von Version 1.0.3 oder höher aus upgraden, folgen Sie bitte dem **soft upgrade** Prozedere.

If you are upgrading from a release *between 1.0.0RC6 and 1.0.2* (inclusive) and really want a live upgrade read the **upgrade section** of the 1.0.3 release notes chapter.

Ein Upgrade einer Vorgängerversion von 1.0.0RC6 benötigt ein **hard upgrade**.

A.44.2 Bugfixes

Fixed a premature exit in `postgis_restore.pl`

Bugfix in der Handhabung der GeometryCollection durch den GEOS-CAPI Konnektor

Verbesserungen bei der Unterstützung von Solaris 2.7 und MingW

BUGFIX in `line_locate_point()`

Handhabung der PostgreSQL Dateipfade fixiert

BUGFIX in `line_substring()`

Added support for localized cluster in regress tester

A.44.3 Neue Funktionalität

Neue Z und M Interpolation in `line_substring()`

neue Z und M Interpolation in `line_interpolate_point()`

added `NumInteriorRing()` alias due to OpenGIS ambiguity

A.45 Release 1.1.0

Freigabedatum: 2005/12/21

This is a Minor release, containing many improvements and new things. Most notably: build procedure greatly simplified; transform() performance drastically improved; more stable GEOS connectivity (CAPI support); lots of new functions; draft topology support.

It is *highly recommended* that you upgrade to GEOS-2.2.x before installing PostGIS, this will ensure future GEOS upgrades won't require a rebuild of the PostGIS library.

A.45.1 Danksagungen

This release includes code from Mark Cave Ayland for caching of proj4 objects. Markus Schaber added many improvements in his JDBC2 code. Alex Bodnaru helped with PostgreSQL source dependency relief and provided Debian specfiles. Michael Fuhr tested new things on Solaris arch. David Techer and Gerald Fenoy helped testing GEOS C-API connector. Hartmut Tschauner provided code for the azimuth() function. Devrim GUNDUZ provided RPM specfiles. Carl Anderson helped with the new area building functions. See the [credits](#) section for more names.

A.45.2 Upgraden

If you are upgrading from release 1.0.3 or later you *DO NOT* need a dump/reload. Simply sourcing the new lwpostgis_upgrade.sql script in all your existing databases will work. See the [soft upgrade](#) chapter for more information.

If you are upgrading from a release *between 1.0.0RC6 and 1.0.2* (inclusive) and really want a live upgrade read the [upgrade section](#) of the 1.0.3 release notes chapter.

Ein Upgrade einer Vorgängerversion von 1.0.0RC6 benötigt ein [hard upgrade](#).

A.45.3 Neue Funktionen

scale() und transscale() als verwandte Methoden zu translate()

line_substring()

line_locate_point()

M(point)

LineMerge(geometry)

shift_longitude(geometry)

azimuth(geometry)

locate_along_measure(geometry, float8)

locate_between_measures(geometry, float8, float8)

SnapToGrid by point offset (up to 4d support)

BuildArea(any_geometry)

OGC BdPolyFromText(linestring_wkt, srid)

OGC BdMPolyFromText(linestring_wkt, srid)

RemovePoint(linestring, offset)

ReplacePoint(linestring, offset, point)

A.45.4 Bugfixes

Speicherleck in polygonize() fixiert

Bugfix der Funktionen zur Typumwandlung in lwgeom_as_anytype

Fixed USE_GEOS, USE_PROJ and USE_STATS elements of postgis_version() output to always reflect library state.

