

# **PostGIS 3.6.0dev Handbuch**

---

**DEV (Wed 06 Nov 2024 05:30:35 PM UTC rev. 6366bd6 )**

# Contents

<b>1</b>	<b>Einführung</b>	<b>1</b>
1.1	Projektleitung . . . . .	1
1.2	Aktuelle Kernentwickler . . . . .	2
1.3	Frühere Kernentwickler . . . . .	2
1.4	Weitere Mitwirkende . . . . .	3
<b>2</b>	<b>PostGIS Installation</b>	<b>6</b>
2.1	Kurzfassung . . . . .	6
2.2	Kompilierung und Installation des Quellcodes: Detaillierte Beschreibung . . . . .	6
2.2.1	Nutzung des Quellcodes . . . . .	7
2.2.2	Systemvoraussetzungen . . . . .	7
2.2.3	Konfiguration . . . . .	8
2.2.4	Build-Prozess . . . . .	10
2.2.5	Build-Prozess für die PostGIS Extensions und deren Bereitstellung . . . . .	10
2.2.6	Softwaretest . . . . .	12
2.2.7	Installation . . . . .	15
2.3	Installation und Verwendung des Adressennormierers . . . . .	16
2.4	Installieren, Aktualisieren von Tiger Geocoder und Laden von Daten . . . . .	16
2.4.1	Tiger Geocoder Aktivieren Sie Ihre PostGIS-Datenbank . . . . .	17
2.4.2	Die Adressennormierer-Extension zusammen mit dem Tiger Geocoder verwenden . . . . .	19
2.4.3	Erforderliche Werkzeuge für das Laden von Tigerdaten . . . . .	19
2.4.4	Aktualisieren der Tiger Geocoder Installation und Daten . . . . .	20
2.5	Übliche Probleme bei der Installation . . . . .	21
<b>3</b>	<b>PostGIS Verwaltung</b>	<b>22</b>
3.1	Leistungsoptimierung . . . . .	22
3.1.1	Startup . . . . .	22
3.1.2	Laufzeit . . . . .	23
3.2	Konfigurieren der Rasterunterstützung . . . . .	23
3.3	Erstellung räumlicher Datenbanken . . . . .	24
3.3.1	Die spatiale Datenbankerweiterung mittels EXTENSION aktivieren . . . . .	24

---

3.3.2	Datenbank räumlich aktivieren, ohne EXTENSION zu verwenden (nicht empfohlen)	24
3.4	Aktualisierung von Geodatenbanken	25
3.4.1	Sanftes Upgrade	25
3.4.1.1	Soft Upgrade 9.1+ mit Erweiterungen	25
3.4.1.2	Soft Upgrade Pre 9.1+ oder ohne Erweiterungen	26
3.4.2	Hartes Upgrade	27
<b>4</b>	<b>Datenverwaltung</b>	<b>29</b>
4.1	Räumliches Datenmodell	29
4.1.1	OGC-Geometrie	29
4.1.1.1	Punkt	30
4.1.1.2	Linie	30
4.1.1.3	LinearRing	30
4.1.1.4	Polygon	30
4.1.1.5	MultiPoint	30
4.1.1.6	MultiLineString	30
4.1.1.7	MultiPolygon	31
4.1.1.8	GeometryCollection	31
4.1.1.9	PolyhedralSurface	31
4.1.1.10	Dreieck	31
4.1.1.11	TIN	31
4.1.2	SQL/MM Teil 3 - Kurven	31
4.1.2.1	CircularString	32
4.1.2.2	CompoundCurve	32
4.1.2.3	KurvenPolygon	32
4.1.2.4	MultiCurve	32
4.1.2.5	MultiSurface	32
4.1.3	WKT und WKB	33
4.2	Geometrie Datentyp	34
4.2.1	PostGIS EWKB und EWKT	34
4.3	Geographie Datentyp	36
4.3.1	Erstellen von Geografietabellen	36
4.3.2	Verwendung von Geografietabellen	37
4.3.3	Wann wird der Datentyp Geografie verwendet?	38
4.3.4	Fortgeschrittene FAQ's zum geographischen Datentyp	39
4.4	Geometrivalidierung	39
4.4.1	Einfache Geometrie	39
4.4.2	Gültige Geometrie	41
4.4.3	Verwaltung der Gültigkeit	43

---

4.5	Räumliche Bezugssysteme . . . . .	44
4.5.1	SPATIAL_REF_SYS Tabelle . . . . .	45
4.5.2	Benutzerdefinierte räumliche Bezugssysteme . . . . .	46
4.6	Räumliche Tabellen . . . . .	46
4.6.1	Erstellung einer räumlichen Tabelle . . . . .	46
4.6.2	GEOMETRY_COLUMNS Ansicht . . . . .	47
4.6.3	Manuelles Registrieren von Geometriespalten . . . . .	48
4.7	Laden von Geodaten . . . . .	50
4.7.1	SQL zum Laden von Daten verwenden . . . . .	50
4.7.2	Verwendung des Shapefile Loaders . . . . .	50
4.8	Extrahieren von Geodaten . . . . .	52
4.8.1	SQL zum Extrahieren von Daten verwenden . . . . .	52
4.8.2	Verwendung des Shapefile-Dumpers . . . . .	53
4.9	Räumliche Indizes . . . . .	54
4.9.1	GiST-Indizes . . . . .	54
4.9.2	BRIN Indizes . . . . .	55
4.9.3	SP-GiST Indizes . . . . .	57
4.9.4	Abstimmung der Indexverwendung . . . . .	57
<b>5</b>	<b>Räumliche Abfrage</b>	<b>59</b>
5.1	Räumliche Beziehungen feststellen . . . . .	59
5.1.1	Dimensionell erweitertes 9-Schnitte-Modell . . . . .	59
5.1.2	Benannte räumliche Beziehungen . . . . .	61
5.1.3	Allgemeine räumliche Beziehungen . . . . .	62
5.2	Räumliche Indizes verwenden . . . . .	64
5.3	Beispiele für Spatial SQL . . . . .	64
<b>6</b>	<b>Performance Tipps</b>	<b>68</b>
6.1	Kleine Tabellen mit großen Geometrien . . . . .	68
6.1.1	Problembeschreibung . . . . .	68
6.1.2	Umgehungslösung . . . . .	68
6.2	CLUSTER auf die geometrischen Indizes . . . . .	69
6.3	Vermeidung von Dimensionsumrechnungen . . . . .	69
<b>7</b>	<b>Referenz PostGIS</b>	<b>70</b>
7.1	PostgreSQL und PostGIS Datentypen - Geometry/Geography/Box . . . . .	70
7.1.1	box2d . . . . .	70
7.1.2	box3d . . . . .	71
7.1.3	geometry . . . . .	71
7.1.4	geometry_dump . . . . .	72

---

7.1.5	geography . . . . .	72
7.2	Geometrische Managementfunktionen . . . . .	73
7.2.1	AddGeometryColumn . . . . .	73
7.2.2	DropGeometryColumn . . . . .	75
7.2.3	DropGeometryTable . . . . .	75
7.2.4	Find_SRID . . . . .	76
7.2.5	Populate_Geometry_Columns . . . . .	77
7.2.6	UpdateGeometrySRID . . . . .	78
7.3	Geometrische Konstruktoren . . . . .	79
7.3.1	ST_Collect . . . . .	79
7.3.2	ST_LineFromMultiPoint . . . . .	81
7.3.3	ST_MakeEnvelope . . . . .	82
7.3.4	ST_MakeLine . . . . .	82
7.3.5	ST_MakePoint . . . . .	84
7.3.6	ST_MakePointM . . . . .	85
7.3.7	ST_MakePolygon . . . . .	86
7.3.8	ST_Point . . . . .	88
7.3.9	ST_PointZ . . . . .	89
7.3.10	ST_PointM . . . . .	90
7.3.11	ST_PointZM . . . . .	90
7.3.12	ST_Polygon . . . . .	91
7.3.13	ST_TileEnvelope . . . . .	92
7.3.14	ST_HexagonGrid . . . . .	93
7.3.15	ST_Hexagon . . . . .	96
7.3.16	ST_SquareGrid . . . . .	97
7.3.17	ST_Square . . . . .	98
7.3.18	ST_Letters . . . . .	99
7.4	Geometrische Zugriffsfunktionen . . . . .	100
7.4.1	GeometryType . . . . .	100
7.4.2	ST_Boundary . . . . .	101
7.4.3	ST_BoundingDiagonal . . . . .	104
7.4.4	ST_CoordDim . . . . .	105
7.4.5	ST_Dimension . . . . .	105
7.4.6	ST_Dump . . . . .	106
7.4.7	ST_DumpPoints . . . . .	108
7.4.8	ST_DumpSegments . . . . .	112
7.4.9	ST_DumpRings . . . . .	114
7.4.10	ST_EndPoint . . . . .	115
7.4.11	ST_Envelope . . . . .	116

---

---

7.4.12	ST_ExteriorRing	117
7.4.13	ST_GeometryN	118
7.4.14	ST_GeometryType	120
7.4.15	ST_HasArc	122
7.4.16	ST_InteriorRingN	122
7.4.17	ST_NumCurves	123
7.4.18	ST_CurveN	123
7.4.19	ST_IsClosed	124
7.4.20	ST_IsCollection	126
7.4.21	ST_IsEmpty	127
7.4.22	ST_IsPolygonCCW	128
7.4.23	ST_IsPolygonCW	129
7.4.24	ST_IsRing	129
7.4.25	ST_IsSimple	130
7.4.26	ST_M	131
7.4.27	ST_MemSize	132
7.4.28	ST_NDims	133
7.4.29	ST_NPoints	134
7.4.30	ST_NRings	134
7.4.31	ST_NumGeometries	135
7.4.32	ST_NumInteriorRings	136
7.4.33	ST_NumInteriorRing	137
7.4.34	ST_NumPatches	137
7.4.35	ST_NumPoints	138
7.4.36	ST_PatchN	138
7.4.37	ST_PointN	139
7.4.38	ST_Points	141
7.4.39	ST_StartPoint	141
7.4.40	ST_Summary	143
7.4.41	ST_X	144
7.4.42	ST_Y	144
7.4.43	ST_Z	145
7.4.44	ST_Zmflag	146
7.4.45	ST_HasZ	147
7.4.46	ST_HasM	148
7.5	Geometrische Editoren	148
7.5.1	ST_AddPoint	148
7.5.2	ST_CollectionExtract	149
7.5.3	ST_CollectionHomogenize	150

---

7.5.4	ST_CurveToLine	152
7.5.5	ST_Scroll	154
7.5.6	ST_FlipCoordinates	155
7.5.7	ST_Force2D	156
7.5.8	ST_Force3D	156
7.5.9	ST_Force3DZ	157
7.5.10	ST_Force3DM	158
7.5.11	ST_Force4D	159
7.5.12	ST_ForceCollection	160
7.5.13	ST_ForceCurve	161
7.5.14	ST_ForcePolygonCCW	161
7.5.15	ST_ForcePolygonCW	162
7.5.16	ST_ForceSFS	162
7.5.17	ST_ForceRHR	163
7.5.18	ST_LineExtend	164
7.5.19	ST_LineToCurve	164
7.5.20	ST_Multi	166
7.5.21	ST_Normalize	166
7.5.22	ST_Project	167
7.5.23	ST_QuantizeCoordinates	168
7.5.24	ST_RemovePoint	170
7.5.25	ST_RemoveRepeatedPoints	170
7.5.26	ST_RemoveIrrelevantPointsForView	171
7.5.27	ST_RemoveSmallParts	174
7.5.28	ST_Reverse	175
7.5.29	ST_Segmentize	176
7.5.30	ST_SetPoint	178
7.5.31	ST_ShiftLongitude	179
7.5.32	ST_WrapX	180
7.5.33	ST_SnapToGrid	181
7.5.34	ST_Snap	182
7.5.35	ST_SwapOrdinates	185
7.6	Geometrievalidierung	186
7.6.1	ST_IsValid	186
7.6.2	ST_IsValidDetail	187
7.6.3	ST_IsValidReason	189
7.6.4	ST_MakeValid	190
7.7	Funktionen des räumlichen Bezugssystems	195
7.7.1	ST_InverseTransformPipeline	195

---

7.7.2	ST_SetSRID	196
7.7.3	ST_SRID	197
7.7.4	ST_Transform	198
7.7.5	ST_TransformPipeline	200
7.7.6	postgis_srs_codes	202
7.7.7	postgis_srs	202
7.7.8	postgis_srs_all	203
7.7.9	postgis_srs_search	204
7.8	Geometrische Konstruktoren	205
7.8.1	Well-known-Text (WKT) Repräsentation	205
7.8.1.1	ST_BdPolyFromText	205
7.8.1.2	ST_BdMPolyFromText	205
7.8.1.3	ST_GeogFromText	206
7.8.1.4	ST_GeographyFromText	206
7.8.1.5	ST_GeomCollFromText	207
7.8.1.6	ST_GeomFromEWKT	207
7.8.1.7	ST_GeomFromMARC21	209
7.8.1.8	ST_GeometryFromText	211
7.8.1.9	ST_GeomFromText	212
7.8.1.10	ST_LineFromText	213
7.8.1.11	ST_MLineFromText	214
7.8.1.12	ST_MPointFromText	215
7.8.1.13	ST_MPolyFromText	216
7.8.1.14	ST_PointFromText	216
7.8.1.15	ST_PolygonFromText	217
7.8.1.16	ST_WKTTToSQL	218
7.8.2	Well-known-Binary (WKB) Repräsentation	219
7.8.2.1	ST_GeogFromWKB	219
7.8.2.2	ST_GeomFromEWKB	219
7.8.2.3	ST_GeomFromWKB	221
7.8.2.4	ST_LineFromWKB	222
7.8.2.5	ST_LinestringFromWKB	222
7.8.2.6	ST_PointFromWKB	223
7.8.2.7	ST_WKBTToSQL	224
7.8.3	Weitere Formate	225
7.8.3.1	ST_Box2dFromGeoHash	225
7.8.3.2	ST_GeomFromGeoHash	225
7.8.3.3	ST_GeomFromGML	226
7.8.3.4	ST_GeomFromGeoJSON	229

---



7.8.3.5	ST_GeomFromKML	230
7.8.3.6	ST_GeomFromTWKB	231
7.8.3.7	ST_GMLToSQL	231
7.8.3.8	ST_LineFromEncodedPolyline	232
7.8.3.9	ST_PointFromGeoHash	233
7.8.3.10	ST_FromFlatGeobufToTable	233
7.8.3.11	ST_FromFlatGeobuf	234
7.9	Geometrieausgabe	234
7.9.1	Well-known-Text (WKT) Repräsentation	234
7.9.1.1	ST_AsEWKT	234
7.9.1.2	ST_AsText	235
7.9.2	Well-known-Binary (WKB) Repräsentation	237
7.9.2.1	ST_AsBinary	237
7.9.2.2	ST_AsEWKB	238
7.9.2.3	ST_AsHEXEWKB	239
7.9.3	Weitere Formate	240
7.9.3.1	ST_AsEncodedPolyline	240
7.9.3.2	ST_AsFlatGeobuf	241
7.9.3.3	ST_AsGeobuf	241
7.9.3.4	ST_AsGeoJSON	242
7.9.3.5	ST_AsGML	244
7.9.3.6	ST_AsKML	248
7.9.3.7	ST_AsLatLonText	249
7.9.3.8	ST_AsMARC21	250
7.9.3.9	ST_AsMVTGeom	253
7.9.3.10	ST_AsMVT	254
7.9.3.11	ST_AsSVG	255
7.9.3.12	ST_AsTWKB	256
7.9.3.13	ST_AsX3D	257
7.9.3.14	ST_GeoHash	261
7.10	Operatoren	262
7.10.1	Bounding-Box-Operatoren	262
7.10.1.1	&&	262
7.10.1.2	&&(geometry,box2df)	263
7.10.1.3	&&(box2df,geometry)	264
7.10.1.4	&&(box2df,box2df)	265
7.10.1.5	&&&	265
7.10.1.6	&&&(geometry,gidx)	267
7.10.1.7	&&&(gidx,geometry)	267

---

7.10.1.8	&&&(gidx,gidx)	268
7.10.1.9	&<	269
7.10.1.10	&<	270
7.10.1.11	&>	271
7.10.1.12	<<	271
7.10.1.13	<<	272
7.10.1.14	=	273
7.10.1.15	>>	274
7.10.1.16	@	275
7.10.1.17	@(geometry,box2df)	276
7.10.1.18	@(box2df,geometry)	277
7.10.1.19	@(box2df,box2df)	278
7.10.1.20	&>	278
7.10.1.21	>>	279
7.10.1.22	~	280
7.10.1.23	~(geometry,box2df)	281
7.10.1.24	~(box2df,geometry)	281
7.10.1.25	~(box2df,box2df)	282
7.10.1.26	~=	283
7.10.2	Operatoren	284
7.10.2.1	<->	284
7.10.2.2	=	286
7.10.2.3	<#>	287
7.10.2.4	<<->>	288
7.11	Lagevergleiche	289
7.11.1	Topologische Beziehungen	289
7.11.1.1	ST_3DIntersects	289
7.11.1.2	ST_Contains	290
7.11.1.3	ST_ContainsProperly	293
7.11.1.4	ST_CoveredBy	295
7.11.1.5	ST_Covers	296
7.11.1.6	ST_Crosses	297
7.11.1.7	ST_Disjoint	299
7.11.1.8	ST_Equals	300
7.11.1.9	ST_Intersects	301
7.11.1.10	ST_LineCrossingDirection	303
7.11.1.11	ST_OrderingEquals	306
7.11.1.12	ST_Overlaps	307
7.11.1.13	ST_Relate	310

---

7.11.1.14	ST_RelateMatch	312
7.11.1.15	ST_Touches	313
7.11.1.16	ST_Within	315
7.11.2	Fernbeziehungen	317
7.11.2.1	ST_3DDWithin	317
7.11.2.2	ST_3DDFullyWithin	318
7.11.2.3	ST_DFullyWithin	319
7.11.2.4	ST_DWithin	320
7.11.2.5	ST_PointInsideCircle	321
7.12	Messfunktionen	322
7.12.1	ST_Area	322
7.12.2	ST_Azimuth	324
7.12.3	ST_Angle	325
7.12.4	ST_ClosestPoint	326
7.12.5	ST_3DClosestPoint	328
7.12.6	ST_Distance	329
7.12.7	ST_3DDistance	331
7.12.8	ST_DistanceSphere	332
7.12.9	ST_DistanceSpheroid	333
7.12.10	ST_FrechetDistance	334
7.12.11	ST_HausdorffDistance	335
7.12.12	ST_Length	336
7.12.13	ST_Length2D	338
7.12.14	ST_3DLength	338
7.12.15	ST_LengthSpheroid	339
7.12.16	ST_LongestLine	340
7.12.17	ST_3DLongestLine	342
7.12.18	ST_MaxDistance	343
7.12.19	ST_3DMaxDistance	344
7.12.20	ST_MinimumClearance	345
7.12.21	ST_MinimumClearanceLine	345
7.12.22	ST_Perimeter	346
7.12.23	ST_Perimeter2D	348
7.12.24	ST_3DPerimeter	348
7.12.25	ST_ShortestLine	349
7.12.26	ST_3DShortestLine	350
7.13	Overlay-Funktionen	352
7.13.1	ST_ClipByBox2D	352
7.13.2	ST_Difference	352

---

7.13.3	ST_Intersection	354
7.13.4	ST_MemUnion	356
7.13.5	ST_Node	357
7.13.6	ST_Split	357
7.13.7	ST_Subdivide	360
7.13.8	ST_SymDifference	362
7.13.9	ST_UnaryUnion	364
7.13.10	ST_Union	364
7.14	Geometrieverarbeitung	367
7.14.1	ST_Buffer	367
7.14.2	ST_BuildArea	372
7.14.3	ST_Centroid	373
7.14.4	ST_ChaikinSmoothing	375
7.14.5	ST_ConcaveHull	377
7.14.6	ST_ConvexHull	380
7.14.7	ST_DelaunayTriangles	382
7.14.8	ST_FilterByM	387
7.14.9	ST_GeneratePoints	388
7.14.10	ST_GeometricMedian	389
7.14.11	ST_LineMerge	390
7.14.12	ST_MaximumInscribedCircle	393
7.14.13	ST_LargestEmptyCircle	395
7.14.14	ST_MinimumBoundingCircle	397
7.14.15	ST_MinimumBoundingRadius	398
7.14.16	ST_OrientedEnvelope	399
7.14.17	ST_OffsetCurve	400
7.14.18	ST_PointOnSurface	403
7.14.19	ST_Polygonize	405
7.14.20	ST_ReducePrecision	407
7.14.21	ST_SharedPaths	409
7.14.22	ST_Simplify	411
7.14.23	ST_SimplifyPreserveTopology	412
7.14.24	ST_SimplifyPolygonHull	414
7.14.25	ST_SimplifyVW	417
7.14.26	ST_SetEffectiveArea	418
7.14.27	ST_TriangulatePolygon	420
7.14.28	ST_VoronoiLines	422
7.14.29	ST_VoronoiPolygons	423
7.15	Deckungen	425