A.45.5 Semantische Funktionsänderungen

Höhere Dimensionen werden von SnapToGrid nicht mehr verworfen

Changed Z() function to return NULL if requested dimension is not available

A.45.6 Verbesserung der Rechenleistung

Much faster transform() function, caching proj4 objects

Removed automatic call to fix_geometry_columns() in AddGeometryColumns() and update_geometry_stats()

A.45.7 JDBC2 funktioniert

Optimierungen im Makefile

Unterstützung für JTS verbessert

Verbessertes System für Regressionstests

Basic consistency check method for geometry collections

Unterstützung von (Hex)(E)wkb

Autoprobing DriverWrapper for HexWKB / EWKT switching

fix compile problems in ValueSetter for ancient jdk releases.

fix EWKT constructors to accept SRID=4711; representation

added preliminary read-only support for java2d geometries

A.45.8 Sonstige Neuigkeiten

Full autoconf-based configuration, with PostgreSQL source dependency relief

GEOS C-API Unterstützung (2.2.0 und höher)

Erstunterstützung für Topologie

Debian and RPM specfiles

Neues lwpostgis_upgrade.sql Skript

A.45.9 Sonstige Änderungen

Unterstützung für JTS verbessert

Stricter mapping between DBF and SQL integer and string attributes

Wider and cleaner regression test suite

Alter JDBC Code aus der Release entfernt

obsoleted direct use of postgis_proc_upgrade.pl

scripts version unified with release version

A.46 Release 1.0.6

Freigabedatum: 2005/12/06

Enthält einige Bugfixes und Verbesserungen.

A.46.1 Upgraden

If you are upgrading from release 1.0.3 or later you *DO NOT* need a dump/reload.

If you are upgrading from a release *between 1.0.0RC6 and 1.0.2* (inclusive) and really want a live upgrade read the [upgrade section](#) of the 1.0.3 release notes chapter.

Ein Upgrade einer Vorgängerversion von 1.0.0RC6 benötigt ein [hard upgrade](#).

A.46.2 Bugfixes

Fixed palloc(0) call in collection deserializer (only gives problem with --enable-cassert)

Fixed bbox cache handling bugs

Schutzverletzung in geom_accum(NULL, NULL) fixiert

Schutzverletzung in addPoint() fixiert

Fixed short-allocation in lwcollection_clone()

Bugfix in segmentize()

Fixed bbox computation of SnapToGrid output

A.46.3 Verbesserungen

Initiale Unterstützung von PostgreSQL 8.2

Added missing SRID mismatch checks in GEOS ops

A.47 Release 1.0.5

Freigabedatum: 2005/11/25

Contains memory-alignment fixes in the library, a segfault fix in loader's handling of UTF8 attributes and a few improvements and cleanups.



Note

Return code of shp2pgsql changed from previous releases to conform to unix standards (return 0 on success).

A.47.1 Upgraden

If you are upgrading from release 1.0.3 or later you *DO NOT* need a dump/reload.

If you are upgrading from a release *between 1.0.0RC6 and 1.0.2* (inclusive) and really want a live upgrade read the [upgrade section](#) of the 1.0.3 release notes chapter.

Ein Upgrade einer Vorgängerversion von 1.0.0RC6 benötigt ein [hard upgrade](#).

A.47.2 Bibliotheksänderungen

Fixed memory alignment problems

Fixed computation of null values fraction in analyzer

Fixed a small bug in the `getPoint4d_p()` low-level function

Speedup of serializer functions

Bugfix in `force_3dm()`, `force_3dz()` und `force_4d()`

A.47.3 Änderungen beim Loader

Fixed return code of `shp2pgsql`

Fixed back-compatibility issue in loader (load of null shapefiles)

Fixed handling of trailing dots in dbf numerical attributes

Bugfix Schutzverletzung in `shp2pgsql` (UTF8 Zeichenkodierung)

A.47.4 Sonstige Änderungen

Schema aware `postgis_proc_upgrade.pl`, support for `pgsql 7.2+`

New "Reporting Bugs" chapter in manual

A.48 Release 1.0.4

Freigabedatum: 2005/09/09

Contains important bug fixes and a few improvements. In particular, it fixes a memory leak preventing successful build of GiST indexes for large spatial tables.

A.48.1 Upgraden

Falls Sie die Version 1.0.3 upgraden benötigen Sie *KEIN* Dump/Reload.

If you are upgrading from a release *between 1.0.0RC6 and 1.0.2* (inclusive) and really want a live upgrade read the **upgrade section** of the 1.0.3 release notes chapter.