---

7.15.1	ST_CoverageInvalidEdges	425
7.15.2	ST_CoverageSimplify	426
7.15.3	ST_CoverageUnion	428
7.16	Affine Transformationen	429
7.16.1	ST_Affine	429
7.16.2	ST_Rotate	431
7.16.3	ST_RotateX	432
7.16.4	ST_RotateY	433
7.16.5	ST_RotateZ	434
7.16.6	ST_Scale	435
7.16.7	ST_Translate	436
7.16.8	ST_TransScale	437
7.17	Clustering-Funktionen	438
7.17.1	ST_ClusterDBSCAN	438
7.17.2	ST_ClusterIntersecting	441
7.17.3	ST_ClusterIntersectingWin	441
7.17.4	ST_ClusterKMeans	442
7.17.5	ST_ClusterWithin	444
7.17.6	ST_ClusterWithinWin	445
7.18	Bounding Box-Funktionen	446
7.18.1	Box2D	446
7.18.2	Box3D	447
7.18.3	ST_EstimatedExtent	447
7.18.4	ST_Expand	448
7.18.5	ST_Extent	450
7.18.6	ST_3DExtent	451
7.18.7	ST_MakeBox2D	452
7.18.8	ST_3DMakeBox	453
7.18.9	ST_XMax	453
7.18.10	ST_XMin	454
7.18.11	ST_YMax	455
7.18.12	ST_YMin	456
7.18.13	ST_ZMax	457
7.18.14	ST_ZMin	458
7.19	Kilometrierung	459
7.19.1	ST_LineInterpolatePoint	459
7.19.2	ST_3DLineInterpolatePoint	461
7.19.3	ST_LineInterpolatePoints	462
7.19.4	ST_LineLocatePoint	463

---

7.19.5	ST_LineSubstring	464
7.19.6	ST_LocateAlong	466
7.19.7	ST_LocateBetween	467
7.19.8	ST_LocateBetweenElevations	469
7.19.9	ST_InterpolatePoint	469
7.19.10	ST_AddMeasure	470
7.20	Trajektorie-Funktionen	471
7.20.1	ST_IsValidTrajectory	471
7.20.2	ST_ClosestPointOfApproach	472
7.20.3	ST_DistanceCPA	473
7.20.4	ST_CPAWithin	473
7.21	Version Funktionen	474
7.21.1	PostGIS_Extensions_Upgrade	474
7.21.2	PostGIS_Full_Version	475
7.21.3	PostGIS_GEOS_Version	476
7.21.4	PostGIS_GEOS_Compiled_Version	476
7.21.5	PostGIS_Liblwgeom_Version	477
7.21.6	PostGIS_LibXML_Version	477
7.21.7	PostGIS_LibJSON_Version	478
7.21.8	PostGIS_Lib_Build_Date	478
7.21.9	PostGIS_Lib_Version	478
7.21.10	PostGIS_PROJ_Version	479
7.21.11	PostGIS_PROJ_Compiled_Version	479
7.21.12	PostGIS_Wagyu_Version	480
7.21.13	PostGIS_Scripts_Build_Date	480
7.21.14	PostGIS_Scripts_Installed	481
7.21.15	PostGIS_Scripts_Released	482
7.21.16	PostGIS_Version	482
7.22	PostGIS Grand Unified Custom Variables (GUCs)	483
7.22.1	postgis.backend	483
7.22.2	postgis.gdal_datapath	483
7.22.3	postgis.gdal_enabled_drivers	484
7.22.4	postgis.enable_outdb_rasters	485
7.22.5	postgis.gdal_vsi_options	486
7.23	Funktionen zur Fehlersuche	487
7.23.1	PostGIS_AddBBox	487
7.23.2	PostGIS_DropBBox	488
7.23.3	PostGIS_HasBBox	488

---

<b>8 Referenz der SFCGAL-Funktionen</b>	<b>490</b>
8.1 Verwaltungsfunktionen der SFCGAL	490
8.1.1 postgis_sfcgal_version	490
8.1.2 postgis_sfcgal_full_version	490
8.2 SFCGAL-Accessoren und -Setzer	491
8.2.1 CG_ForceLHR	491
8.2.2 CG_IsPlanar	491
8.2.3 CG_IsSolid	492
8.2.4 CG_MakeSolid	492
8.2.5 CG_Orientation	492
8.2.6 CG_Area	493
8.2.7 CG_3DArea	493
8.2.8 CG_Volume	494
8.2.9 ST_ForceLHR	495
8.2.10 ST_IsPlanar	496
8.2.11 ST_IsSolid	496
8.2.12 ST_MakeSolid	497
8.2.13 ST_Orientation	497
8.2.14 ST_3DArea	498
8.2.15 ST_Volume	499
8.3 SFCGAL-Verarbeitung und Beziehungsfunktionen	500
8.3.1 CG_Intersection	500
8.3.2 CG_Intersects	500
8.3.3 CG_3DIntersects	501
8.3.4 CG_Difference	502
8.3.5 ST_3DDifference	503
8.3.6 CG_3DDifference	503
8.3.7 CG_Distance	504
8.3.8 CG_3DDistance	505
8.3.9 ST_3DConvexHull	506
8.3.10 CG_3DConvexHull	506
8.3.11 ST_3DIntersection	507
8.3.12 CG_3DIntersection	508
8.3.13 CG_Union	510
8.3.14 ST_3DUnion	511
8.3.15 CG_3DUnion	511
8.3.16 ST_AlphaShape	512
8.3.17 CG_AlphaShape	513
8.3.18 CG_ApproxConvexPartition	516

---

8.3.19	ST_ApproximateMedialAxis . . . . .	517
8.3.20	CG_ApproximateMedialAxis . . . . .	518
8.3.21	ST_ConstrainedDelaunayTriangles . . . . .	519
8.3.22	CG_ConstrainedDelaunayTriangles . . . . .	519
8.3.23	ST_Extrude . . . . .	521
8.3.24	CG_Extrude . . . . .	521
8.3.25	CG_ExtrudeStraightSkeleton . . . . .	523
8.3.26	CG_GreeneApproxConvexPartition . . . . .	524
8.3.27	ST_MinkowskiSum . . . . .	525
8.3.28	CG_MinkowskiSum . . . . .	525
8.3.29	ST_OptimalAlphaShape . . . . .	527
8.3.30	CG_OptimalAlphaShape . . . . .	528
8.3.31	CG_OptimalConvexPartition . . . . .	530
8.3.32	CG_StraightSkeleton . . . . .	531
8.3.33	ST_StraightSkeleton . . . . .	532
8.3.34	ST_Tesselate . . . . .	534
8.3.35	CG_Tesselate . . . . .	534
8.3.36	CG_Triangulate . . . . .	536
8.3.37	CG_Visibility . . . . .	537
8.3.38	CG_YMonotonePartition . . . . .	538
<b>9</b>	<b>Topologie</b>	<b>540</b>
9.1	Topologische Datentypen . . . . .	540
9.1.1	getfaceedges_returntype . . . . .	540
9.1.2	TopoGeometry . . . . .	541
9.1.3	validatetopology_returntype . . . . .	541
9.2	Topologische Domänen . . . . .	542
9.2.1	TopoElement . . . . .	542
9.2.2	TopoElementArray . . . . .	542
9.3	Verwaltung von Topologie und TopoGeometry . . . . .	543
9.3.1	AddTopoGeometryColumn . . . . .	543
9.3.2	RenameTopoGeometryColumn . . . . .	544
9.3.3	DropTopology . . . . .	545
9.3.4	RenameTopology . . . . .	545
9.3.5	DropTopoGeometryColumn . . . . .	546
9.3.6	Populate_Topology_Layer . . . . .	546
9.3.7	TopologySummary . . . . .	547
9.3.8	ValidateTopology . . . . .	548
9.3.9	ValidateTopologyRelation . . . . .	550

---



9.3.10	FindTopology	551
9.3.11	FindLayer	551
9.4	Verwaltung der Topologie-Statistiken	552
9.5	Topologie Konstruktoren	552
9.5.1	CreateTopology	552
9.5.2	CopyTopology	553
9.5.3	ST_InitTopoGeo	554
9.5.4	ST_CreateTopoGeo	554
9.5.5	TopoGeo_AddPoint	555
9.5.6	TopoGeo_AddLineString	556
9.5.7	TopoGeo_AddPolygon	556
9.5.8	TopoGeo_LoadGeometry	557
9.6	Topologie Editoren	557
9.6.1	ST_AddIsoNode	557
9.6.2	ST_AddIsoEdge	558
9.6.3	ST_AddEdgeNewFaces	558
9.6.4	ST_AddEdgeModFace	559
9.6.5	ST_RemEdgeNewFace	560
9.6.6	ST_RemEdgeModFace	560
9.6.7	ST_ChangeEdgeGeom	561
9.6.8	ST_ModEdgeSplit	562
9.6.9	ST_ModEdgeHeal	563
9.6.10	ST_NewEdgeHeal	563
9.6.11	ST_MoveIsoNode	564
9.6.12	ST_NewEdgesSplit	564
9.6.13	ST_RemoveIsoNode	565
9.6.14	ST_RemoveIsoEdge	566
9.7	Zugriffsfunktionen zur Topologie	566
9.7.1	GetEdgeByPoint	566
9.7.2	GetFaceByPoint	567
9.7.3	GetFaceContainingPoint	568
9.7.4	GetNodeByPoint	569
9.7.5	GetTopologyID	569
9.7.6	GetTopologySRID	570
9.7.7	GetTopologyName	570
9.7.8	ST_GetFaceEdges	571
9.7.9	ST_GetFaceGeometry	572
9.7.10	GetRingEdges	572
9.7.11	GetNodeEdges	573

---

9.8	Topologie Verarbeitung	574
9.8.1	Polygonize	574
9.8.2	AddNode	574
9.8.3	AddEdge	575
9.8.4	AddFace	576
9.8.5	ST_Simplify	578
9.8.6	RemoveUnusedPrimitives	578
9.9	TopoGeometry Konstruktoren	579
9.9.1	CreateTopoGeom	579
9.9.2	toTopoGeom	581
9.9.3	TopoElementArray_Agg	582
9.9.4	TopoElement	582
9.10	TopoGeometry Editoren	583
9.10.1	clearTopoGeom	583
9.10.2	TopoGeom_addElement	584
9.10.3	TopoGeom_remElement	584
9.10.4	TopoGeom_addTopoGeom	585
9.10.5	toTopoGeom	585
9.11	TopoGeometry Accessors	585
9.11.1	GetTopoGeomElementArray	585
9.11.2	GetTopoGeomElements	586
9.11.3	ST_SRID	586
9.12	TopoGeometry Ausgabe	587
9.12.1	AsGML	587
9.12.2	AsTopoJSON	590
9.13	Räumliche Beziehungen einer Topologie	591
9.13.1	Equals	591
9.13.2	Intersects	592
9.14	Importieren und Exportieren von Topologien	592
9.14.1	Verwendung des Topologie-Exporters	593
9.14.2	Verwendung des Topologie-Importers	593
<b>10</b>	<b>Rasterdatenverwaltung, -abfrage und Anwendungen</b>	<b>594</b>
10.1	Laden und Erstellen von Rastertabellen	594
10.1.1	Verwendung von raster2pgsql zum Laden von Rastern	594
10.1.1.1	Beispiel für die Verwendung	594
10.1.1.2	raster2pgsql-Optionen	595
10.1.2	Erzeugung von Rastern mit den PostGIS Rasterfunktionen	597
10.1.3	Verwendung von "out db"-Wolkenrastern	597

---

10.2 Raster Katalog . . . . .	598
10.2.1 Rasterspalten Katalog . . . . .	598
10.2.2 Raster Übersicht/Raster Overviews . . . . .	599
10.3 Eigene Anwendungen mit PostGIS Raster erstellen . . . . .	600
10.3.1 PHP Beispiel: Ausgabe mittels ST_AsPNG in Verbindung mit anderen Rasterfunktionen . . . . .	600
10.3.2 ASP.NET C# Beispiel: Ausgabe mittels ST_AsPNG in Verbindung mit anderen Rasterfunktionen . . . . .	601
10.3.3 Applikation für die Java-Konsole, welche eine Rasterabfrage als Bilddatei ausgibt . . . . .	603
10.3.4 Verwenden Sie PLPython um Bilder via SQL herauszuschreiben . . . . .	604
10.3.5 FASTER mit PSQL ausgeben . . . . .	604
<b>11 Referenz Raster</b>	<b>606</b>
11.1 Datentypen zur Unterstützung von Rastern. . . . .	607
11.1.1 geomval . . . . .	607
11.1.2 addbandarg . . . . .	607
11.1.3 rastbandarg . . . . .	607
11.1.4 raster . . . . .	608
11.1.5 reclassarg . . . . .	608
11.1.6 summarystats . . . . .	609
11.1.7 unionarg . . . . .	609
11.2 Rastermanagement . . . . .	610
11.2.1 AddRasterConstraints . . . . .	610
11.2.2 DropRasterConstraints . . . . .	612
11.2.3 AddOverviewConstraints . . . . .	613
11.2.4 DropOverviewConstraints . . . . .	614
11.2.5 PostGIS_GDAL_Version . . . . .	614
11.2.6 PostGIS_Raster_Lib_Build_Date . . . . .	614
11.2.7 PostGIS_Raster_Lib_Version . . . . .	615
11.2.8 ST_GDALDrivers . . . . .	615
11.2.9 ST_Contour . . . . .	620
11.2.10 ST_InterpolateRaster . . . . .	621
11.2.11 UpdateRasterSRID . . . . .	622
11.2.12 ST_CreateOverview . . . . .	622
11.3 Raster Constructors . . . . .	623
11.3.1 ST_AddBand . . . . .	623
11.3.2 ST_AsRaster . . . . .	626
11.3.3 ST_Band . . . . .	628
11.3.4 ST_MakeEmptyCoverage . . . . .	629
11.3.5 ST_MakeEmptyRaster . . . . .	631
11.3.6 ST_Tile . . . . .	632

---

11.3.7	ST_Retile	634
11.3.8	ST_FromGDALRaster	634
11.4	Zugriffsfunktionen auf Raster	635
11.4.1	ST_GeoReference	635
11.4.2	ST_Height	636
11.4.3	ST_IsEmpty	637
11.4.4	ST_MemSize	637
11.4.5	ST_MetaData	638
11.4.6	ST_NumBands	639
11.4.7	ST_PixelHeight	639
11.4.8	ST_PixelWidth	640
11.4.9	ST_ScaleX	641
11.4.10	ST_ScaleY	642
11.4.11	ST_RasterToWorldCoord	643
11.4.12	ST_RasterToWorldCoordX	644
11.4.13	ST_RasterToWorldCoordY	645
11.4.14	ST_Rotation	646
11.4.15	ST_SkewX	646
11.4.16	ST_SkewY	647
11.4.17	ST_SRID	648
11.4.18	ST_Summary	648
11.4.19	ST_UpperLeftX	649
11.4.20	ST_UpperLeftY	650
11.4.21	ST_Width	650
11.4.22	ST_WorldToRasterCoord	651
11.4.23	ST_WorldToRasterCoordX	651
11.4.24	ST_WorldToRasterCoordY	652
11.5	Zugriffsfunktionen auf Rasterbänder	653
11.5.1	ST_BandMetaData	653
11.5.2	ST_BandNoDataValue	654
11.5.3	ST_BandIsNoData	655
11.5.4	ST_BandPath	656
11.5.5	ST_BandFileSize	656
11.5.6	ST_BandFileTimestamp	657
11.5.7	ST_BandPixelType	657
11.5.8	ST_MinPossibleValue	658
11.5.9	ST_HasNoBand	659
11.6	Zugriffsfunktionen und Änderungsmethoden für Rasterpixel	660
11.6.1	ST_PixelAsPolygon	660

---

11.6.2	ST_PixelAsPolygons	660
11.6.3	ST_PixelAsPoint	661
11.6.4	ST_PixelAsPoints	662
11.6.5	ST_PixelAsCentroid	663
11.6.6	ST_PixelAsCentroids	663
11.6.7	ST_Value	665
11.6.8	ST_NearestValue	668
11.6.9	ST_SetZ	669
11.6.10	ST_SetM	670
11.6.11	ST_Neighborhood	672
11.6.12	ST_SetValue	674
11.6.13	ST_SetValues	675
11.6.14	ST_DumpValues	683
11.6.15	ST_PixelOfValue	684
11.7	Raster Editoren	685
11.7.1	ST_SetGeoReference	685
11.7.2	ST_SetRotation	687
11.7.3	ST_SetScale	687
11.7.4	ST_SetSkew	688
11.7.5	ST_SetSRID	689
11.7.6	ST_SetUpperLeft	690
11.7.7	ST_Resample	690
11.7.8	ST_Rescale	691
11.7.9	ST_Reskew	693
11.7.10	ST_SnapToGrid	694
11.7.11	ST_Resize	695
11.7.12	ST_Transform	696
11.8	Editoren für Rasterbänder	699
11.8.1	ST_SetBandNoDataValue	699
11.8.2	ST_SetBandIsNoData	700
11.8.3	ST_SetBandPath	701
11.8.4	ST_SetBandIndex	703
11.9	Rasterband Statistik und Analytik	704
11.9.1	ST_Count	704
11.9.2	ST_CountAgg	705
11.9.3	ST_Histogram	706
11.9.4	ST_Quantile	708
11.9.5	ST_SummaryStats	710
11.9.6	ST_SummaryStatsAgg	712

---

11.9.7 ST_ValueCount . . . . .	713
11.10 Rastereingabe . . . . .	715
11.10.1 ST_RastFromWKB . . . . .	715
11.10.2 ST_RastFromHexWKB . . . . .	716
11.11 Ausgabe von Rastern . . . . .	717
11.11.1 ST_AsBinary/ST_AsWKB . . . . .	717
11.11.2 ST_AsHexWKB . . . . .	718
11.11.3 ST_AsGDALRaster . . . . .	718
11.11.4 ST_AsJPEG . . . . .	720
11.11.5 ST_AsPNG . . . . .	721
11.11.6 ST_AsTIFF . . . . .	722
11.12 Rasterverarbeitung: Kartenalgebra . . . . .	722
11.12.1 ST_Clip . . . . .	722
11.12.2 ST_ColorMap . . . . .	726
11.12.3 ST_Grayscale . . . . .	729
11.12.4 ST_Intersection . . . . .	731
11.12.5 ST_MapAlgebra (callback function version) . . . . .	732
11.12.6 ST_MapAlgebra (expression version) . . . . .	739
11.12.7 ST_MapAlgebraExpr . . . . .	741
11.12.8 ST_MapAlgebraExpr . . . . .	744
11.12.9 ST_MapAlgebraFct . . . . .	748
11.12.10 ST_MapAlgebraFct . . . . .	752
11.12.11 ST_MapAlgebraFctNgb . . . . .	756
11.12.12 ST_Reclass . . . . .	758
11.12.13 ST_Union . . . . .	760
11.13 Integrierte Map Algebra Callback Funktionen . . . . .	761
11.13.1 ST_Distinct4ma . . . . .	761
11.13.2 ST_InvDistWeight4ma . . . . .	762
11.13.3 ST_Max4ma . . . . .	763
11.13.4 ST_Mean4ma . . . . .	764
11.13.5 ST_Min4ma . . . . .	765
11.13.6 ST_MinDist4ma . . . . .	766
11.13.7 ST_Range4ma . . . . .	767
11.13.8 ST_StdDev4ma . . . . .	768
11.13.9 ST_Sum4ma . . . . .	769
11.14 Rasterverarbeitung: DEM (Elevation) . . . . .	770
11.14.1 ST_Aspect . . . . .	770
11.14.2 ST_HillShade . . . . .	772
11.14.3 ST_Roughness . . . . .	774

---

11.14.4 ST_Slope . . . . .	774
11.14.5 ST_TPI . . . . .	776
11.14.6 ST_TRI . . . . .	777
11.15 Rasterverarbeitung: Raster zu Geometrie . . . . .	777
11.15.1 Box3D . . . . .	777
11.15.2 ST_ConvexHull . . . . .	778
11.15.3 ST_DumpAsPolygons . . . . .	779
11.15.4 ST_Envelope . . . . .	780
11.15.5 ST_MinConvexHull . . . . .	781
11.15.6 ST_Polygon . . . . .	782
11.16 Rasteroperatoren . . . . .	784
11.16.1 && . . . . .	784
11.16.2 &< . . . . .	784
11.16.3 &> . . . . .	785
11.16.4 = . . . . .	786
11.16.5 @ . . . . .	786
11.16.6 ~= . . . . .	787
11.16.7 ~ . . . . .	787
11.17 Räumliche Beziehungen von Rastern und Rasterbändern . . . . .	788
11.17.1 ST_Contains . . . . .	788
11.17.2 ST_ContainsProperly . . . . .	789
11.17.3 ST_Covers . . . . .	790
11.17.4 ST_CoveredBy . . . . .	791
11.17.5 ST_Disjoint . . . . .	791
11.17.6 ST_Intersects . . . . .	792
11.17.7 ST_Overlaps . . . . .	793
11.17.8 ST_Touches . . . . .	794
11.17.9 ST_SameAlignment . . . . .	795
11.17.10 ST_NotSameAlignmentReason . . . . .	796
11.17.11 ST_Within . . . . .	797
11.17.12 ST_DWithin . . . . .	798
11.17.13 ST_DFullyWithin . . . . .	799
11.18 Raster Tipps . . . . .	800
11.18.1 Out-DB Raster . . . . .	800
11.18.1.1 Das Verzeichnis enthält eine Vielzahl an Dateien . . . . .	800
11.18.1.2 Die maximale Anzahl geöffneter Dateien . . . . .	800
11.18.1.2.1 Die maximale Anzahl geöffneter Dateien für das ganze System . . . . .	801
11.18.1.2.2 Die maximale Anzahl geöffneter Dateien pro Prozess . . . . .	801

---

<b>12 PostGIS Extras</b>	<b>803</b>
12.1 Adressennormierer	803
12.1.1 Funktionsweise des Parsers	803
12.1.2 Adressennormierer Datentypen	804
12.1.2.1 stdaddr	804
12.1.3 Adressennormierer Tabellen	804
12.1.3.1 rules table	804
12.1.3.2 lex table	807
12.1.3.3 gaz table	808
12.1.4 Adressennormierer Funktionen	808
12.1.4.1 debug_standardize_address	808
12.1.4.2 parse_address	809
12.1.4.3 standardize_address	810
12.2 Tiger Geokoder	812
12.2.1 Drop_Indexes_Generate_Script	812
12.2.2 Drop_Nation_Tables_Generate_Script	814
12.2.3 Drop_State_Tables_Generate_Script	814
12.2.4 Geocode	815
12.2.5 Geocode_Intersection	817
12.2.6 Get_Geocode_Setting	818
12.2.7 Get_Tract	819
12.2.8 Install_Missing_Indexes	820
12.2.9 Loader_Generate_Census_Script	821
12.2.10 Loader_Generate_Script	823
12.2.11 Loader_Generate_Nation_Script	825
12.2.12 Missing_Indexes_Generate_Script	826
12.2.13 Normalize_Address	826
12.2.14 Pagc_Normalize_Address	828
12.2.15 Pprint_Addy	830
12.2.16 Reverse_Geocode	831
12.2.17 Topology_Load_Tiger	833
12.2.18 Set_Geocode_Setting	835
<b>13 PostGIS Spezialfunktionen Index</b>	<b>836</b>
13.1 PostGIS-Aggregat-Funktionen	836
13.2 PostGIS-Fenster-Funktionen	837
13.3 PostGIS SQL-MM-kompatible Funktionen	837
13.4 PostGIS-Funktionen zur Unterstützung der Geografie	841
13.5 PostGIS-Funktionen zur Unterstützung von Rastern	843



13.6 PostGIS Geometrie / Geographie / Raster Dump Funktionen . . . . .	849
13.7 PostGIS-Box-Funktionen . . . . .	849
13.8 PostGIS-Funktionen, die 3D unterstützen . . . . .	850
13.9 PostGIS Funktionen zur Unterstützung gekrümmter Geometrien . . . . .	856
13.10 PostGIS-Funktionen zur Unterstützung polyedrischer Flächen . . . . .	859
13.11 PostGIS Funktionsunterstützungsmatrix . . . . .	862
13.12 Neue, erweiterte oder geänderte PostGIS-Funktionen . . . . .	880
13.12.1 PostGIS-Funktionen neu oder erweitert in 3.5 . . . . .	880
13.12.2 PostGIS-Funktionen neu oder erweitert in 3.4 . . . . .	882
13.12.3 PostGIS-Funktionen neu oder erweitert in 3.3 . . . . .	884
13.12.4 PostGIS-Funktionen neu oder erweitert in 3.2 . . . . .	884
13.12.5 PostGIS-Funktionen neu oder erweitert in 3.1 . . . . .	886
13.12.6 PostGIS-Funktionen neu oder erweitert in 3.0 . . . . .	887
13.12.7 PostGIS-Funktionen neu oder erweitert in 2.5 . . . . .	889
13.12.8 PostGIS-Funktionen neu oder erweitert in 2.4 . . . . .	891
13.12.9 PostGIS-Funktionen neu oder erweitert in 2.3 . . . . .	892
13.12.10 PostGIS-Funktionen neu oder erweitert in 2.2 . . . . .	894
13.12.11 PostGIS-Funktionen neu oder erweitert in 2.1 . . . . .	897
13.12.12 PostGIS-Funktionen neu oder erweitert in 2.0 . . . . .	903
13.12.13 PostGIS-Funktionen neu oder erweitert in 1.5 . . . . .	914
13.12.14 PostGIS-Funktionen neu oder erweitert in 1.4 . . . . .	916
13.12.15 PostGIS-Funktionen neu oder erweitert in 1.3 . . . . .	917
<b>14 Meldung von Problemen</b> . . . . .	<b>918</b>
14.1 Software Bugs melden . . . . .	918
14.2 Probleme mit der Dokumentation melden . . . . .	918
<b>A Anhang</b> . . . . .	<b>920</b>
A.1 PostGIS 3.6.0 . . . . .	920

---

## Abstract

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



Dieses Handbuch beschreibt die Version 3.6.0dev



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

# Chapter 1

## Einführung

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

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

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

### 1.1 Projektleitung

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

**Raúl Marín Rodríguez** MVT-Unterstützung, Fehlerbehebung, Leistungs- und Stabilitätsverbesserungen, GitHub-Kuratierung, Anpassung von PostGIS an PostgreSQL-Versionen

**Regina Obe** CI- und Website-Wartung, Windows-Produktions- und Experimental-Builds, Dokumentation, Anpassung von PostGIS an PostgreSQL-Releases, X3D-Unterstützung, TIGER-Geocoder-Unterstützung, Verwaltungsfunktionen.

**Darafei Praliaskouski** Indexverbesserungen, Fehlerbehebung und Verbesserungen der Geometrie-/Geografiefunktionen, SFC-GAL, Raster, GitHub-Kuratierung und CI-Wartung.

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

**Sandro Santilli** Fehlerkorrekturen und -wartung, ci-Wartung, Git-Mirror-Verwaltung, Verwaltungsfunktionen, Integration neuer GEOS-Funktionen und Anpassung an GEOS-Releases, Topologie-Unterstützung sowie Raster-Framework und Low-Level-API-Funktionen.

## 1.2 Aktuelle Kernentwickler

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

**Loïc Bartoletti** SFCGAL-Verbesserungen und -Wartung sowie CI-Unterstützung

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

**Martin Davis** GEOS-Verbesserungen und Dokumentation

**Björn Harrtell** MapBox Vector Tile, GeoBuf, und Flatgeobuf Funktionen. Gitea-Tests und GitLab-Experimente.

**Aliaksandr Kalenik** Geometrieverarbeitung, PostgreSQL gist, allgemeine Fehlerbehebung

## 1.3 Frühere Kernentwickler

**Bborie Park** Vorheriges PSC-Mitglied. Rasterentwicklung, Integration mit GDAL, Rasterloader, Benutzerunterstützung, allgemeine Fehlerbehebung, Testen auf verschiedenen Betriebssystemen (Slackware, Mac, Windows und mehr)

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

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

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

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

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

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

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

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

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

**Pierre Racine** Architekt der PostGIS-Rasterimplementierung. Raster-Gesamtarchitektur, Prototyping, Programmierunterstützung

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

---

## 1.4 Weitere Mitwirkende

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

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

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

- [Cadcorp](#)
- [Camptocamp](#)
- [Carto](#)
- [Knackige Daten](#)
- [Stadt Boston \(DND\)](#)
- [Stadt Helsinki](#)
- [Clevere Elefanten-Lösungen](#)
- [Genossenschaft Alveo](#)
- [Deimos Raumfahrt](#)
- [Faunalia](#)
- [Geografische Daten BC](#)
- [HighGo](#)
- [Hunter Systeme Gruppe](#)
- [INIA-CSIC](#)
- [ISciences, LLC](#)
- [Kontur](#)
- [Lidwala Beratende Ingenieure](#)
- [LISAsoft](#)
- [Logische Rückverfolgung & Tracing International AG](#)
- [Maponics](#)
- [Michigan Tech Forschungsinstitut](#)
- [Natürliche Ressourcen Kanada](#)
- [Norwegisches Institut für Forstwirtschaft und Landschaftspflege](#)
- [Norwegisches Institut für Bioökonomieforschung \(NIBIO\)](#)
- [OSGeo](#)
- [Oslandia](#)
- [Palantir-Technologien](#)
- [Paragon Gesellschaft](#)
- [R3 GIS](#)
- [Refraktionen Forschung](#)
- [Region Toskana - SITA](#)
- [Sichere Software](#)
- [Sirius Corporation plc](#)
- [Stadt Uster](#)
- [UC Davis Zentrum für durch Vektoren übertragene Krankheiten](#)
- [Universität Laval](#)
- [U.S. Außenministerium \(HIU\)](#)
- [Zonar-Systeme](#)

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

---

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

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

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

**Wichtige Support-Bibliotheken** Die **GEOS** Geometrieoperationsbibliothek

Die **GDAL** Geospatial Data Abstraction Library (GDAL) wurde verwendet, um einen Großteil der in PostGIS 2 eingeführten Rasterfunktionalität zu unterstützen. Verbesserungen, die in GDAL benötigt werden, um PostGIS zu unterstützen, werden dem GDAL-Projekt zur Verfügung gestellt.

Die **PROJ** kartographische Projektionsbibliothek

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

## Chapter 2

# PostGIS Installation

Dieses Kapitel erläutert die notwendigen Schritte zur Installation von PostGIS.

### 2.1 Kurzfassung

Zum Kompilieren müssen die Abhängigkeiten im Suchpfad eingetragen sein:

```
tar -xvzf postgis-3.6.0dev.tar.gz
cd postgis-3.6.0dev
./configure
make
make install
```

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

### 2.2 Kompilierung und Installation des Quellcodes: Detaillierte Beschreibung

---

#### 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 allgemeine Kompilierungsanweisungen. Wenn Sie für Windows etc. oder ein anderes Betriebssystem kompilieren, können Sie zusätzliche, detailliertere Hilfe unter [PostGIS User contributed compile guides](#) und [PostGIS Dev Wiki](#) finden.

Vorgefertigte Pakete für verschiedene Betriebssysteme sind in [PostGIS Vorgefertigte Pakete](#) aufgeführt.

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

---

Das PostGIS-Modul ist eine Erweiterung für den PostgreSQL-Backend-Server. Als solches benötigt PostGIS 3.6.0dev *vollen Zugriff auf die Header des PostgreSQL-Servers*, um kompiliert werden zu können. Es kann mit den PostgreSQL-Versionen 12 - 17 kompiliert werden. Frühere Versionen von PostgreSQL werden *nicht unterstützt*.

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

---



**Note**

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



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

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

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

## 2.2.1 Nutzung des Quellcodes

Das PostGIS Quellarchiv kann von der Download Webseite <https://postgis.net/stuff/postgis-3.6.0dev.tar.gz> bezogen werden.

```
wget https://postgis.net/stuff/postgis-3.6.0dev.tar.gz
tar -xvzf postgis-3.6.0dev.tar.gz
cd postgis-3.6.0dev
```

Dadurch wird das Verzeichnis `postgis-3.6.0dev` im aktuellen Arbeitsverzeichnis erzeugt.

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

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

Um die Installation fortzusetzen ist in das neu erstellte Verzeichnis `postgis-3.6.0dev` zu wechseln.

```
./configure
```

## 2.2.2 Systemvoraussetzungen

Zur Kompilation und Anwendung stellt PostGIS die folgenden Systemanforderungen:

### Notwendige Systemvoraussetzungen

- PostgreSQL 12 - 17. Eine vollständige Installation von PostgreSQL (einschließlich der Server-Header) ist erforderlich. PostgreSQL ist verfügbar unter <https://www.postgresql.org> .  
Eine vollständige PostgreSQL/PostGIS-Unterstützungsmatrix und PostGIS/GEOS-Unterstützungsmatrix finden Sie unter <https://trac.osgeo.org/postgis/wiki/UsersWikiPostgreSQLPostGIS>
- GNU C Compiler (`gcc`). Es können auch andere ANSI C Compiler zur PostGIS Kompilation verwendet werden, aber die Kompilation mit `gcc` macht die geringsten Probleme.
- GNU Make (`gmake` oder `make`). Für viele Systeme ist GNU `make` die Standardversion von `make`. Überprüfe die Version durch `make -v`. Andere Versionen von `make` können das PostGIS `Makefile` nicht richtig ausführen.
- Proj Reprojektion Bibliothek. Proj 6.1 oder höher ist erforderlich. Die Proj-Bibliothek wird zur Unterstützung der Koordinatenreprojektion in PostGIS verwendet. Proj steht zum Download zur Verfügung unter <https://proj.org/> .
- GEOS-Geometriebibliothek, Version 3.8.0 oder höher, aber GEOS 3.12+ ist erforderlich, um alle neuen Funktionen und Eigenschaften voll nutzen zu können. GEOS steht zum Download bereit unter <https://libgeos.org> .
- LibXML2, Version 2.5.x oder höher. LibXML2 wird derzeit in einigen Importfunktionen (`ST_GeomFromGML` und `ST_GeomFromKML`) verwendet. LibXML2 steht unter <https://gitlab.gnome.org/GNOME/libxml2/-/releases> zum Download bereit.

- 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 3+ is preferred. This is required for raster support. <https://gdal.org/download.html>.
- Wenn mit PostgreSQL+JIT kompiliert wird, ist die LLVM-Version  $\geq 6$  erforderlich <https://trac.osgeo.org/postgis/ticket/4125>.

### Optionale Systemanforderungen

- GDAL (pseudo optional) nur wenn Sie kein Rasterunterstützung möchten, können Sie es weglassen. Sorgen Sie außerdem dafür das Treiber, die Sie brauchen wie in Section 3.2 beschrieben, aktiviert sind.
- GTK (benötigt GTK+2.0, 2.8+) um den "shp2pgsql-gui shape file loader" zu kompilieren. <http://www.gtk.org/> .
- SFCGAL, 1.4.1 or higher is required and 1.5.0+ is needed to be able to use all functionality. SFCGAL can be used to provide additional 2D and 3D advanced analysis functions to PostGIS cf Chapter 8. And also allow to use SFCGAL rather than GEOS for some 2D functions provided by both backends (like `ST_Intersection` or `ST_Area`, for instance). A PostgreSQL configuration variable `postgis.backend` allow end user to control which backend he want to use if SFCGAL is installed (GEOS by default). Nota: SFCGAL 1.2 require at least CGAL 4.3 and Boost 1.54 (cf: <https://sfcgal.org> <https://gitlab.com/sfcgal/SFCGAL/>).
- Um den Section 12.1 zu bauen, benötigen Sie auch PCRE <http://www.pcre.org> (welches in der Regel auf Nix-Systemen bereits installiert ist). Section 12.1 wird automatisch gebaut, wenn es eine PCRE-Bibliothek erkennt, oder Sie geben einen gültigen `--with-pcre-dir=/path/to/pcre` während configure an.
- Um `ST_AsMVT` verwenden zu können, wird die `protobuf-c` Bibliothek (für die Anwendung) und der `protoc-c` Compiler (für die Kompilation) benötigt. Weiters ist `pkg-config` erforderlich um die korrekte Minimumversion von `protobuf-c` zu bestimmen. Siehe [protobuf-c](#).
- CUnit (CUnit). Wird für Regressionstest benötigt. <http://cunit.sourceforge.net/>
- DocBook (`xsltproc`) ist für die Kompilation der Dokumentation notwendig. Docbook steht unter <http://www.docbook.org/> zur Verfügung.
- DBLatex (`dblatex`) ist zur Kompilation der Dokumentation im PDF-Format nötig. DBLatex liegt unter <http://dblatex.sourceforge.net/> vor.
- ImageMagick (`convert`) wird zur Erzeugung von Bildern für die Dokumentation benötigt. ImageMagick kann von <http://www.imagemagick.org/> bezogen werden.

### 2.2.3 Konfiguration

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

#### `./configure`

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

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

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

**--prefix=PREFIX** 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; 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, die für die Durchführung von GeomFromKML/GML-Prozessen erforderlich ist. Normalerweise wird sie gefunden, wenn Sie libxml installiert haben, aber wenn nicht oder Sie eine bestimmte Version verwenden wollen, müssen Sie PostGIS auf eine bestimmte `xml2-config` Datei verweisen, damit Softwareinstallationen das LibXML-Installationsverzeichnis finden können. Verwenden Sie diesen Parameter (**--with-xml2config=/path/to/xml2-config**), um manuell eine bestimmte LibXML-Installation anzugeben, gegen die PostGIS gebaut werden soll.
- with-projdir=DIR** Proj4 ist eine Bibliothek, die von PostGIS zur Koordinatentransformation benötigt wird. Benutzen Sie bitte diesen Parameter (**--with-projdir=/path/to/projdir**) um ein bestimmtes Proj4 Installationsverzeichnis anzugeben, für das PostGIS kompiliert werden soll.
- with-libiconv=DIR** Das Verzeichnis in dem iconv installiert ist.
- with-jsongdir=DIR** **JSON-C** ist eine MIT-lizenzierte JSON Bibliothek, die von PostGIS für ST\_GeomFromJSON benötigt wird. Benutzen Sie bitte diesen Parameter (**--with-jsongdir=/path/to/jsongdir**), 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.
- without-raster** Ohne Rasterunterstützung kompilieren.
- without-topology** Ausschalten der Topologie Unterstützung. Es existiert keine entsprechende Bibliothek, da sich die gesamte benötigte Logik in der postgis-3.6.0dev Bibliothek befindet.
- with-gettext=no** Standardmäßig versucht PostGIS gettext zu detektieren und kompiliert mit gettext Unterstützung. Wenn es allerdings zu Inkompatibilitätsproblemen kommt, die zu einem Zusammenbrechen des Loader führen, so können Sie das mit diesem Befehl zur Gänze deaktivieren. Siehe Ticket <http://trac.osgeo.org/postgis/ticket/748> für ein Beispiel wie dieses Problem gelöst werden kann. Sie verpassen nicht viel, wenn Sie dies deaktivieren, da es für die internationale Hilfe zum GUI Loader/Label verwendet wird, welcher nicht dokumentiert und immer noch experimentell ist.
- with-sfcgal=PATH** Ohne diesen Switch wird PostGIS ohne sfcgal Unterstützung installiert. `PATH` ist ein optionaler Parameter, welcher einen alternativen Pfad zu sfcgal-config angibt.
-

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

---

#### Note



Wenn Sie PostGIS vom [Code Repository](#) bezogen haben, müssen Sie zu allererst das Skript ausführen

**./autogen.sh**

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

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

---

## 2.2.4 Build-Prozess

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

### make

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

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

### make comments

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

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

### make cheatsheets

## 2.2.5 Build-Prozess für die PostGIS Extensions und deren Bereitstellung

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

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

### make comments

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

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

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

---

**Note**

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

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

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

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

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

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

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

name	default_version	installed_version
address_standardizer	3.6.0dev	3.6.0dev
address_standardizer_data_us	3.6.0dev	3.6.0dev
postgis	3.6.0dev	3.6.0dev
postgis_raster	3.6.0dev	3.6.0dev
postgis_sfcgal	3.6.0dev	
postgis_tiger_geocoder	3.6.0dev	3.6.0dev
postgis_topology	3.6.0dev	

(6 rows)

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

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

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

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

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

```
List of installed extensions
```

```
-[ RECORD 1 ]-----
Name          | postgis
Version       | 3.6.0dev
Schema        | public
Description   | PostGIS geometry, geography, and raster spat..
-[ RECORD 2 ]-----
Name          | postgis_raster
Version       | 3.0.0dev
Schema        | public
Description   | PostGIS raster types and functions
-[ RECORD 3 ]-----
Name          | postgis_tiger_geocoder
Version       | 3.6.0dev
Schema        | tiger
Description   | PostGIS tiger geocoder and reverse geocoder
-[ RECORD 4 ]-----
Name          | postgis_topology
Version       | 3.6.0dev
Schema        | topology
Description   | PostGIS topology spatial types and functions
```

### Warning



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

Wenn Sie 3.6.0dev ohne unser wunderbares Extension System installiert haben, können Sie auf erweiterungsbasiert wechseln, indem Sie folgende Befehle ausführen, welche die Funktionen in ihre entsprechenden Erweiterungen pakettieren.