Ein Upgrade einer Vorgängerversion von 1.0.0RC6 benötigt ein **hard upgrade**.

A.48.2 Bugfixes

Memory leak plugged in GiST indexing

Segfault fix in `transform()` handling of proj4 errors

Fixed some proj4 texts in `spatial_ref_sys` (missing `+proj`)

Loader: fixed string functions usage, reworked NULL objects check, fixed segfault on MULTILINESTRING input.

Fixed bug in `MakeLine` dimension handling

Bugfix in `translate()`, beschädigte das Umgebungsrechteck

A.48.3 Verbesserungen

Verbesserung der Dokumentation

More robust selectivity estimator

Kleinere Geschwindigkeitsverbesserung in distance()

Kleinere Bereinigungen

Bereinigung der GiST Indizierung

Looser syntax acceptance in box3d parser

A.49 Release 1.0.3

Freigabedatum: 2005/08/08

Contains some bug fixes - *including a severe one affecting correctness of stored geometries* - and a few improvements.

A.49.1 Upgraden

Due to a bug in a bounding box computation routine, the upgrade procedure requires special attention, as bounding boxes cached in the database could be incorrect.

An **hard upgrade** procedure (dump/reload) will force recomputation of all bounding boxes (not included in dumps). This is *required* if upgrading from releases prior to 1.0.0RC6.

If you are upgrading from versions 1.0.0RC6 or up, this release includes a perl script (utils/rebuild_bbox_caches.pl) to force recomputation of geometries' bounding boxes and invoke all operations required to propagate eventual changes in them (geometry statistics update, reindexing). Invoke the script after a make install (run with no args for syntax help). Optionally run utils/postgis_proc_upgrade.pl to refresh postgis procedures and functions signatures (see **Soft upgrade**).

A.49.2 Bugfixes

Heftiger Bugfix in lwgeom's Berechnung des 2D Umgebungsrechteck

Bugfix in WKT (-w) POINT handling in loader

Bugfix im Dumper für 64bit-Architektur

Bugfix in dumper handling of user-defined queries

Bugfix im create_undef.pl Skript

A.49.3 Verbesserungen

Small performance improvement in canonical input function

Kleinere Bereinigungen im Loader

Support for multibyte field names in loader

Improvement in the postgis_restore.pl script

New rebuild_bbox_caches.pl util script

A.50 Release 1.0.2

Freigabedatum: 2005/07/04

Enthält einige Bugfixes und Verbesserungen.

A.50.1 Upgraden

If you are upgrading from release 1.0.0RC6 or up you *DO NOT* need a dump/reload.

Ein Upgrade von älteren Versionen benötigt ein Dump/Reload. Für weiterführende Information siehe [Upgrading](#)

A.50.2 Bugfixes

Fehlertolerante B-Baum Operatoren

Memory leak plugged in pg_error

Fixierung des R-Baum Index

Bereinigung der Kompilationsskripts (vermeiden der Mischung aus CFLAGS und CXXFLAGS)

A.50.3 Verbesserungen

Neue Möglichkeiten zur Erzeugung von Indizes im Loader (-I Option)

Erstunterstützung von PostgreSQL 8.1dev

A.51 Release 1.0.1

Freigabedatum: 2005/05/24

Enthält ein paar Bugfixes und einige Verbesserungen.

A.51.1 Upgraden

If you are upgrading from release 1.0.0RC6 or up you *DO NOT* need a dump/reload.

Ein Upgrade von älteren Versionen benötigt ein Dump/Reload. Für weiterführende Information siehe [Upgrading](#)

A.51.2 Bibliotheksänderungen

Bugfix in der 3D-Berechnung von length_spheroid()

BUGFIX in join selectivity estimator

A.51.3 Sonstige Änderungen/Ergänzungen

BUGFIX in shp2pgsql escape functions

better support for concurrent postgis in multiple schemas

documentation fixes

jdbc2: compile with "-target 1.2 -source 1.2" by default

NEW -k Switch für pgsql2shp

NEW support for custom createdb options in postgis_restore.pl

BUGFIX in pgsql2shp attribute names unicity enforcement

Bugfix in der Projektionsdefinition für Paris

postgis_restore.pl Codebereinigung

A.52 Release 1.0.0

Freigabedatum: 2005/04/19

Final 1.0.0 release. Contains a few bug fixes, some improvements in the loader (most notably support for older postgis versions), and more docs.