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

## 2.2.6 Softwaretest

Wenn Sie die Kompilation von PostGIS überprüfen wollen:

### make check

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

**Note**

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

**Caution**

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

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

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

.
.
.

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

.
.
.

Running tests

.
.
.

Run tests: 134
Failed: 0

-- if you build with SFCGAL

.
.
.

Running tests

.
.
.

Run tests: 13
Failed: 0

-- if you built with raster support
```

```

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

.
.
.
Running tests

.
.
.
Run tests: 101
Failed: 0

-- topology regress

.
.
.
Running tests

.
.
.
Run tests: 51
Failed: 0

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

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

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

```

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

Für den `address_standardizer`:

```

cd extensions/address_standardizer
make install
make installcheck

```



Die Ausgabe sollte folgendermaßen aussehen:

```

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

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

```

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

```

cd extensions/postgis_tiger_geocoder
make install
make installcheck

```

Die Ausgabe sollte folgendermaßen aussehen:

```

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

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

```

## 2.2.7 Installation

Um PostGIS zu installieren geben Sie bitte folgendes ein

### **make install**

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

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

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

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

### make comments-install



#### Note

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

## 2.3 Installation und Verwendung des Adressennormierers

Die Erweiterung `address_standardizer` musste als getrenntes Paket heruntergeladen werden. Ab PostGIS 2.2 ist es mitgebündelt. Für weitere Informationen zu dem `address_standardizer`, was er kann und wie man ihn für spezielle Bedürfnisse konfigurieren kann, siehe Section 12.1.

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

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

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

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

```
CREATE EXTENSION address_standardizer;
```

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

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

Die Ausgabe sollte wie folgt sein:

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

## 2.4 Installieren, Aktualisieren von Tiger Geocoder und Laden von Daten

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

## 2.4.1 Tiger Geocoder Aktivieren Sie Ihre PostGIS-Datenbank

1. Diese Anleitung geht davon aus, dass Ihre PostgreSQL-Installation bereits die `postgis_tiger_geocoder`-Erweiterung installiert hat.
2. Verbinden Sie sich zu Ihrer Datenbank über `psql`, `pgAdmin` oder ein anderes Werkzeug und führen Sie die folgenden SQL Befehle aus. Wenn Sie in eine Datenbank installieren, die bereits PostGIS beinhaltet, dann müssen Sie den ersten Schritt nicht ausführen. Wenn Sie auch die Erweiterung `fuzzystrmatch` bereits installiert haben, so müssen Sie auch den zweiten Schritt nicht ausführen.

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

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

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

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

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

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

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

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

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

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

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

Folgendermaßen können Sie diese aktivieren:

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

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

6. Erstellen Sie einen Ordner mit der Bezeichnung `gisdata` im Root des Servers oder auf Ihrem lokalen PC, wenn Sie eine schnelle Netzwerkverbindung zu dem Server haben. In diesen Ordner werden die Dateien von Tiger heruntergeladen und aufbereitet. Wenn Sie den Ordner nicht im Root des Servers haben wollen, oder für die Staging-Umgebung in eine anderen Ordner wechseln wollen, dann können Sie das Attribut `staging_fold` in der Tabelle `tiger.loader_variables` editieren.
7. Erstellen Sie einen Ordner "temp" in dem Ordner `gisdata` oder wo immer Sie `staging_fold` haben wollen. Dies wird der Ordner, in dem der Loader die heruntergeladenen Tigerdaten extrahiert.
8. Anschließend führen Sie die SQL Funktion **Loader\_Generate\_Nation\_Script** aus, um sicherzustellen dass die Bezeichnung Ihres benutzerdefinierten Profils verwendet wird und kopieren das Skript in eine `.sh` oder `.bat` Datei. Um zum Beispiel das Skript zum Laden einer Nation zu erzeugen:

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

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

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

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

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

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

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

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

Dies enthält nur Daten, wenn Sie `zcta5` zum Laden markiert haben.

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

```
count
-----
   33931
(1 row)
```

11. Standardmäßig werden die Tabellen, die `bg`, `tract`, `tabblock20` entsprechen, nicht geladen. Diese Tabellen werden vom Geocoder nicht verwendet, aber von den Leuten für Bevölkerungsstatistiken genutzt. Wenn Sie sie als Teil Ihrer Staatslasten laden möchten, führen Sie die folgende Anweisung aus, um sie zu aktivieren.

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

Alternativ können Sie diese Tabellen nach dem Laden der Länderdaten importieren, indem Sie das **Loader\_Generate\_Census\_Script** verwenden

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



### Warning

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

13. 

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

14. Die vorher erzeugten, befehlzeilenorientierten Skripts ausführen.

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

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

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

## 2.4.2 Die Adressnormierer-Extension zusammen mit dem Tiger Geokodierer verwenden

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

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

## 2.4.3 Erforderliche Werkzeuge für das Laden von Tigerdaten

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

Um Daten laden zu können benötigen Sie folgende Werkzeuge:

- Ein Werkzeug, um die Zip-Dateien der Census Webseite zu entpacken.  
Auf UNIX-ähnlichen Systemen: Das Programm `unzip`, das üblicherweise auf den meisten UNIX-ähnlichen Systemen bereits vorinstalliert ist.  
Auf Windows 7-zip, ein freies Werkzeug zum komprimieren/entkomprimieren, das Sie von <http://www.7-zip.org/> herunterladen können.
- Das `shp2pgsql` Kommandozeilenprogramm, welches standardmäßig mit PostGIS mitinstalliert wird.
- `wget`, ein Download-Manager, der üblicherweise auf den meisten UNIX/Linux Systemen vorinstalliert ist.  
Für Windows können Sie vorkompilierte Binärdateien von <http://gnuwin32.sourceforge.net/packages/wget.htm> herunterladen

Wenn Sie von `tiger_2010` aktualisieren, müssen Sie zuerst `Drop_Nation_Tables_Generate_Script` generieren und ausführen. Bevor Sie die Daten eines Staates laden, müssen Sie die landesweiten Daten laden, was Sie mit `Loader_Generate_Nation_Script` tun. Dabei wird ein Loader-Skript für Sie generiert. `Loader_Generate_Nation_Script` ist ein einmaliger Schritt, der für Upgrades (von Tiger-Volkszählungsdaten aus dem Vorjahr) und für Neuinstallationen durchgeführt werden sollte.

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

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

```
SELECT install_missing_indexes();
```

aus.

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

## 2.4.4 Aktualisieren der Tiger Geocoder Installation und Daten

Aktualisieren Sie zunächst Ihre `postgis_tiger_geocoder` Erweiterung wie folgt:

```
ALTER EXTENSION postgis_tiger_geocoder UPDATE;
```

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

```
SELECT drop_nation_tables_generate_script();
```

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

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

### Auf Windows:

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

### Auf Unix/Linux:

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

Unter Section [2.4.1](#) finden Sie Anweisungen zur Ausführung des Generierungsskripts. Dieser Vorgang muss nur einmal durchgeführt werden.



#### Note

Sie können verschiedene Jahrestabellen für die einzelnen Bundesländer verwenden und jedes Bundesland separat aktualisieren. Bevor Sie ein Bundesland aktualisieren, müssen Sie zunächst die Statustabellen des Vorjahres für dieses Bundesland mit `Drop_State_Tables_Generate_Script` löschen.

## 2.5 Übliche Probleme bei der Installation

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

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

```
SELECT version();
```

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

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

```
SELECT postgis_full_version();
```

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

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

## Chapter 3

# PostGIS Verwaltung

### 3.1 Leistungsoptimierung

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

Allgemeine Informationen zur Optimierung von PostgreSQL finden Sie unter [Tuning your PostgreSQL Server](#).

Für PostgreSQL 9.4+ kann die Konfiguration auf Serverebene eingestellt werden, ohne `postgresql.conf` oder `postgresql.auto.conf` zu berühren, indem der Befehl `ALTER SYSTEM` verwendet wird.

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

Zusätzlich zu den Postgres-Einstellungen verfügt PostGIS über einige benutzerdefinierte Einstellungen, die unter [Section 7.22](#) aufgeführt sind.

#### 3.1.1 Startup

Diese Einstellungen werden in `postgresql.conf` konfiguriert:

##### `constraint_exclusion`

- Standard: Partition
- Dies wird im Allgemeinen für die Partitionierung von Tabellen verwendet. Die Voreinstellung hierfür ist "partition", was ideal für PostgreSQL 8.4 und höher ist, da es den Planer dazu zwingt, Tabellen nur dann für die Berücksichtigung von Einschränkungen zu analysieren, wenn sie sich in einer vererbten Hierarchie befinden, und den Planer ansonsten nicht zu bestrafen.

##### `shared_buffers`

- Standard: ~128MB in PostgreSQL 9.6
- Setzen Sie den Wert auf etwa 25 % bis 40 % des verfügbaren RAM. Unter Windows können Sie diesen Wert möglicherweise nicht so hoch einstellen.

`max_worker_processes` Diese Einstellung ist nur für PostgreSQL 9.4+ verfügbar. Für PostgreSQL 9.6+ hat diese Einstellung zusätzliche Bedeutung, da sie die maximale Anzahl von Prozessen steuert, die Sie für parallele Abfragen haben können.



- Voreinstellung: 8
- Legt die maximale Anzahl von Hintergrundprozessen fest, die das System unterstützen kann. Dieser Parameter kann nur beim Start des Servers gesetzt werden.

### 3.1.2 Laufzeit

**work\_mem** - legt die Größe des für Sortiervorgänge und komplexe Abfragen verwendeten Speichers fest

- Standard: 1-4MB
- Anpassung für große Datenbanken, komplexe Abfragen, viel RAM
- Verringern Sie den Wert bei vielen gleichzeitigen Benutzern oder geringem RAM.
- Wenn Sie viel RAM und wenig Entwickler haben:

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

**maintenance\_work\_mem** - die Speichergröße, die für VACUUM, CREATE INDEX, etc. verwendet wird.

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

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

### **max\_parallel\_Arbeiter\_pro\_Gruppe**

Diese Einstellung ist nur für PostgreSQL 9.6+ verfügbar und wirkt sich nur auf PostGIS 2.3+ aus, da nur PostGIS 2.3+ parallele Abfragen unterstützt. Wenn sie auf einen Wert größer als 0 gesetzt wird, können einige Abfragen, z.B. solche, die Beziehungsfunktionen wie `ST_Intersects` beinhalten, mehrere Prozesse verwenden und können dabei mehr als doppelt so schnell laufen. Wenn Sie viele Prozessoren zur Verfügung haben, sollten Sie den Wert auf so viele Prozessoren ändern, wie Sie haben. Stellen Sie außerdem sicher, dass `max_worker_processes` mindestens so hoch ist wie diese Zahl.

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

## 3.2 Konfigurieren der Rasterunterstützung

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

Ab PostGIS 2.1.3 sind out-of-db-Raster und alle Rastertreiber von vornherein deaktiviert. Um diese zu aktivieren, müssen Sie die folgenden Umgebungsvariablen `POSTGIS_GDAL_ENABLED_DRIVERS` und `POSTGIS_ENABLE_OUTDB_RASTERS` in der Serverumgebung setzen. Ab PostGIS 2.2 können Sie einen plattformübergreifenden Ansatz mit Hilfe der entsprechenden Section [7.22](#) verwenden.

Wenn Sie Offline-Raster aktivieren möchten:

```
POSTGIS_ENABLE_OUTDB_RASTERS=1
```

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

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

```
POSTGIS_GDAL_ENABLED_DRIVERS=ENABLE_ALL
```

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

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

**Note**

Unter Windows darf die Treiberliste nicht unter Hochkomma gestellt werden

Das Setzen von Umgebungsvariablen variiert je nach Betriebssystem. Für eine PostgreSQL Installation unter Ubuntu oder Debian mit apt-postgresql, sollte `/etc/postgresql/10/main/environment` editiert werden, wobei sich 10 auf die PostgreSQL-Version und main auf den Cluster bezieht.

Unter Windows können Sie, wenn Sie als Dienst ausgeführt werden, Systemvariablen festlegen, die Sie unter Windows 7 mit einem Rechtsklick auf Computer->Eigenschaften Erweiterte Systemeinstellungen oder im Explorer unter Systemsteuerung\Alle Systemsteuerungselemente\System erreichen. Klicken Sie dann auf *Erweiterte Systemeinstellungen ->Erweitert->Umgebungsvariablen* und fügen Sie neue Systemvariablen hinzu.

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

## 3.3 Erstellung räumlicher Datenbanken

### 3.3.1 Die spatiale Datenbankerweiterung mittels EXTENSION aktivieren

Wenn ab PostgreSQL 9.1 die Extensions/Postgis-Module installiert sind, kann die Datenbank mit Hilfe des EXTENSION-Mechanismus in eine spatiale Datenbank verwandelt werden.

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

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

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

### 3.3.2 Datenbank räumlich aktivieren, ohne EXTENSION zu verwenden (nicht empfohlen)

**Note**

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

Das Hinzufügen von PostGIS-Objekten und Funktionsdefinitionen in Ihre Datenbank erfolgt durch das Laden der verschiedenen sql-Dateien, die sich in `[prefix]/share/contrib` befinden, wie während der Erstellungsphase angegeben.

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

Für einen vollständigen Satz von EPSG-Koordinatensystem-Definitionsbezeichnern können Sie auch die Definitionsdatei `spatial_ref_sys.sql` laden und die Tabelle `spatial_ref_sys` befüllen. Damit können Sie `ST_Transform()`-Operationen an Geometrien durchführen.

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

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

```
DB=[yourdatabase]
SCRIPTSDIR=`pg_config --sharedir`/contrib/postgis-3.4/

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

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

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

## 3.4 Aktualisierung von Geodatenbanken

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

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

PostGIS bietet ein SOFT UPGRADE-Verfahren für Minor- oder Bugfix-Versionen und ein HARD UPGRADE-Verfahren für Major-Versionen.

Bevor Sie versuchen, PostGIS zu aktualisieren, lohnt es sich immer, Ihre Daten zu sichern. Wenn Sie das Flag `-Fc` für `pg_dump` verwenden, können Sie den Dump bei einem HARD UPGRADE immer wiederherstellen.

### 3.4.1 Sanftes Upgrade

Wenn Sie Ihre Datenbank mithilfe von Erweiterungen installiert haben, müssen Sie das Upgrade auch mithilfe des Erweiterungsmodells durchführen. Wenn Sie mit dem alten SQL-Skript installiert haben, sollten Sie Ihre Installation auf Erweiterungen umstellen, da das Skript nicht mehr unterstützt wird.

#### 3.4.1.1 Soft Upgrade 9.1+ mit Erweiterungen

Wenn Sie PostGIS ursprünglich mit Erweiterungen installiert haben, dann müssen Sie das Upgrade auch mit Erweiterungen durchführen. Ein kleines Upgrade mit Erweiterungen ist ziemlich schmerzlos.

Wenn Sie mit PostGIS 3 oder höher arbeiten, sollten Sie die Funktion `PostGIS_Extensions_Upgrade` verwenden, um auf die neueste Version zu aktualisieren, die Sie installiert haben.

```
SELECT postgis_extensions_upgrade();
```

Wenn Sie PostGIS 2.5 oder niedriger verwenden, gehen Sie wie folgt vor:

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

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

```
ALTER EXTENSION postgis UPDATE TO "3.6.0dev";
ALTER EXTENSION postgis_topology UPDATE TO "3.6.0dev";
```

Wenn Sie eine Fehlermeldung wie diese erhalten:

```
No migration path defined for ... to 3.6.0dev
```

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

Wenn Sie eine Meldung wie diese erhalten:

```
Version "3.6.0dev" of extension "postgis" is already installed
```

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

```
ALTER EXTENSION postgis UPDATE TO "3.6.0devnext";
ALTER EXTENSION postgis_topology UPDATE TO "3.6.0devnext";
```



#### Note

Wenn Sie PostGIS ursprünglich ohne Versionsangabe installiert haben, können Sie die Neuinstallation der postgis-Erweiterung vor der Wiederherstellung oft überspringen, da das Backup nur `CREATE EXTENSION postgis` hat und somit die neueste Version bei der Wiederherstellung übernimmt.



#### Note

Wenn Sie die PostGIS-Erweiterung von einer Version vor 3.0.0 aktualisieren, erhalten Sie eine neue Erweiterung `postgis_raster`, die Sie getrost löschen können, wenn Sie die Rasterunterstützung nicht benötigen. Sie können sie wie folgt entfernen:

```
DROP EXTENSION postgis_raster;
```

### 3.4.1.2 Soft Upgrade Pre 9.1+ oder ohne Erweiterungen

Dieser Abschnitt gilt nur für diejenigen, die PostGIS ohne Erweiterungen installiert haben. Wenn Sie Erweiterungen haben und versuchen, mit diesem Ansatz zu aktualisieren, erhalten Sie Meldungen wie:

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

**HINWEIS:** Wenn Sie von PostGIS 1.\* zu PostGIS 2.\* oder von PostGIS 2.\* vor r7409 wechseln, können Sie dieses Verfahren nicht verwenden, sondern müssen ein **HARD UPGRADE** durchführen.

Nach dem Kompilieren und Installieren (`make install`) sollten Sie eine Reihe von `*_upgrade.sql` Dateien in den Installationsordnern finden. Sie können sie alle mit auflisten:

```
ls `pg_config --sharedir`/contrib/postgis-3.6.0dev/*_upgrade.sql
```

Laden Sie sie alle nacheinander, beginnend mit `postgis_upgrade.sql`.

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

Das gleiche Verfahren gilt für Raster-, Topologie- und `sfcgal`-Erweiterungen, mit Upgrade-Dateien namens `rtpostgis_upgrade.sql`, `topology_upgrade.sql` und `sfcgal_upgrade.sql`. Falls Sie sie benötigen:

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

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

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

Wir empfehlen Ihnen, zu einer erweiterungsbasierten Installation zu wechseln, indem Sie

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

**Note**

Wenn Sie die `postgis_upgrade.sql` speziell für das Upgrade Ihrer Version nicht finden können, verwenden Sie eine Version, die zu früh für ein Soft-Upgrade ist und müssen ein **HARD UPGRADE** durchführen.

Die Funktion `PostGIS_Full_Version` sollte Sie mit der Meldung "procs need upgrade" über die Notwendigkeit eines solchen Upgrades informieren.

### 3.4.2 Hartes Upgrade

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

Der Dump/Reload-Prozess wird durch das `postgis_restore`-Skript unterstützt, das dafür sorgt, dass alle Definitionen, die zu PostGIS gehören (einschließlich der alten), aus dem Dump ausgelassen werden. So können Sie Ihre Schemata und Daten in einer Datenbank mit installiertem PostGIS wiederherstellen, ohne dass es zu Fehlern bei doppelten Symbolen kommt oder veraltete Objekte mitgenommen werden.

Ergänzende Anleitungen für Windows-Benutzer finden Sie unter **Windows Hard Upgrade**.

Das Verfahren ist wie folgt:

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

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

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

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

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

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

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

In den folgenden Fällen kann es zu Fehlern kommen:

1. Einige Ihrer Ansichten oder Funktionen verwenden veraltete PostGIS-Objekte. Um dies zu beheben, können Sie versuchen, das Skript `legacy.sql` vor der Wiederherstellung zu laden, oder Sie müssen eine Version von PostGIS wiederherstellen, die diese Objekte noch enthält, und nach der Portierung Ihres Codes erneut eine Migration versuchen. Wenn der `legacy.sql` Weg für Sie funktioniert, vergessen Sie nicht, Ihren Code so zu korrigieren, dass er keine veralteten Funktionen mehr verwendet und sie durch Laden von `uninstall_legacy.sql` fallen lässt.
2. Einige benutzerdefinierte Datensätze von `spatial_ref_sys` in der Dump-Datei haben einen ungültigen SRID-Wert. Gültige SRID-Werte sind größer als 0 und kleiner als 999000. Werte im Bereich 999000.999999 sind für den internen Gebrauch reserviert, während Werte  $> 999999$  überhaupt nicht verwendet werden können. Alle benutzerdefinierten Datensätze mit ungültigen SRIDs werden beibehalten, wobei die Werte  $> 999999$  in den reservierten Bereich verschoben werden, aber die Tabelle `spatial_ref_sys` würde eine Prüfbeschränkung verlieren, die dafür sorgt, dass diese Invariante beibehalten wird, und möglicherweise auch ihren Primärschlüssel (wenn mehrere ungültige SRIDs in denselben reservierten SRID-Wert umgewandelt werden).