A.52.1 Upgraden

If you are upgrading from release 1.0.0RC6 you *DO NOT* need a dump/reload.

Upgrading from any other precedent release requires a dump/reload. See the [upgrading](#) chapter for more informations.

A.52.2 Bibliotheksänderungen

BUGFIX in transform() releasing random memory address

BUGFIX in force_3dm() allocating less memory then required

BUGFIX in join selectivity estimator (defaults, leaks, tuplecount, sd)

A.52.3 Sonstige Änderungen/Ergänzungen

BUGFIX in shp2pgsql escape of values starting with tab or single-quote

NEU Kapitel für Loader/Dumper

NEU shp2pgsql Unterstützung für alte (HWGEOM) PostGIS Versionen

NEU -p (prepare) Flag für shp2pgsql

NEU Kapitel über OGC Konformität

NEU autoconf Unterstützung für die JTS Bibliothek

BUGFIX in estimator testers (support for LWGEOM and schema parsing)

A.53 Release 1.0.0RC6

Freigabedatum: 2005/03/30

Sechster Release Kandidat für 1.0.0. Enthält ein paar Bugfixes und Codebereinigungen.

A.53.1 Upgraden

You need a dump/reload to upgrade from precedent releases. See the [upgrading](#) chapter for more informations.

A.53.2 Bibliotheksänderungen

BUGFIX in multi()

early return [when noop] from multi()

A.53.3 Skriptänderungen

{x,y}{min,max}(box2d) Funktionen gelöscht

A.53.4 Sonstige Änderungen

BUGFIX in postgis_restore.pl Skript

BUGFIX in dumper's 64bit support

A.54 Release 1.0.0RC5

Freigabedatum: 2005/03/25

Fünfter Release Kandidat für 1.0.0. Enthält ein paar Bugfixes und eine Verbesserung.

A.54.1 Upgraden

If you are upgrading from release 1.0.0RC4 you *DO NOT* need a dump/reload.

Upgrading from any other precedent release requires a dump/reload. See the [upgrading](#) chapter for more informations.

A.54.2 Bibliotheksänderungen

BUGFIX (Schutzverletzung) in der Box3D Berechnung ("yes, another!").

BUGFIX (segfaulting) in estimated_extent().

A.54.3 Sonstige Änderungen

Small build scripts and utilities refinements.

Additional performance tips documented.

A.55 Release 1.0.0RC4

Freigabedatum: 2005/03/18

Vierter Release Kandidat für 1.0.0. Enthält Bugfixes und einige Verbesserungen.

A.55.1 Upgraden

You need a dump/reload to upgrade from precedent releases. See the [upgrading](#) chapter for more informations.

A.55.2 Bibliotheksänderungen

BUGFIX (segfaulting) in geom_accum().

BUGFIX für 64bit-Architekturen.

BUGFIX in box3d computation function with collections.

NEW subselects support in selectivity estimator.

Early return from force_collection.

Consistency check fix in SnapToGrid().

Box2d output changed back to 15 significant digits.

A.55.3 Skriptänderungen

NEUE distance_sphere() Funktion.

"get_proj4_from_srid" geändert, sodass jetzt PL/PGSQL anstatt SQL verwendet wird.

A.55.4 Sonstige Änderungen

BUGFIX in loader and dumper handling of MultiLine shapes

BUGFIX in loader, skipping all but first hole of polygons.

jdbc2: Codebereinigung, Verbesserung des Makefile

FLEX and YACC variables set `*after*` postgres Makefile.global is included and only if the postgres `*stripped*` version evaluates to the empty string

Added already generated parser in release

Verfeinerung des Kompilationsskripts

improved version handling, central Version.config

improvements in postgres_restore.pl

A.56 Release 1.0.0RC3

Freigabedatum: 2005/02/24

Dritter Release Kandidat für 1.0.0. Enthält viele Bugfixes und Verbesserungen.