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

```
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);
```

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

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

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

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

führen Sie diesen Befehl aus:

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

Erstellen Sie dann Ihre neue Datenbank, aktivieren Sie die erforderlichen Postgis-Erweiterungen und fügen Sie die französische System-IGN mit **diesem Skript** ein, um Ihre Daten zu importieren:

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

## Chapter 4

# Datenverwaltung

### 4.1 Räumliches Datenmodell

#### 4.1.1 OGC-Geometrie

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

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

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

Koordinaten können optionale Z- und M-Ordinatenwerte enthalten. Die Z-Ordinate wird häufig zur Darstellung der Höhe verwendet. Die M-Ordinate enthält einen Messwert, der Zeit oder Entfernung darstellen kann. Wenn Z- oder M-Werte in einem Geometriewert vorhanden sind, müssen sie für jeden Punkt in der Geometrie definiert werden. Wenn eine Geometrie Z- oder M-Ordinaten hat, ist die **Koordinaten-Dimension** 3D; wenn sie sowohl Z- als auch M-Werte hat, ist die Koordinaten-Dimension 4D.

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

Die Geometrie **Dimension** ist eine Eigenschaft von Geometriotypen. Punkttypen haben die Dimension 0, lineare Typen haben die Dimension 1, und polygonale Typen haben die Dimension 2. Sammlungen haben die Dimension der maximalen Elementdimension.

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

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

Das Geometriemodell ermöglicht die Auswertung topologischer räumlicher Beziehungen, wie in Section 5.1.1 beschrieben. Um dies zu unterstützen, werden für jeden Geometrietyt die Konzepte **interior**, **boundary** und **exterior** definiert. Geometrien sind topologisch geschlossen, sie enthalten also immer ihren Rand. Der Rand ist eine Geometrie der Dimension eins weniger als die der Geometrie selbst.

Das OGC-Geometriemodell definiert Gültigkeitsregeln für jeden Geometrietyt. Diese Regeln stellen sicher, dass die Geometriewerte realistische Situationen darstellen (z.B. ist es möglich, ein Polygon mit einem Loch außerhalb der Schale zu spezifizieren, was aber geometrisch keinen Sinn ergibt und daher ungültig ist). PostGIS erlaubt auch die Speicherung und Bearbeitung von ungültigen Geometriewerten. Dies ermöglicht es, sie zu erkennen und bei Bedarf zu korrigieren. Siehe Section 4.4

#### 4.1.1.1 Punkt

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

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

#### 4.1.1.2 Linie

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

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

#### 4.1.1.3 LinearRing

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

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

#### 4.1.1.4 Polygon

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

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

#### 4.1.1.5 MultiPoint

Ein MultiPoint ist eine Sammlung von Punkten.

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

#### 4.1.1.6 MultiLineString

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

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



#### 4.1.1.7 MultiPolygon

Ein MultiPolygon ist eine Sammlung von nicht überlappenden, nicht benachbarten Polygonen. Die Polygone der Sammlung dürfen sich nur an einer endlichen Anzahl von Punkten berühren.

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

#### 4.1.1.8 GeometryCollection

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

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

#### 4.1.1.9 PolyhedralSurface

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

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

#### 4.1.1.10 Dreieck

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

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

#### 4.1.1.11 TIN

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

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

### 4.1.2 SQL/MM Teil 3 - Kurven

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



#### Note

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

#### 4.1.2.1 CircularString

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

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

#### 4.1.2.2 CompoundCurve

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

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

#### 4.1.2.3 KurvenPolygon

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

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

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

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

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

#### 4.1.2.4 MultiCurve

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

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

#### 4.1.2.5 MultiSurface

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

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



## 4.2 Geometrie Datentyp

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

Der `Geometry` Datentyp ist *opaque*, was bedeutet, dass der gesamte Zugriff über den Aufruf von Funktionen auf Geometriewerte erfolgt. Funktionen ermöglichen die Erstellung von Geometrieobjekten, den Zugriff auf alle internen Felder oder deren Aktualisierung sowie die Berechnung neuer Geometriewerte. PostGIS unterstützt alle Funktionen, die in der OGC *Simple feature access - Part 2: SQL option* (SFS) Spezifikation spezifiziert sind, sowie viele andere. Siehe [Chapter 7](#) für die vollständige Liste der Funktionen.



### Note

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

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

Um die Abfrage von Geometrien effizient zu gestalten, definiert PostGIS verschiedene Arten von räumlichen Indizes und räumliche Operatoren, um diese zu verwenden. Siehe [Section 4.9](#) und [Section 5.2](#) für Details.

### 4.2.1 PostGIS EWKB und EWKT

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

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

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

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

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

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

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

- Bei 4D-Geometrien entfällt der ZM-Bezeichner:

OGC: PUNKT ZM (1 2 3 4)

EWKT: PUNKT (1 2 3 4)

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

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



#### Caution

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

Beispiele für die EWKT-Textdarstellung von räumlichen Objekten sind:

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

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

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

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

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

## 4.3 Geographie Datentyp

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

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

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

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

Wie der Datentyp Geometrie sind auch die Geodaten über einen Spatial Reference System Identifier (SRID) mit einem räumlichen Bezugssystem verbunden. Jedes in der Tabelle `spatial_ref_sys` definierte geodätische (long/lat-basierte) Raumbezugssystem kann verwendet werden. (Vor PostGIS 2.2 unterstützte der Geografietypp nur das geodätische WGS 84 (SRID:4326)). Sie können Ihr eigenes geodätisches Raumbezugssystem hinzufügen, wie in Section 4.5.2 beschrieben.

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

### 4.3.1 Erstellen von Geografietabellen

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

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

Der Geografietypp unterstützt zwei optionale Typmodifikatoren:

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

Beispiele für die Erstellung von Tabellen mit geografischen Spalten:

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

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

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

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

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

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

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

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

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

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

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

```
SELECT * FROM geography_columns;
```

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

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

### 4.3.2 Verwendung von Geografietabellen

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

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

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

```
-- NAD 83 lon/lat
SELECT 'SRID=4269;POINT(-123 34)::geography;
        geography
-----
0101000020AD10000000000000000000C05EC000000000000004140
```

```
-- NAD27 lon/lat
SELECT 'SRID=4267;POINT(-123 34)::geography;
        geography
-----
0101000020AB10000000000000000000C05EC000000000000004140
```

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

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

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

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

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

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

```
-- Distance calculation using GEOGRAPHY
SELECT ST_Distance('LINESTRING(-122.33 47.606, 0.0 51.5) '::geography, 'POINT(-21.96 64.15) ←
  '::geography);
  st_distance
-----
122235.23815667
```

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

```
-- Distance calculation using GEOMETRY
SELECT ST_Distance('LINESTRING(-122.33 47.606, 0.0 51.5) '::geometry, 'POINT(-21.96 64.15) ←
  '::geometry);
  st_distance
-----
13.342271221453624
```

### 4.3.3 Wann wird der Datentyp Geografie verwendet?

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

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

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

Für einen Vergleich, welche Funktionalität von Geography vs. Geometry unterstützt wird, siehe Section [13.11](#). Für eine kurze Liste mit der Beschreibung der geographischen Funktionen, siehe Section [13.4](#)



### 4.3.4 Fortgeschrittene FAQ's zum geographischen Datentyp

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

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

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

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

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

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

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

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

## 4.4 Geometrieverifizierung

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

### 4.4.1 Einfache Geometrie

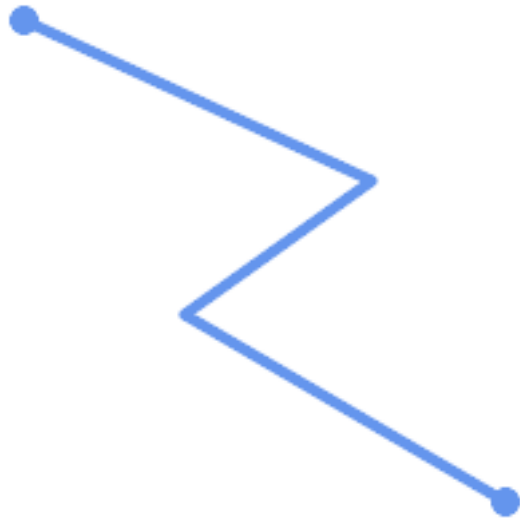
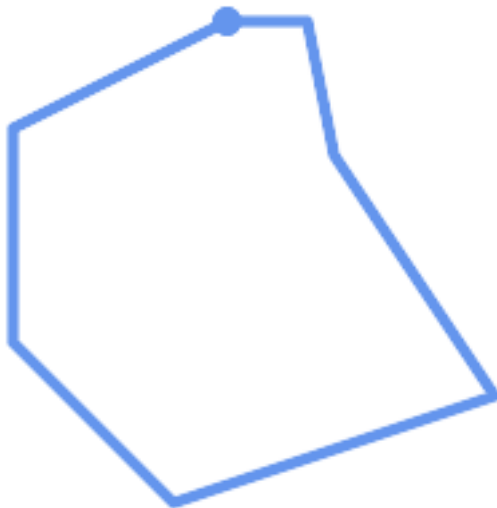
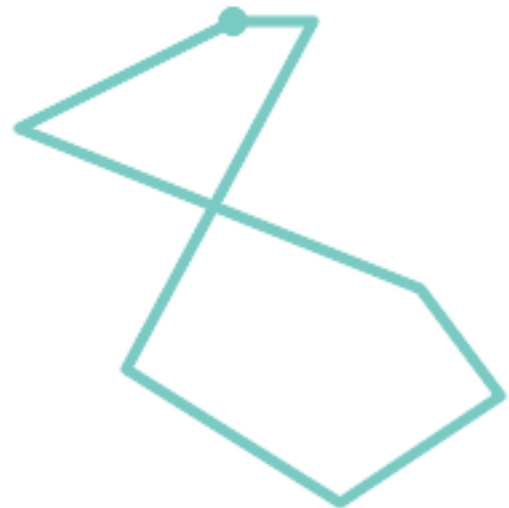
Eine *einfache* Geometrie ist eine Geometrie, die keine anomalen geometrischen Punkte wie Selbstschnittpunkte oder Selbsttangente aufweist.

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

`MULTIPOINTS` sind *simple*, wenn sich keine zwei Koordinaten (`POINTS`) decken (keine identischen Koordinatenpaare aufweisen).

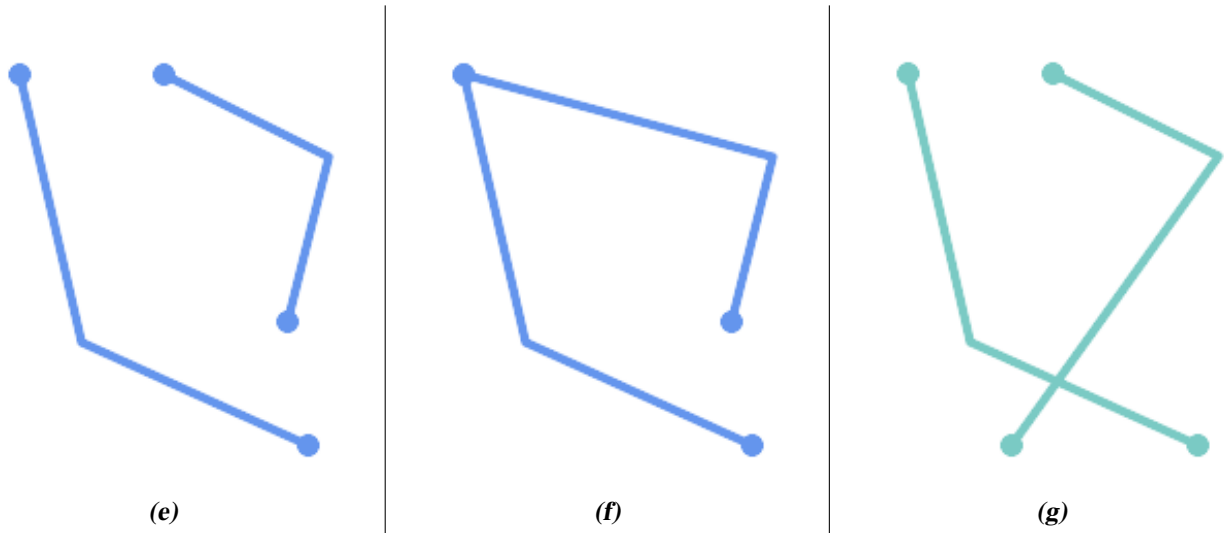
Ein `LINestring` ist *einfach*, wenn er, abgesehen von den Endpunkten, nicht zweimal durch denselben Punkt verläuft. Wenn die Endpunkte eines einfachen Linienstrangs identisch sind, wird er *geschlossen* und als linearer Ring bezeichnet.

*(a) und (c) sind einfache LINESTRINGS. (b) und (d) sind nicht einfach. (c) ist ein geschlossener linearer Ring.*

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

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

*(e) und (f) sind einfache MULTILINESTRINGS. (g) ist nicht einfach.*



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

Um zu prüfen, ob eine Geometrie einfach ist, verwenden Sie die Funktion **ST\_IsSimple**:

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

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

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

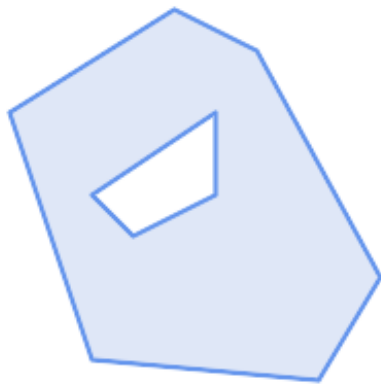
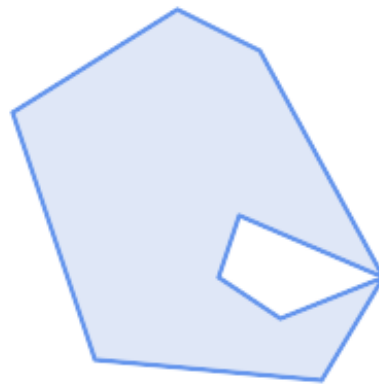
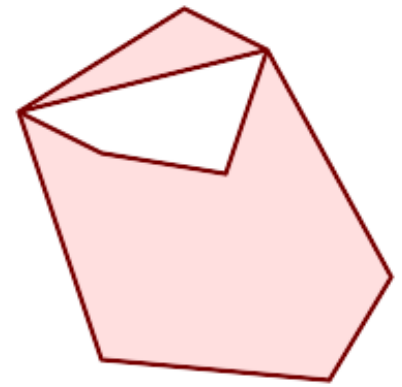
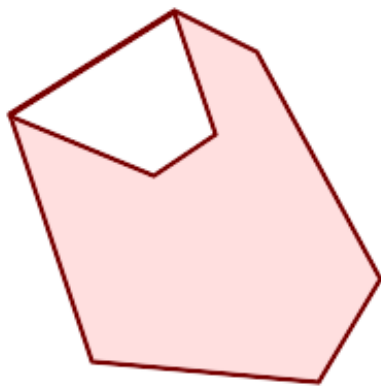
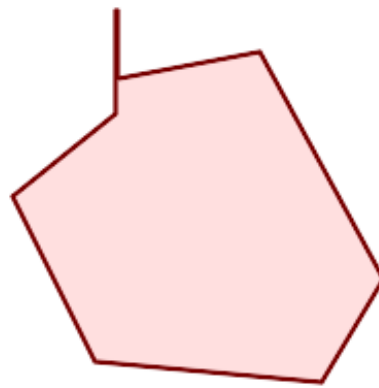
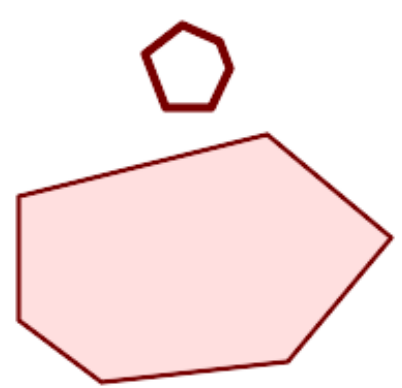
#### 4.4.2 Gültige Geometrie

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

Ein POLYGON ist *gültig* wenn:

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

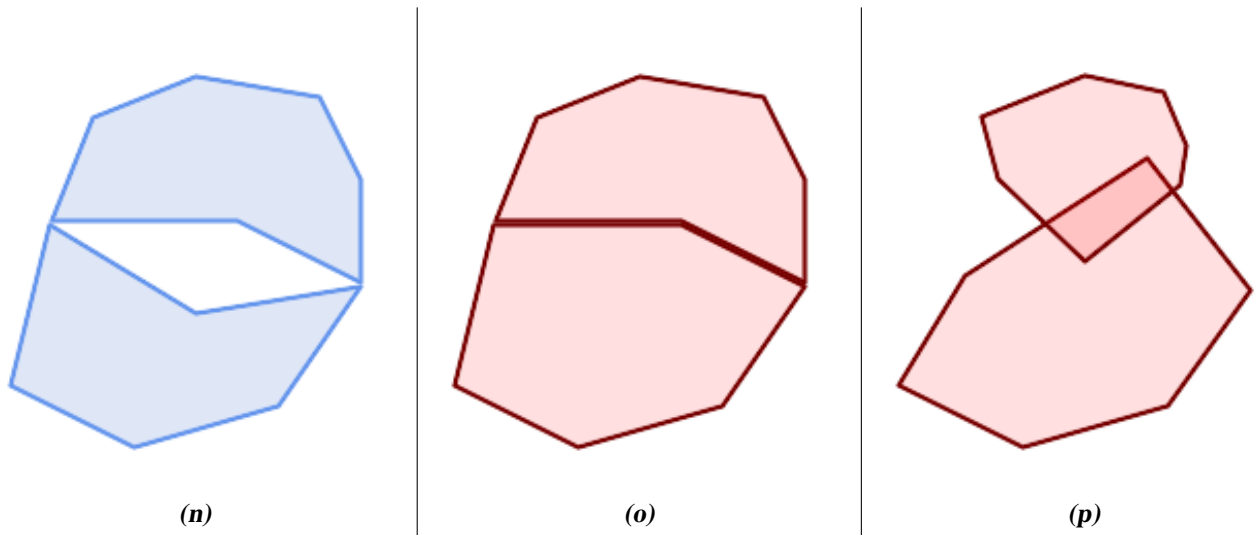
**(h)** und **(i)** sind gültig POLYGONS. **(j-m)** sind ungültig. **(j)** kann als gültiges MULTIPOLYGON dargestellt werden.

**(h)****(i)****(j)****(k)****(l)****(m)**

Ein MULTIPOLYGON ist *gültig* wenn:

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

**(n)** ist ein gültiges MULTIPOLYGON. **(o)** und **(p)** sind ungültig.



Diese Regeln bedeuten, dass gültige polygonale Geometrie auch *einfach* ist.

Für die lineare Geometrie ist die einzige Gültigkeitsregel, dass `LINESTRING`s mindestens zwei Punkte und eine Länge ungleich Null haben muss (oder äquivalent dazu mindestens zwei verschiedene Punkte.) Beachten Sie, dass nicht-einfache (sich selbst schneidende) Linien gültig sind.

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

len_nonzero	len_zero	self_int
t	f	t

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

#### 4.4.3 Verwaltung der Gültigkeit

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

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

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



#### Note

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

Um zu prüfen, ob eine Geometrie gültig ist, verwenden Sie die Funktion **ST\_IsValid**:

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

Informationen über die Art und den Ort einer geometrischen Ungültigkeit werden von der Funktion **ST\_IsValidDetail** geliefert:

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

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

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

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

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

## 4.5 Räumliche Bezugssysteme

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

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

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

Eine Geometrie ist mit einem räumlichen Bezugssystem durch ihren SRID-Wert verbunden, auf den über **ST\_SRID** zugegriffen wird. Der SRID für eine Geometrie kann mit **ST\_SetSRID** zugewiesen werden. Einige Geometrieconstructorfunktionen ermöglichen die Angabe eines SRID (z. B. **ST\_Point** und **ST\_MakeEnvelope**). Das Format **EWKT** unterstützt SRIDs mit dem Präfix **SRID=n;**.

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

### 4.5.1 SPATIAL\_REF\_SYS Tabelle

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

Die Definition der Tabelle `spatial_ref_sys` lautet:

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

Die Spalten sind:

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

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

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

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

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

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

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

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

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

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

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

## 4.5.2 Benutzerdefinierte räumliche Bezugssysteme

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

Eine Ressource zum Auffinden von nicht im Kernsatz definierten räumlichen Bezugssystemen ist <http://spatialreference.org/>

Einige häufig verwendete Raumbezugssysteme sind: **4326 - WGS 84 Long Lat**, **4269 - NAD 83 Long Lat**, **3395 - WGS 84 World Mercator**, **2163 - US National Atlas Equal Area**, und die 60 WGS84 UTM-Zonen. UTM-Zonen sind mit am besten für Messungen geeignet, decken aber nur 6-Grad-Regionen ab. (Um zu bestimmen, welche UTM-Zone für Ihr Gebiet von Interesse zu verwenden ist, siehe **utmzone PostGIS plpgsql helper function**).

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

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

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

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

## 4.6 Räumliche Tabellen

### 4.6.1 Erstellung einer räumlichen Tabelle

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

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

Der Geometrietyp unterstützt zwei optionale **Typmodifikatoren**:

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



Beispiele für die Erstellung von Tabellen mit Geometriespalten:

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

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

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

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

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

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

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

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

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

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

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

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

## 4.6.2 GEOMETRY\_COLUMNS Ansicht

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

```
\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 Spalten sind:

**f\_table\_catalog**, **f\_table\_schema**, **f\_table\_name** Der voll qualifizierte Name der Merkmalstabelle, die die Geometriespalte enthält. Es gibt kein PostgreSQL-Analogon für "catalog", daher wird diese Spalte leer gelassen. Für "schema" wird der Name des PostgreSQL-Schemas verwendet (`public` ist der Standard).

**f\_geometry\_column** Der Name der Geometriespalte in der Feature-Tabelle.

**coord\_dimension** Die Koordinatenabmessung (2, 3 oder 4) der Spalte.

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

**type** Der Datentyp des Geobjekts. 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.

### 4.6.3 Manuelles Registrieren von Geometriespalten

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

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

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

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

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

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

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

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

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

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

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

In psql:

```
\d pois_ny;
```

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

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

Beide registrieren sich korrekt in "geometry\_columns"

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

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

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

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

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

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

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

Dies kann sich in zukünftigen Versionen von PostGIS ändern, aber im Moment müssen Sie dies tun, um die korrekte Registrierung der einschränkungs-basierten Ansichtsspalte zu erzwingen:

```

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

```

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

## 4.7 Laden von Geodaten

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

### 4.7.1 SQL zum Laden von Daten verwenden

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

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

```

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

```

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

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

### 4.7.2 Verwendung des Shapefile Loaders

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

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

**(claldp) Dies sind sich gegenseitig ausschließende Optionen:**

- c Erzeugt eine neue Tabelle und füllt sie aus dem Shapefile. *Dies ist der Standardmodus.*
- a Fügt Daten aus dem Shapefile in die Datenbanktabelle ein. Beachten Sie, dass die Dateien die gleichen Attribute und Datentypen haben müssen, um mit dieser Option mehrere Dateien zu laden.
- d Löscht die Datenbanktabelle, bevor eine neue Tabelle mit den Daten im Shapefile erstellt wird.
- p Erzeugt nur den SQL-Code zur Erstellung der Tabelle, ohne irgendwelche Daten hinzuzufügen. Kann verwendet werden, um die Erstellung und das Laden einer Tabelle vollständig zu trennen.
- ? Zeigt die Hilfe an.
- D Verwendung des PostgreSQL "dump" Formats für die Datenausgabe. Kann mit -a, -c und -d kombiniert werden. Ist wesentlich schneller als das standardmäßige SQL "insert" Format. Verwenden Sie diese Option wenn Sie sehr große Datensätze haben.
- s [**<FROM\_SRID>** : ] **<SRID>** Erstellt und befüllt die Geometrietabelle mit der angegebenen SRID. Optional kann für das Shapefile eine FROM\_SRID angegeben werden, worauf dann die Geometrie in die Ziel-SRID projiziert wird.
- k Erhält die Groß- und Kleinschreibung (Spalte, Schema und Attribute). Beachten Sie bitte, dass die Attributnamen in Shape-dateien immer Großbuchstaben haben.
- i Wandeln Sie alle Ganzzahlen in standard 32-bit Integer um, erzeugen Sie keine 64-bit BigInteger, auch nicht dann wenn der DBF-Header dies unterstellt.
- I Einen GIST Index auf die Geometriespalte anlegen.
- m -m *a\_file\_name* bestimmt eine Datei, in welcher die Abbildungen der (langen) Spaltennamen in die 10 Zeichen langen DBF Spaltennamen festgelegt sind. Der Inhalt der Datei besteht aus einer oder mehreren Zeilen die jeweils zwei, durch Leerzeichen getrennte Namen enthalten, aber weder vorne noch hinten mit Leerzeichen versehen werden dürfen. Zum Beispiel:
 

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

- n -n Es wird nur die \*.dbf-Datei importiert. Wenn das Shapefile nicht Ihren Daten entspricht, wird automatisch auf diesen Modus geschaltet und nur die \*.dbf-Datei geladen. Daher müssen Sie diese Flag nur dann setzen, wenn sie einen vollständigen Shapefile-Satz haben und lediglich die Attributdaten, und nicht die Geometrie, laden wollen.
- G Verwendung des geographischen Datentyps in WGS84 (SRID=4326), anstelle des geometrischen Datentyps (benötigt Längen- und Breitenangaben).
- T **<tablespace>** Den Tablespace für die neue Tabelle festlegen. Solange der -X Parameter nicht angegeben wird, benutzen die Indizes weiterhin den standardmäßig festgelegten Tablespace. Die PostgreSQL Dokumentation beinhaltet eine gute Beschreibung, wann es sinnvoll ist, eigene Tablespaces zu verwenden.
- X **<tablespace>** Den Tablespace bestimmen, in dem die neuen Tabellenindizes angelegt werden sollen. Gilt für den Primärschlüsselindex und wenn "-l" verwendet wird, auch für den räumlichen GIST-Index.
- Z Wenn dieses Flag verwendet wird, verhindert es die Erzeugung von ANALYZE Anweisungen. Ohne das Flag -Z (Standardverhalten) werden die Anweisungen ANALYZE erzeugt.

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

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

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

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

## 4.8 Extrahieren von Geodaten

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

### 4.8.1 SQL zum Extrahieren von Daten verwenden

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

```
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)
```

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

**ST\_Intersects** Diese Funktion bestimmt ob sich zwei geometrische Objekte einen gemeinsamen Raum teilen

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

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

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

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

Überprüfung ob einige der Strassen in die Polygonfläche hineinreichen:

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

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

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

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

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

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

## 4.8.2 Verwendung des Shapefile-Dumpers

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

```
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** Die Verwendung eines binären Cursors macht die Berechnung schneller; funktioniert aber nur, wenn alle nicht-geometrischen Attribute in den Datentyp "text" umgewandelt werden können.
- r** RAW-Modus. Das Attribut `gid` wird nicht verworfen und Spaltennamen werden nicht maskiert.
- m filename** Bildet die Identifikatoren in Namen mit 10 Zeichen ab. Der Inhalt der Datei besteht aus Zeilen von jeweils zwei durch Leerzeichen getrennten Symbolen, jedoch ohne vor- oder nachgestellte Leerzeichen: `VERYLONGSYMBOL SHORTONE ANOTHERVERYLONGSYMBOL SHORTER` etc.

## 4.9 Räumliche Indizes

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

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

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

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

Siehe auch den [PostGIS Workshop Abschnitt über räumliche Indizes](#), und das [PostgreSQL Handbuch](#).

### 4.9.1 GiST-Indizes

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

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

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

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



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

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

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

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

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

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

## 4.9.2 BRIN Indizes

BRIN steht für "Block Range Index". Es handelt sich um eine allgemeine Indexmethode, die in PostgreSQL 9.5 eingeführt wurde. BRIN ist eine *lossy* Indexmethode, was bedeutet, dass eine sekundäre Prüfung erforderlich ist, um zu bestätigen, dass ein Datensatz mit einer bestimmten Suchbedingung übereinstimmt (was für alle bereitgestellten räumlichen Indizes der Fall ist). Sie ermöglicht eine viel schnellere Indexerstellung und eine viel geringere Indexgröße bei einer angemessenen Leseleistung. Sein Hauptzweck besteht darin, die Indizierung sehr großer Tabellen auf Spalten zu unterstützen, die eine Korrelation mit ihrer physischen Position innerhalb der Tabelle aufweisen. Zusätzlich zur räumlichen Indizierung kann BRIN die Suche auf verschiedenen Arten von Attributdatenstrukturen (Integer, Arrays usw.) beschleunigen. Weitere Informationen finden Sie im [PostgreSQL-Handbuch](#).

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Zusammenfassung der Verwendung von BRIN für räumliche Daten:

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

### 4.9.3 SP-GiST Indizes

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

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

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

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

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

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

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

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

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

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

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

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

kNN Suche wird zurzeit nicht unterstützt.

### 4.9.4 Abstimmung der Indexverwendung

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

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

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

Sie sollten Ihre Datenbanken auf jeden Fall regelmäßig leeren. Viele PostgreSQL-DBAs lassen VACUUM regelmäßig als Cron-Job außerhalb der Spitzenzeiten laufen.

- Wenn das Vakuumieren nicht hilft, können Sie den Planer vorübergehend zwingen, die Indexinformationen zu verwenden, indem Sie den Befehl **SET ENABLE\_SEQSCAN TO OFF**; verwenden. Auf diese Weise können Sie überprüfen, ob der Planer überhaupt in der Lage ist, einen indexbeschleunigten Abfrageplan für Ihre Abfrage zu erstellen. Sie sollten diesen Befehl nur zur Fehlersuche verwenden; im Allgemeinen weiß der Planer besser als Sie, wann Indizes verwendet werden sollten. Nachdem Sie Ihre Abfrage ausgeführt haben, vergessen Sie nicht, **SET ENABLE\_SEQSCAN TO ON**; auszuführen, damit der Planer bei anderen Abfragen normal arbeitet.
- Wenn **SET ENABLE\_SEQSCAN TO OFF**; Ihre Abfrage schneller laufen lässt, ist Ihr Postgres wahrscheinlich nicht auf Ihre Hardware abgestimmt. Wenn Sie feststellen, dass der Planer die Kosten für sequentielle und Index-Scans falsch einschätzt, versuchen Sie, den Wert von `RANDOM_PAGE_COST` in `postgresql.conf` zu reduzieren, oder verwenden Sie **SET RANDOM\_PAGE\_COST TO 1.1**;. Der Standardwert für `RANDOM_PAGE_COST` ist 4.0. Versuchen Sie, ihn auf 1,1 (für SSD) oder 2,0 (für schnelle Magnetplatten) zu setzen. Wenn Sie den Wert verringern, wird der Planer mit größerer Wahrscheinlichkeit Index-Scans verwenden.
- Wenn **SET ENABLE\_SEQSCAN TO OFF**; Ihrer Abfrage nicht hilft, verwendet die Abfrage möglicherweise ein SQL-Konstrukt, das der Postgres-Planer noch nicht optimieren kann. Es kann möglich sein, die Abfrage so umzuschreiben, dass der Planer sie verarbeiten kann. Zum Beispiel kann eine Subquery mit einem Inline-SELECT keinen effizienten Plan erzeugen, kann aber möglicherweise mit einem LATERAL JOIN umgeschrieben werden.

Weitere Informationen finden Sie im Postgres-Handbuch im Abschnitt [Query Planning](#).

## Chapter 5

# Räumliche Abfrage

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

### 5.1 Räumliche Beziehungen feststellen

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

#### 5.1.1 Dimensionell erweitertes 9-Schnitte-Modell

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

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

##### Grenze

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

##### Innenbereich

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

##### Äußeres

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

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

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


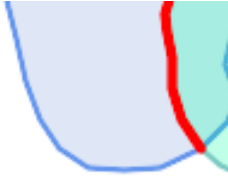

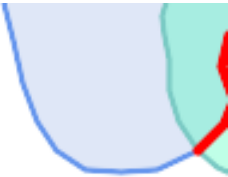
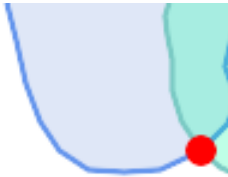
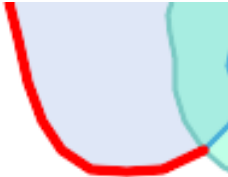
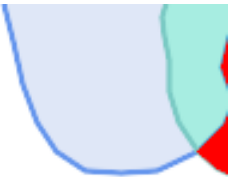
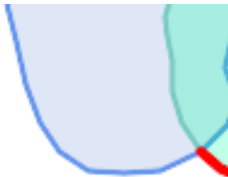
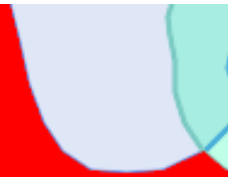
- 0 => Punkt
- 1 => Zeile
- 2 => Fläche
- F => leere Menge

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

	<b>Innenbereich</b>	<b>Grenze</b>	<b>Äußeres</b>
<b>Innenbereich</b>	$\dim( I(a) \cap I(b) )$	$\dim( I(a) \cap B(b) )$	$\dim( I(a) \cap E(b) )$
<b>Grenze</b>	$\dim( B(a) \cap I(b) )$	$\dim( B(a) \cap B(b) )$	$\dim( B(a) \cap E(b) )$
<b>Äußeres</b>	$\dim( E(a) \cap I(b) )$	$\dim( E(a) \cap B(b) )$	$\dim( E(a) \cap E(b) )$

Für zwei sich überschneidende polygonale Geometrien sieht dies folgendermaßen aus:



	Innenbereich	Grenze	Äußeres
Innenbereich	 $dim( I(a) \cap I(b) ) = 2$	 $dim( I(a) \cap B(b) ) = 1$	 $dim( I(a) \cap E(b) ) = 2$
Grenze	 $dim( B(a) \cap I(b) ) = 1$	 $dim( B(a) \cap B(b) ) = 0$	 $dim( B(a) \cap E(b) ) = 1$
Äußeres	 $dim( E(a) \cap I(b) ) = 2$	 $dim( E(a) \cap B(b) ) = 1$	 $dim( E(a) \cap E(b) ) = 2$

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

Weitere Informationen finden Sie unter:

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

### 5.1.2 Benannte räumliche Beziehungen

Um die Bestimmung allgemeiner räumlicher Beziehungen zu erleichtern, definiert das OGC SFS eine Reihe von *genannten räumlichen Beziehungsprädikaten*. PostGIS stellt diese als die Funktionen [ST\\_Contains](#), [ST\\_Crosses](#), [ST\\_Disjoint](#), [ST\\_Equals](#), [ST\\_Intersects](#), [ST\\_Overlaps](#), [ST\\_Touches](#), [ST\\_Within](#) zur Verfügung. Es definiert auch die Nicht-Standard-Beziehungsprädikate [ST\\_Covers](#), [ST\\_CoveredBy](#) und [ST\\_ContainsProperly](#).

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

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

Weitere Einzelheiten und Abbildungen finden Sie im [PostGIS Workshop](#).

### 5.1.3 Allgemeine räumliche Beziehungen

In manchen Fällen reichen die genannten räumlichen Beziehungen nicht aus, um die gewünschten räumlichen Filterbedingungen zu schaffen.

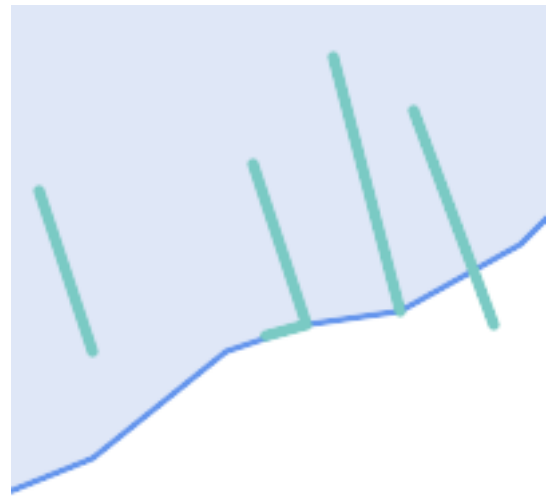


Nehmen wir zum Beispiel einen linearen Datensatz, der ein Straßennetz darstellt. Es kann erforderlich sein, alle Straßenabschnitte zu identifizieren, die sich nicht in einem Punkt, sondern in einer Linie kreuzen (vielleicht um eine Geschäftsregel zu validieren). In diesem Fall bietet `ST_Crosses` nicht den erforderlichen räumlichen Filter, da es für lineare Merkmale `true` nur dann zurückgibt, wenn sie sich in einem Punkt kreuzen.

Eine zweistufige Lösung würde darin bestehen, zunächst den tatsächlichen Schnittpunkt (`ST_Intersection`) von Paaren von Straßenlinien zu berechnen, die sich räumlich schneiden (`ST_Intersects`), und dann zu prüfen, ob der `ST_GeometryType` des Schnittpunkts 'LINESTRING' ist (wobei Fälle, die `GEOMETRYCOLLECTIONS` von `[MULTI] POINTS`, `[MULTI] LINESTRINGS` usw. zurückgeben, korrekt behandelt werden).

Es ist klar, dass eine einfachere und schnellere Lösung wünschenswert ist.





Ein zweites Beispiel ist das Auffinden von Anlegestellen, die die Grenze eines Sees auf einer Linie schneiden und bei denen ein Ende der Anlegestelle auf dem Ufer liegt. Mit anderen Worten, wenn ein Kai innerhalb eines Sees liegt, aber nicht vollständig von diesem umschlossen ist, die Grenze eines Sees auf einer Linie schneidet und genau einer der Endpunkte des Kais innerhalb oder auf der Grenze des Sees liegt. Es ist möglich, eine Kombination von räumlichen Prädikaten zu verwenden, um die gewünschten Merkmale zu finden:

- `ST_Contains`(See, Kai) = TRUE
  - `ST_ContainsProperly`(See, Kai) = FALSE
  - `ST_GeometryType(ST_Intersection(Kai, See)) = 'LINESTRING'`
  - `ST_NumGeometries(ST_Multi(ST_Intersection(ST_Boundary(Kai), ST_Boundary(See)))) = 1`
- ... aber das ist natürlich ziemlich kompliziert.

Diese Anforderungen können erfüllt werden, indem die vollständige DE-9IM-Schnittmatrixmatrix berechnet wird. PostGIS bietet hierfür die Funktion `ST_Relate`:

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

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

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

Mit Hilfe von Kreuzungsmatrixmustern können spezifische räumliche Beziehungen auf eine prägnantere Weise bewertet werden. Die Funktionen `ST_Relate` und `ST_RelateMatch` können zum Testen von Kreuzungsmatrixmustern verwendet werden. Für das erste obige Beispiel lautet das Schnittmatrixmuster, das zwei Linien angibt, die sich in einer Linie schneiden, `'1*1***1**'`:

```
-- Find road segments that intersect in a line
SELECT a.id
FROM roads a, roads b
WHERE a.id != b.id
```

```
AND a.geom && b.geom
AND ST_Relate(a.geom, b.geom, '1*1***1**');
```

Für das zweite Beispiel lautet das Schnittmatrixmuster, das eine Linie teilweise innerhalb und teilweise außerhalb eines Polygons angibt, **'102101FF2'**:

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

## 5.2 Räumliche Indizes verwenden

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

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

Indexgestützte Funktionen fügen der räumlichen Bedingung automatisch einen Begrenzungsrahmenoperator hinzu. Zu den indexbasierten Funktionen gehören die benannten räumlichen Beziehungsprädikate **ST\_Contains**, **ST\_ContainsProperly**, **ST\_CoveredBy**, **ST\_Covers**, **ST\_Crosses**, **ST\_Intersects**, **ST\_Overlaps**, **ST\_Touches**, **ST\_Within**, **ST\_Within** und **ST\_3DIntersects** sowie die Entfernungsprädikate **ST\_DWithin**, **ST\_DFullyWithin**, **ST\_3DDFullyWithin** und **ST\_3DDWithin**.

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

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

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

Die Anzahl der zu verarbeitenden Zeilen kann durch die Verwendung der indexbasierten Funktion **ST\_DWithin** erheblich reduziert werden:

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

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

Weitere Informationen und Beispiele finden Sie im [PostGIS Workshop](#).