A.56.1 Upgraden

You need a dump/reload to upgrade from precedent releases. See the [upgrading](#) chapter for more informations.

A.56.2 Bibliotheksänderungen

BUGFIX in transform(): missing SRID, better error handling.

BUGFIX in memory alignment handling

BUGFIX in force_collection() causing mapserver connector failures on simple (single) geometry types.

BUGFIX in GeometryFromText() missing to add a bbox cache.

reduced precision of box2d output.

prefixed DEBUG macros with PGIS_ to avoid clash with postgres one

plugged a leak in GEOS2POSTGIS converter

Reduced memory usage by early releasing query-context pallocated one.

A.56.3 Skriptänderungen

BUGFIX in 72 index bindings.

BUGFIX in probe_geometry_columns() to work with PG72 and support multiple geometry columns in a single table

NEU bool::text Typumwandlung

Zur Steigerung der Rechenleistung wurden einige Funktionen von STABLE auf IMMUTABLE geändert.

A.56.4 JDBC Änderungen

jdbc2: small patches, box2d/3d tests, revised docs and license.

jdbc2: bug fix and testcase in for pgjdbc 8.0 type autoregistration

jdbc2: Removed use of jdk1.4 only features to enable build with older jdk releases.

jdbc2: Added support for building against pg72jdbc2.jar

jdbc2: updated and cleaned makefile

jdbc2: added BETA support for jts geometry classes

jdbc2: Skip known-to-fail tests against older PostGIS servers.

jdbc2: Fixed handling of measured geometries in EWKT.

A.56.5 Sonstige Änderungen

new performance tips chapter in manual

documentation updates: pgsql72 requirement, lwpostgis.sql

Einige Änderungen in autoconf

BUILDDATE extraction made more portable

spatial_ref_sys.sql fixiert, damit das Vacuum nicht auf die gesamte Datenbank ausgeführt wird.

spatial_ref_sys: changed Paris entries to match the ones distributed with 0.x.

A.57 Release 1.0.0RC2

Freigabedatum: 2005/01/26

Second release candidate for 1.0.0 containing bug fixes and a few improvements.

A.57.1 Upgraden

You need a dump/reload to upgrade from precedent releases. See the [upgrading](#) chapter for more informations.

A.57.2 Bibliotheksänderungen

BUGFIX in der Berechnung von Box3D für PointArray

Bugfix in der Definition von distance_spheroid

Bugfix in transform() : fehlendes Update des BBox Zwischenspeichers

NEUER jdbc driver (jdbc2)

GEOMETRYCOLLECTION(EMPTY) Syntax für Rückwärtskompatibilität

Schnellere Binärausgabe

Striktere OGC WKB/WKT Konstruktoren

A.57.3 Skriptänderungen

Genauere Verwendung von STABLE, IMMUTABLE, STRICT in lwpostgis.sql

Striktere OGC WKB/WKT Konstruktoren

A.57.4 Sonstige Änderungen

Schnellerer und stabilerer LaderFaster (i18n und nicht)

Erstes autoconf Skript

A.58 Release 1.0.0RC1

Freigabedatum: 2005/01/13

This is the first candidate of a major postgis release, with internal storage of postgis types redesigned to be smaller and faster on indexed queries.

A.58.1 Upgraden

You need a dump/reload to upgrade from precedent releases. See the [upgrading](#) chapter for more informations.

A.58.2 Änderungen

Faster canonical input parsing.

Lossless canonical output.

EWKB Canonical binary IO with PG>73.

Unterstützung von 4D-Koordinaten, verlustfreie shapefile->postgis->shapefile Konvertierung.

Neue Funktionen: UpdateGeometrySRID(), AsGML(), SnapToGrid(), ForceRHR(), estimated_extent(), accum().

Vertical positioning indexed operators.

JOIN selectivity function.

More geometry constructors / editors.

PostGIS Extension API.

UTF8 Unterstützung im Loader.