## 5.3 Beispiele für Spatial SQL

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

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

Die Definition der Tabelle `bc_municipality` lautet:

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

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

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

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

km_roads
-----
70842.1243039643
```

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

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

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

hectares
-----
32657.9103824927
```

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

kilometers
-----
4.89151904172838
```

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

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

gid	name	area
12	SPALLUMCHEEN	257374619.430216

## Chapter 6

# Performance Tipps

### 6.1 Kleine Tabellen mit großen Geometrien

#### 6.1.1 Problembeschreibung

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

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

Stellen Sie sich nun eine Abfrage vor, die den geometrischen Operator `&&` verwendet, um ein Umgebungsrechteck mit nur wenigen Zeilen zu ermitteln. Der Abfrageoptimierer stellt fest, dass die Tabelle nur 3 Speicherseiten und 80 Zeilen aufweist. Er nimmt an, dass ein sequentieller Scan bei einer derart kleinen Tabelle wesentlich schneller abläuft als die Verwendung eines Indexes. 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>

#### 6.1.2 Umgehungslösung

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

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

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

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

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

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

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

## 6.2 CLUSTER auf die geometrischen Indizes

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

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

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

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

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

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

## 6.3 Vermeidung von Dimensionsumrechnungen

Manchmal kann es vorkommen, das Sie 3D- oder 4D-Daten in Ihrer Tabelle haben, aber immer mit den OpenGIS compliant ST\_AsText() oder ST\_AsBinary() Funktionen, die lediglich 2D Geometrien ausgeben, zugreifen. Dies geschieht indem intern die ST\_Force2D() Funktion aufgerufen wird, welche einen wesentlichen Overhead für große Geometrien aufweist. Um diesen Overhead zu vermeiden kann es praktikabel sein diese zusätzlichen Dimensionen ein für alle mal im Voraus zu löschen:

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

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

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

## Chapter 7

# Referenz PostGIS

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

### Note



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

## 7.1 PostgreSQL und PostGIS Datentypen - Geometry/Geography/Box

### 7.1.1 box2d

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

#### Beschreibung

Box3D ist ein geometrischer Datentyp, der den umschreibenden Quader einer oder mehrerer geometrischer Objekte abbildet. ST\_3DExtent gibt ein Box3D-Objekt zurück.

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

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

#### Verhaltensweise bei der Typumwandlung

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

Typumwandlung nach	Verhaltensweise
box3d	implizit
geometry	implizit



**Siehe auch**

Section [13.7](#)

**7.1.2 box3d**

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

**Beschreibung**

Box3D ist ein geometrischer Datentyp, der den umschreibenden Quader einer oder mehrerer geometrischer Objekte abbildet. ST\_3DExtent gibt ein Box3D-Objekt zurück.

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

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

**Verhaltensweise bei der Typumwandlung**

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

Typumwandlung nach	Verhaltensweise
box	implizit
box2d	implizit
geometry	implizit

**Siehe auch**

Section [13.7](#)

**7.1.3 geometry**

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

**Beschreibung**

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

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

**Verhaltensweise bei der Typumwandlung**

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

Typumwandlung nach	Verhaltensweise
box	implizit
box2d	implizit

box3d	implizit
Bytea	implizit
geography	implizit
Text	implizit

**Siehe auch**

Section [4.1](#), Section [4.3](#)

**7.1.4 geometry\_dump**

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

**Beschreibung**

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

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

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

**Siehe auch**

Section [13.6](#)

**7.1.5 geography**

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

**Beschreibung**

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

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

**Verhaltensweise bei der Typumwandlung**

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

Typumwandlung nach	Verhaltensweise
<code>geometry</code>	explizit

**Siehe auch**

Section [4.3](#), Section [4.3](#)

## 7.2 Geometrische Managementfunktionen

### 7.2.1 AddGeometryColumn

AddGeometryColumn — Entfernt eine Geometriespalte aus einer räumlichen Tabelle.

**Synopsis**

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

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

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

**Beschreibung**

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

**Note**

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

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

**Note**

Änderung: 2.0.0 Views können nicht mehr händisch in "geometry\_columns" registriert werden. Views auf eine Geometrie in Typmod-Tabellen, bei denen keine Adapterfunktion verwendet wird, registrieren sich selbst auf korrekte Weise, da sie die Typmod-Verhaltensweise von der Spalte der Stammtabelle erben. Views die 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 [4.6.3](#).



Diese Methode implementiert die [OGC Simple Features Implementation Specification for SQL 1.1](#).



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



Diese Methode unterstützt kreisförmige Strings und Kurven.

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

## Beispiele

```

-- Create schema to hold data
CREATE SCHEMA my_schema;
-- Create a new simple PostgreSQL table
CREATE TABLE my_schema.my_spatial_table (id serial);

-- Describing the table shows a simple table with a single "id" column.
postgis=# \d my_schema.my_spatial_table
                                     Table "my_schema.my_spatial_table"
Column | Type | Modifiers
-----+-----+-----
 id    | integer | not null default nextval('my_schema.my_spatial_table_id_seq'::regclass)

-- Add a spatial column to the table
SELECT AddGeometryColumn ('my_schema','my_spatial_table','geom',4326,'POINT',2);

-- Add a point using the old constraint based behavior
SELECT AddGeometryColumn ('my_schema','my_spatial_table','geom_c',4326,'POINT',2, false);

--Add a curvepolygon using old constraint behavior
SELECT AddGeometryColumn ('my_schema','my_spatial_table','geomcp_c',4326,'CURVEPOLYGON',2, ←
    false);

-- Describe the table again reveals the addition of a new geometry columns.
\d my_schema.my_spatial_table
                                     addgeometrycolumn
-----+-----+-----
my_schema.my_spatial_table.geomcp_c SRID:4326 TYPE:CURVEPOLYGON DIMS:2
(1 row)

                                     Table "my_schema.my_spatial_table"
Column | Type | Modifiers
-----+-----+-----
 id    | integer | not null default nextval('my_schema. ←
    my_spatial_table_id_seq'::regclass)
 geom  | geometry(Point,4326) |
 geom_c | geometry |
 geomcp_c | geometry |
Check constraints:
 "enforce_dims_geom_c" CHECK (st_ndims(geom_c) = 2)
 "enforce_dims_geomcp_c" CHECK (st_ndims(geomcp_c) = 2)
 "enforce_geotype_geom_c" CHECK (geometrytype(geom_c) = 'POINT'::text OR geom_c IS NULL)
 "enforce_geotype_geomcp_c" CHECK (geometrytype(geomcp_c) = 'CURVEPOLYGON'::text OR ←
    geomcp_c IS NULL)
 "enforce_srid_geom_c" CHECK (st_srid(geom_c) = 4326)
 "enforce_srid_geomcp_c" CHECK (st_srid(geomcp_c) = 4326)

-- geometry_columns view also registers the new columns --
SELECT f_geometry_column As col_name, type, srid, coord_dimension As ndims
FROM geometry_columns
WHERE f_table_name = 'my_spatial_table' AND f_table_schema = 'my_schema';

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

```

**Siehe auch**

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

**7.2.2 DropGeometryColumn**

DropGeometryColumn — Entfernt eine Geometriespalte aus einer räumlichen Tabelle.

**Synopsis**

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

**Beschreibung**

Entfernt eine geometrische Spalte aus der Geometrietabelle. Der "schema\_name" muss mit dem Feld "f\_table\_schema" in der Tabelle "geometry\_columns" übereinstimmen.



Diese Methode implementiert die [OGC Simple Features Implementation Specification for SQL 1.1](#).



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



Diese Methode unterstützt kreisförmige Strings und Kurven.

**Note**

Änderung: 2.0.0 Diese Funktion wurde zwecks Abwärtskompatibilität eingeführt. Seit geometry\_columns ein View auf den Systemkatalog ist, können Sie die Geometriespalte, so wie jede andere Tabellenspalte, mit ALTER TABLE löschen.

**Beispiele**

```
SELECT DropGeometryColumn ('my_schema', 'my_spatial_table', 'geom');
      ----RESULT output ----
                        dropgeometrycolumn
-----
my_schema.my_spatial_table.geom effectively removed.

-- In PostGIS 2.0+ the above is also equivalent to the standard
-- the standard alter table. Both will deregister from geometry_columns
ALTER TABLE my_schema.my_spatial_table DROP column geom;
```

**Siehe auch**

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

**7.2.3 DropGeometryTable**

DropGeometryTable — Löscht eine Tabelle und alle Referenzen in dem geometry\_columns View.

## Synopsis

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

## Beschreibung

Löscht eine Tabelle und deren Verweise in "geometry\_columns". Anmerkung: verwendet current\_schema() wenn kein Schema angegeben wird, eine Schema erkennende pgsqll Installation vorausgesetzt.



### Note

Änderung: 2.0.0 Diese Funktion wurde zwecks Abwärtskompatibilität eingeführt. Seit geometry\_columns ein View auf den Systemkatalog ist, können Sie eine Tabelle mit einer Geometriespalte, so wie jede andere Tabelle, mit DROP TABLE löschen.

## Beispiele

```
SELECT DropGeometryTable ('my_schema', 'my_spatial_table');  
----RESULT output ----  
my_schema.my_spatial_table dropped.  
  
-- The above is now equivalent to --  
DROP TABLE my_schema.my_spatial_table;
```

## Siehe auch

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

## 7.2.4 Find\_SRID

Find\_SRID — Gibt die für eine Geometriespalte definierte SRID zurück.

## Synopsis

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

## Beschreibung

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

## Beispiele

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

## Siehe auch

[ST\\_SRID](#)

## 7.2.5 Populate\_Geometry\_Columns

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

### Synopsis

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

### Beschreibung

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

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

- `enforce_dims_the_geom` - stellt sicher, dass jede Geometrie dieselbe Dimension hat (siehe [ST\\_NDims](#))
- `enforce_geotype_the_geom` - stellt sicher, dass jede Geometrie vom selben Datentyp ist (siehe [GeometryType](#))
- `enforce_srid_the_geom` - stellt sicher, dass jede Geometrie die selbe Projektion hat (siehe [ST\\_SRID](#))

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

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

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

Verfügbarkeit: 1.4.0

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

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





rausgesetzt.



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



Diese Methode unterstützt kreisförmige Strings und Kurven.

## Beispiele

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

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

Ändert die SRID der Straßentabelle auf 4326

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

Das vorhergegangene Beispiel ist gleichbedeutend mit 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](#)

## 7.3 Geometrische Konstruktoren

### 7.3.1 ST\_Collect

**ST\_Collect** — Erzeugt eine GeometryCollection oder Multi\*-Geometrie aus einer Reihe von Geometrien.

#### Synopsis

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

## Beschreibung

Sammelt Geometrien in einer Geometriesammlung. Das Ergebnis ist entweder ein Multi\* oder eine GeometryCollection, je nachdem, ob die Eingabegeometrien denselben oder einen unterschiedlichen Typ haben (homogen oder heterogen). Die Eingangsgeometrien bleiben in der Sammlung unverändert.

**Variante 1:** akzeptiert zwei Eingabegeometrien

**Variante 2:** akzeptiert eine Reihe von Geometrien

**Variante 3:** Aggregatfunktion, die ein Rowset von Geometrien akzeptiert.



### Note

Handelt es sich bei einer der Eingabegeometrien um Sammlungen (Multi\* oder GeometryCollection), gibt ST\_Collect eine GeometryCollection zurück (da dies der einzige Typ ist, der verschachtelte Sammlungen enthalten kann). Um dies zu verhindern, verwenden Sie **ST\_Dump** in einer Unterabfrage, um die Eingabekollektionen in ihre atomaren Elemente zu zerlegen (siehe Beispiel unten).



### Note

ST\_Collect und **ST\_Union** scheinen ähnlich zu sein, funktionieren aber in Wirklichkeit ganz anders. ST\_Collect fasst Geometrien in einer Sammlung zusammen, ohne sie in irgendeiner Weise zu verändern. ST\_Union führt Geometrien dort zusammen, wo sie sich überschneiden, und teilt Linienstränge an Schnittpunkten. Es kann einzelne Geometrien zurückgeben, wenn es Grenzen auflöst.

Verfügbarkeit: 1.4.0 - ST\_MakeLine(geomarray) wurde eingeführt. ST\_MakeLine Aggregatfunktion wurde verbessert, um mehr Punkte schneller handhaben zu können.



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



Diese Methode unterstützt kreisförmige Strings und Kurven.

## Beispiele - Verwendung von XLink

Sammeln Sie 2D-Punkte.

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

```
st_astext
```

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

Sammeln Sie 3D-Punkte.

```
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)
```

Kurven sammeln.

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

```
st_astext
```

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

### Beispiele: Verwendung der Feld-Version

Verwendung eines Array-Konstruktors für eine Subquery.

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

Verwendung eines Array-Konstruktors für Werte.

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

--wkt collect --
MULTILINESTRING((1 2,3 4),(3 4,4 5))
```

### Beispiele: Spatiale Aggregatversion

Erstellen mehrerer Sammlungen durch Gruppierung von Geometrien in einer Tabelle.

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

### Siehe auch

[ST\\_Dump](#), [ST\\_AsBinary](#)

## 7.3.2 ST\_LineFromMultiPoint

`ST_LineFromMultiPoint` — Erzeugt einen LineString aus einer MultiPoint Geometrie.

### Synopsis

geometry **ST\_LineFromMultiPoint**(geometry aMultiPoint);

### Beschreibung

Erzeugt einen LineString aus einer MultiPoint Geometrie.

Für Punkt mit X-, Y- und M-Koordinaten verwenden Sie bitte [ST\\_MakePointM](#) .



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

### Beispiele

Erzeugt einen LineString aus einer MultiPoint Geometrie.

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

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

**Siehe auch**

[ST\\_AsEWKT](#), [ST\\_AsKML](#)

### 7.3.3 ST\_MakeEnvelope

`ST_MakeEnvelope` — Erzeugt ein rechteckiges Polygon aus den gegebenen Minimum- und Maximumwerten. Die Eingabewerte müssen in dem Koordinatenreferenzsystem sein, welches durch die SRID angegeben wird.

**Synopsis**

geometry **ST\_MakeEnvelope**(float xmin, float ymin, float xmax, float ymax, integer srid=unknown);

**Beschreibung**

Erzeugt ein rechteckiges Polygon das durch die Minima und Maxima angegeben wird. durch die gegebene Hülle. Die Eingabewerte müssen in dem Koordinatenreferenzsystem sein, welches durch die SRID angegeben wird. Wenn keine SRID angegeben ist, so wird das Koordinatenreferenzsystem "unknown" angenommen

Verfügbarkeit: 1.5

Erweiterung: 2.0: es wurde die Möglichkeit eingeführt, eine Einhüllende/Envelope festzulegen, ohne dass die SRID spezifiziert ist.

**Beispiel: Ein Umgebungsrechteck Polygon erzeugen**

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

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

**Siehe auch**

[ST\\_MakePoint](#), [ST\\_MakePoint](#), [ST\\_Point](#), [ST\\_SRID](#)

### 7.3.4 ST\_MakeLine

`ST_MakeLine` — Erzeugt einen Linienzug aus einer Punkt-, Mehrfachpunkt- oder Liniengeometrie.

**Synopsis**

geometry **ST\_MakeLine**(geometry geom1, geometry geom2);  
geometry **ST\_MakeLine**(geometry[] geoms\_array);  
geometry **ST\_MakeLine**(geometry set geoms);

## Beschreibung

Erzeugt einen LineString, der die Punkte von Point-, MultiPoint- oder LineString-Geometrien enthält. Andere Geometrietypen verursachen einen Fehler.

**Variante 1:** akzeptiert zwei Eingabegeometrien

**Variante 2:** akzeptiert eine Reihe von Geometrien

**Variante 3:** Aggregatfunktion, die ein Rowset von Geometrien akzeptiert. Um die Reihenfolge der Eingabegeometrien sicherzustellen, verwenden Sie `ORDER BY` im Funktionsaufruf oder eine Unterabfrage mit einer `ORDER BY` Klausel.

Wiederholte Knoten am Anfang von Eingabe-LineStrings werden zu einem einzigen Punkt zusammengezogen. Wiederholte Punkte in Punkt- und Multipunkt-Eingaben werden nicht zusammengeklappt. [ST\\_RemoveRepeatedPoints](#) kann verwendet werden, um wiederholte Punkte aus dem Ausgabe-LineString zu klappen.



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

Verfügbarkeit: Mit 2.3.0 wurde die Unterstützung zur Eingabe von MultiPoint Elementen eingeführt

Verfügbarkeit: 2.0.0 - Unterstützung zur Eingabe von LineString Elementen eingeführt

Verfügbarkeit: 1.4.0 - `ST_MakeLine(geomarray)` wurde eingeführt. `ST_MakeLine` Aggregatfunktion wurde verbessert, um mehr Punkte schneller handhaben zu können.

## Beispiele: Verwendung der Feld-Version

Erstellen Sie eine Linie, die aus zwei Punkten besteht.

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

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

Erzeugt eine BOX3D, die durch 2 geometrische 3D-Punkte definiert wird.

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

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

Erzeugt einen Linienzug aus einer Punkt-, Mehrfachpunkt- oder Liniengeometrie.

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

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

## Beispiele: Verwendung der Feld-Version

Erstellen Sie eine Zeile aus einem Array, das durch eine Unterabfrage mit Bestellung gebildet wird.

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

Erstellen einer 3D-Linie aus einem Array von 3D-Punkten

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

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

### Beispiele: Spatiale Aggregatversion

Diese Beispiel nimmt eine Abfolge von GPS Punkten entgegen und erzeugt einen Datensatz für jeden GPS Pfad, wobei das Geometriefeld ein Linienzug ist, welcher in der Reihenfolge der Aufnahme route aus den GPS Punkten zusammengesetzt wird.

Die Verwendung des Aggregats `ORDER BY` liefert einen korrekt geordneten `LineString`.

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

Vor PostgreSQL 9 kann die Reihenfolge in einer Subquery verwendet werden. Allerdings kann es vorkommen, dass der Abfrageplan die Reihenfolge der Unterabfrage nicht berücksichtigt.

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

### Siehe auch

[ST\\_RemoveRepeatedPoints](#), [ST\\_AsText](#), [ST\\_GeomFromText](#), [ST\\_MakePoint](#)

## 7.3.5 ST\_MakePoint

`ST_MakePoint` — Erzeugt eine 2D-, 3DZ- oder 4D-Punktgeometrie.

### Synopsis

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

### Beschreibung

Erzeugt eine 2D XY, 3D XYZ oder 4D XYZM Punktgeometrie. Verwenden Sie [ST\\_MakePointM](#), um Punkte mit XYM-Koordinaten zu erstellen.

Verwenden Sie [ST\\_SetSRID](#), um eine SRID für den erstellten Punkt anzugeben.

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

Bei geodätischen Koordinaten ist X der Längengrad und Y der Breitengrad.

**Note**

Die Funktionen [ST\\_Point](#), [ST\\_PointZ](#), [ST\\_PointM](#) und [ST\\_PointZM](#) können verwendet werden, um Punkte mit einer bestimmten SRID zu erstellen.



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

**Beispiele**

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

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

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

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

**Siehe auch**

[ST\\_GeomFromText](#), [ST\\_PointFromText](#), [ST\\_SetSRID](#), [ST\\_MakePointM](#), [ST\\_Point](#), [ST\\_PointZ](#), [ST\\_PointM](#), [ST\\_PointZM](#)

### 7.3.6 ST\_MakePointM

`ST_MakePointM` — Erzeugt einen Punkt mit x, y und measure/Kilometrierungs Koordinaten.

**Synopsis**

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

**Beschreibung**

Erstellt einen Punkt mit X-, Y- und M-Koordinaten (Maß). Verwenden Sie [ST\\_MakePoint](#), um Punkte mit XY-, XYZ- oder XYZM-Koordinaten zu erstellen.

Verwenden Sie [ST\\_SetSRID](#), um eine SRID für den erstellten Punkt anzugeben.

**Note**

Bei geodätischen Koordinaten ist X der Längengrad und Y der Breitengrad.

**Note**

Die Funktionen [ST\\_PointM](#) und [ST\\_PointZM](#) können verwendet werden, um Punkte mit einem M-Wert und einem bestimmten SRID zu erstellen.

## Beispiele



### Note

**ST\_AsEWKT** wird für die Textausgabe verwendet, da **ST\_AsText** keine M-Werte unterstützt.

Punkt mit unbekanntem SRID erstellen.

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

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

Erzeugt einen Punkt mit x, y und measure/Kilometrierungs Koordinaten.

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

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

Ermittelt das Maß des erstellten Punktes.

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

result
-----
10
```

### Siehe auch

[ST\\_MakePoint](#), [ST\\_SetSRID](#), [ST\\_PointM](#), [ST\\_PointZM](#)

## 7.3.7 ST\_MakePolygon

**ST\_MakePolygon** — Erzeugt ein Polygon aus einer Schale und einer optionalen Liste von Löchern.

### 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.

**Variante 1:** Akzeptiert eine Shell LineString.

**Variante 2:** Akzeptiert einen Shell LineString und ein Array von inneren (Loch) LineStrings. Ein Geometrie-Array kann mit den PostgreSQL-Konstrukten `array_agg()`, `ARRAY[]` oder `ARRAY()` erstellt werden.



**Note**

Diese Funktion akzeptiert keine MULTILINESTRINGS. Verwenden Sie bitte **ST\_LineMerge** oder **ST\_Dump** um Linienzüge zu erzeugen.



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

**Beispiele: Verwendung der Feld-Version**

Erzeugt einen LineString aus einem codierten Linienzug.

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

Erstellen Sie ein Polygon aus einem offenen LineString, indem Sie **ST\_StartPoint** und **ST\_AddPoint** verwenden, um es zu schließen.

```
SELECT ST_MakePolygon( ST_AddPoint(foo.open_line, ST_StartPoint(foo.open_line)) )
FROM (
  SELECT ST_GeomFromText('LINESTRING(75 29,77 29,77 29, 75 29)') As open_line) As foo;
```

Erzeugt einen LineString aus einem codierten Linienzug.

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

```
st_asewkt
```

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

Erstellen eines Polygons aus einem LineString mit Maßen

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

```
st_asewkt
```

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

**Beispiele: Außenhülle mit inneren Ringen**

Erzeugung eines Donuts mit einem Ameisenloch

```
SELECT ST_MakePolygon( ST_ExteriorRing( ST_Buffer(ring.line,10)),
  ARRAY[ ST_Translate(ring.line, 1, 1),
        ST_ExteriorRing(ST_Buffer(ST_Point(20,20),1)) ]
  )
FROM (SELECT ST_ExteriorRing(
  ST_Buffer(ST_Point(10,10),10,10)) AS line ) AS ring;
```

Erstellen Sie einen Satz von Provinzgrenzen mit Löchern, die Seen darstellen. Die Eingabe ist eine Tabelle mit Provinz-Polygonen/MultiPolygonen und eine Tabelle mit Wasserlinien. Die Linien, die Seen bilden, werden mit **ST\_IsClosed** ermittelt. Das Provinzlinienwerk wird mit **ST\_Boundary** extrahiert. Wie von **ST\_MakePolygon** gefordert, wird die Begrenzung mit **ST\_LineMerge** zu einem einzigen LineString gezwungen. (Beachten Sie jedoch, dass dies ein ungültiges Polygon ergibt, wenn eine Provinz mehr als eine Region oder Inseln hat). Die Verwendung eines LEFT JOIN stellt sicher, dass alle Provinzen einbezogen werden, auch wenn sie keine Seen haben.

**Note**

Das Konstrukt mit CASE wird verwendet, da die Übergabe eines NULL-Feldes an ST\_MakePolygon NULL ergibt.

```
SELECT p.gid, p.province_name,
       CASE WHEN array_agg(w.geom) IS NULL
            THEN p.geom
            ELSE ST_MakePolygon( ST_LineMerge( ST_Boundary( p.geom ),
                                             array_agg(w.geom) ) END
FROM   provinces p LEFT JOIN waterlines w
       ON (ST_Within(w.geom, p.geom) AND ST_IsClosed(w.geom))
GROUP BY p.gid, p.province_name, p.geom;
```

Eine andere Technik besteht darin, eine korrelierte Unterabfrage und den ARRAY()-Konstruktor zu verwenden, der einen Zeilensatz in ein Array umwandelt.

```
SELECT p.gid, p.province_name,
       CASE WHEN EXISTS( SELECT w.geom
                        FROM waterlines w
                        WHERE ST_Within(w.geom, p.geom)
                        AND ST_IsClosed(w.geom) )
            THEN ST_MakePolygon(
                 ST_LineMerge( ST_Boundary( p.geom ),
                 ARRAY( SELECT w.geom
                        FROM waterlines w
                        WHERE ST_Within(w.geom, p.geom)
                        AND ST_IsClosed(w.geom) ) )
            ELSE p.geom
       END AS geom
FROM   provinces p;
```

**Siehe auch**

[ST\\_BuildArea](#) [ST\\_Polygon](#)

**7.3.8 ST\_Point**

ST\_Point — Erzeugt einen Punkt mit X-, Y- und SRID-Werten.

**Synopsis**

geometry **ST\_Point**(float x, float y);

geometry **ST\_Point**(float x, float y, integer srid=unknown);

**Beschreibung**

Gibt einen Punkt mit den angegebenen X- und Y-Koordinatenwerten zurück. Dies ist das SQL-MM-Äquivalent für [ST\\_MakePoint](#), das nur X und Y benötigt.

**Note**

Bei geodätischen Koordinaten ist X der Längengrad und Y der Breitengrad.

Verbessert: 3.2.0 srid wurde als zusätzliches optionales Argument hinzugefügt. Ältere Installationen erfordern die Kombination mit ST\_SetSRID, um das Raster auf der Geometrie zu markieren.



Diese Methode setzt die SQL/MM-Spezifikation um. SQL-MM 3: 6.1.2

### Beispiele: Geometrie

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

Erstellen eines Punktes mit angegebener SRID:

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

Alternative Möglichkeit der Angabe von SRID:

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

### Beispiele: Geographie

Erstellen Sie die Punkte **geography** mit der Syntax `:: cast`:

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

Pre-PostGIS 3.2 Code, mit CAST:

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

Wenn die Punktkoordinaten nicht in einem geodätischen Koordinatensystem (z. B. WGS84) vorliegen, müssen sie vor der Übertragung in eine Geografie neu projiziert werden. In diesem Beispiel wird ein Punkt in Pennsylvania State Plane feet (SRID 2273) auf WGS84 (SRID 4326) projiziert.

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

### Siehe auch

[ST\\_MakePoint](#), [ST\\_PointZ](#), [ST\\_PointM](#), [ST\\_PointZM](#), [ST\\_SetSRID](#), [ST\\_Transform](#)

## 7.3.9 ST\_PointZ

ST\_PointZ — Erzeugt einen Punkt mit X-, Y-, Z- und SRID-Werten.

### Synopsis

geometry **ST\_PointZ**(float x, float y, float z, integer srid=unknown);

### Beschreibung

Gibt einen ST\_Point mit den gegebenen Koordinatenwerten aus. Ein OGC-Alias für ST\_MakePoint.

Verbessert: 3.2.0 srid wurde als zusätzliches optionales Argument hinzugefügt. Ältere Installationen erfordern die Kombination mit ST\_SetSRID, um das Raster auf der Geometrie zu markieren.

## Beispiele

```
SELECT ST_PointZ(-71.104, 42.315, 3.4, 4326)
```

```
SELECT ST_PointZ(-71.104, 42.315, 3.4, srid => 4326)
```

```
SELECT ST_PointZ(-71.104, 42.315, 3.4)
```

## Siehe auch

[ST\\_MakePoint](#), [ST\\_PointFromText](#), [ST\\_SetSRID](#), [ST\\_MakePointM](#)

### 7.3.10 ST\_PointM

`ST_PointM` — Erzeugt einen Punkt mit den Werten X, Y, M und SRID.

#### Synopsis

geometry **ST\_PointM**(float x, float y, float m, integer srid=unknown);

#### Beschreibung

Gibt einen `ST_Point` mit den gegebenen Koordinatenwerten aus. Ein OGC-Alias für `ST_MakePoint`.

Verbessert: 3.2.0 srid wurde als zusätzliches optionales Argument hinzugefügt. Ältere Installationen erfordern die Kombination mit `ST_SetSRID`, um das Raster auf der Geometrie zu markieren.

## Beispiele

```
SELECT ST_PointM(-71.104, 42.315, 3.4, 4326)
```

```
SELECT ST_PointM(-71.104, 42.315, 3.4, srid => 4326)
```

```
SELECT ST_PointM(-71.104, 42.315, 3.4)
```

## Siehe auch

[ST\\_MakePoint](#), [ST\\_PointFromText](#), [ST\\_SetSRID](#), [ST\\_MakePointM](#)

### 7.3.11 ST\_PointZM

`ST_PointZM` — Erzeugt einen Punkt mit den Werten X, Y, Z, M und SRID.

#### Synopsis

geometry **ST\_PointZM**(float x, float y, float z, float m, integer srid=unknown);

---

## Beschreibung

Gibt einen `ST_Point` mit den gegebenen Koordinatenwerten aus. Ein OGC-Alias für `ST_MakePoint`.

Verbessert: 3.2.0 `srid` wurde als zusätzliches optionales Argument hinzugefügt. Ältere Installationen erfordern die Kombination mit `ST_SetSRID`, um das Raster auf der Geometrie zu markieren.

## Beispiele

```
SELECT ST_PointZM(-71.104, 42.315, 3.4, 4.5, 4326)
```

```
SELECT ST_PointZM(-71.104, 42.315, 3.4, 4.5, srid => 4326)
```

```
SELECT ST_PointZM(-71.104, 42.315, 3.4, 4.5)
```

## Siehe auch

[ST\\_MakePoint](#), [ST\\_Point](#), [ST\\_PointM](#), [ST\\_PointZ](#), [ST\\_SetSRID](#)

## 7.3.12 ST\_Polygon

`ST_Polygon` — Erzeugt ein Polygon aus einem `LineString` mit einem angegebenen `SRID`.

## Synopsis

geometry **ST\_Polygon**(geometry lineString, integer srid);

## Beschreibung

Gibt ein Polygon aus dem angegebenen `LineString` zurück und setzt das räumliche Bezugssystem aus dem `srid`.

`ST_Polygon` ist ähnlich wie [ST\\_MakePolygon](#) Variante 1 mit dem Zusatz, dass der `SRID` gesetzt wird.

, [ST\\_MakePoint](#), [ST\\_SetSRID](#)



### Note

Diese Funktion akzeptiert keine `MULTILINESTRINGs`. Verwenden Sie bitte [ST\\_LineMerge](#) oder [ST\\_Dump](#) um Linienzüge zu erzeugen.



Diese Methode implementiert die [OGC Simple Features Implementation Specification for SQL 1.1](#).



Diese Methode setzt die SQL/MM-Spezifikation um. SQL-MM 3: 8.3.2



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

## Beispiele

Erstellen Sie ein 2D-Polygon.

```
SELECT ST_AsText( ST_Polygon('LINESTRING(75 29, 77 29, 77 29, 75 29)')::geometry, 4326) );  
  
-- result --  
POLYGON((75 29, 77 29, 77 29, 75 29))
```

Erstellen Sie ein 3D-Polygon.

```
SELECT ST_AsEWKT( ST_Polygon( ST_GeomFromEWKT('LINESTRING(75 29 1, 77 29 2, 77 29 3, 75 29 1) ←  
1)'), 4326) );  
  
-- result --  
SRID=4326;POLYGON((75 29 1, 77 29 2, 77 29 3, 75 29 1))
```

## Siehe auch

[ST\\_AsEWKT](#), [ST\\_AsText](#), [ST\\_GeomFromEWKT](#), [ST\\_GeomFromText](#), [ST\\_LineMerge](#), [ST\\_MakePolygon](#)

### 7.3.13 ST\_TileEnvelope

`ST_TileEnvelope` — Erzeugt ein rechteckiges Polygon in [Web Mercator](#) (SRID:3857) unter Verwendung des [XYZ-Kachelsystems](#).

#### Synopsis

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

#### Beschreibung

Erzeugt ein rechteckiges Polygon, das die Ausdehnung einer Kachel im [XYZ-Kachelsystem](#) angibt. Die Kachel wird durch die Zoomstufe Z und den XY-Index der Kachel im Raster auf dieser Stufe angegeben. Kann verwendet werden, um die Kachelgrenzen zu definieren, die von [ST\\_AsMVTGeom](#) benötigt werden, um Geometrie in den MVT-Kachelkoordinatenraum zu konvertieren.

Standardmäßig ist die Kachelhülle im [Web Mercator](#)-Koordinatensystem (SRID:3857) unter Verwendung des Standardbereichs des Web Mercator-Systems (-20037508.342789, 20037508.342789). Dies ist das am häufigsten verwendete Koordinatensystem für MVT-Kacheln. Der optionale Parameter `bounds` kann verwendet werden, um Kacheln in einem beliebigen Koordinatensystem zu erzeugen. Es handelt sich um eine Geometrie, die den SRID und die Ausdehnung des Quadrats "Zoomstufe Null" hat, in das das XYZ-Kachelsystem eingeschrieben ist.

Der optionale Parameter `margin` kann verwendet werden, um eine Kachel um den angegebenen Prozentsatz zu vergrößern. Z.B. `margin=0.125` vergrößert die Kachel um 12,5%, was bei einer Kachelgröße von 4096, wie sie in [ST\\_AsMVTGeom](#) verwendet wird, einem `Puffer=512` entspricht. Dies ist nützlich, um einen Kachelpuffer zu erstellen, der Daten enthält, die außerhalb des sichtbaren Bereichs der Kachel liegen, deren Vorhandensein sich aber auf die Darstellung der Kachel auswirkt. Beispielsweise könnte ein Städtename (ein Punkt) in der Nähe eines Kachelrandes liegen, so dass seine Beschriftung auf zwei Kacheln gerendert werden sollte, obwohl sich der Punkt im sichtbaren Bereich nur einer Kachel befindet. Wenn Sie erweiterte Kacheln in einer Abfrage verwenden, wird der Stadtpunkt in beide Kacheln aufgenommen. Verwenden Sie stattdessen einen negativen Wert, um die Kachel zu verkleinern. Werte unter -0,5 sind unzulässig, da die Kachel dann vollständig verschwinden würde. Geben Sie keinen Rand an, wenn Sie [ST\\_AsMVTGeom](#) verwenden. Siehe das Beispiel für [ST\\_AsMVT](#).

Erweiterung: 2.0.0 Standardwert für den optionalen Parameter SRID eingefügt.

Verfügbarkeit: 2.1.0

**Beispiel: Ein Umgebungsrechteck Polygon erzeugen**

```
SELECT ST_AsText( ST_TileEnvelope(2, 1, 1) );

st_astext
-----
POLYGON((-10018754.1713945 0,-10018754.1713945 10018754.1713945,0 10018754.1713945,0 ↔
0,-10018754.1713945 0))

SELECT ST_AsText( ST_TileEnvelope(3, 1, 1, ST_MakeEnvelope(-180, -90, 180, 90, 4326) ) );

st_astext
-----
POLYGON((-135 45,-135 67.5,-90 67.5,-90 45,-135 45))
```

**Siehe auch**[ST\\_MakeEnvelope](#)**7.3.14 ST\_HexagonGrid**

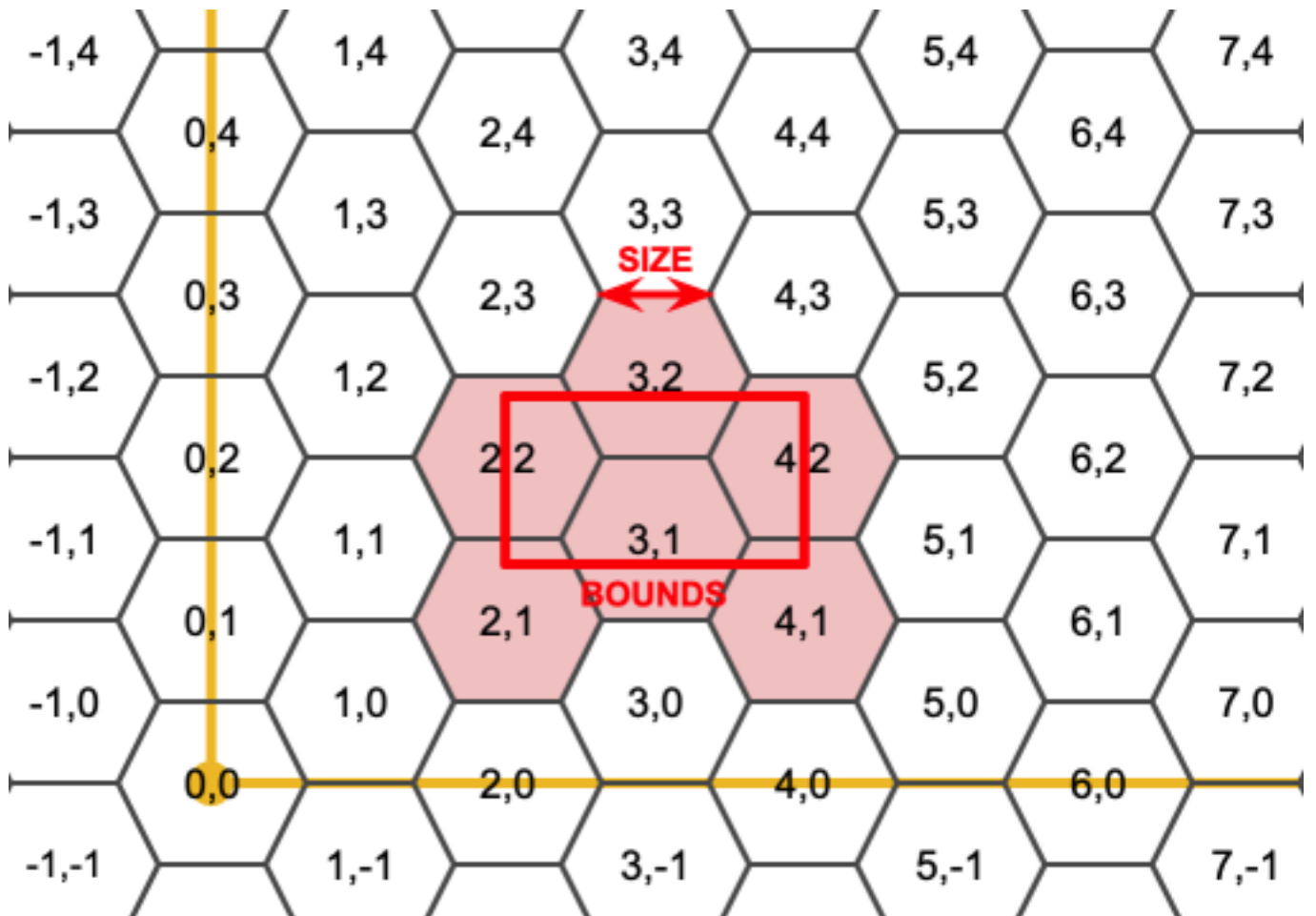
**ST\_HexagonGrid** — Gibt eine Menge von Sechsecken und Zellindizes zurück, die die Grenzen des Arguments Geometrie vollständig abdecken.

**Synopsis**

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

**Beschreibung**

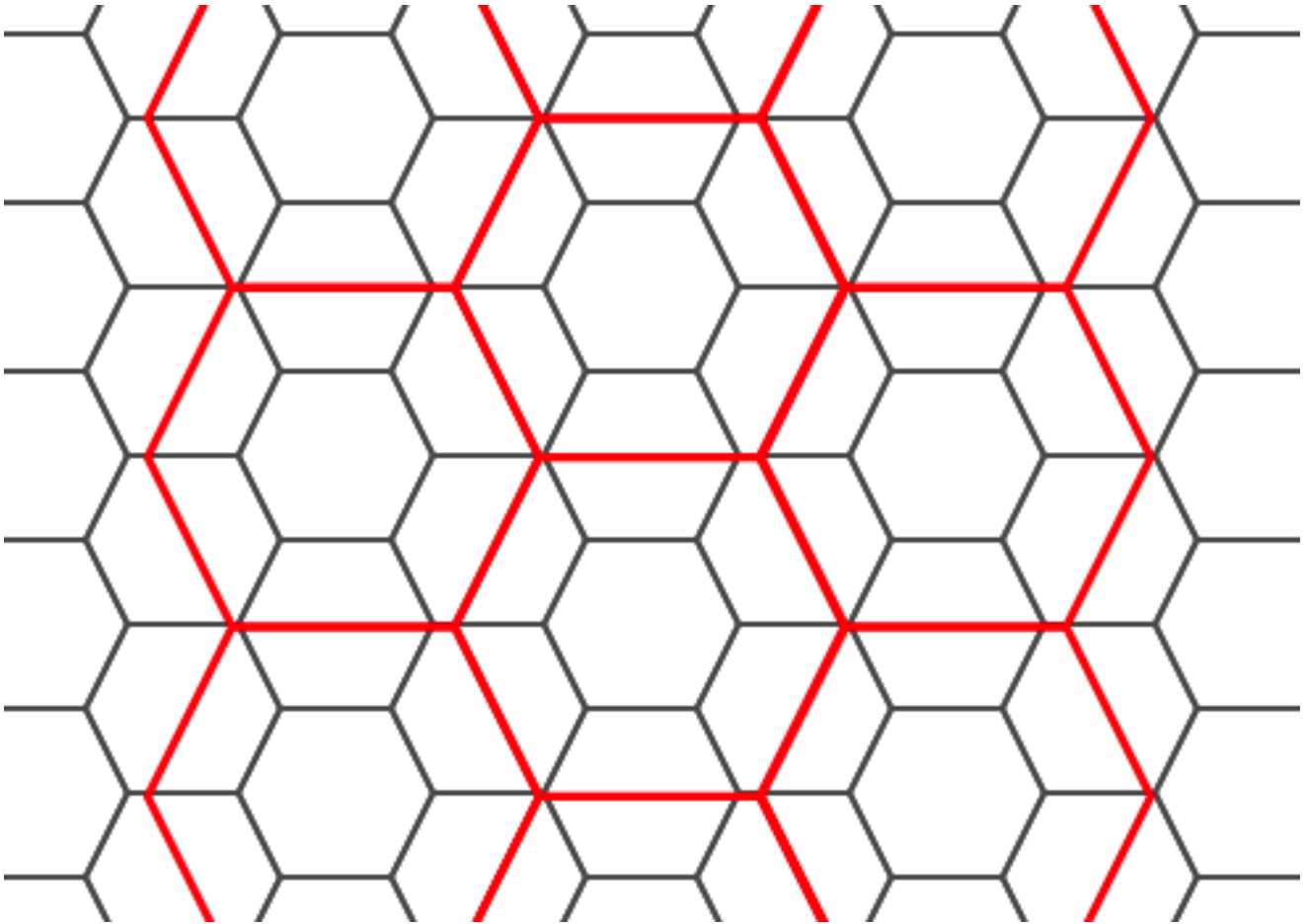
Beginnt mit dem Konzept einer Sechseckkachelung der Ebene. (Es handelt sich nicht um eine hexagonale Kachelung der Erdkugel, sondern um das **H3** Kachelungsschema.) Für eine gegebene planare SRS und eine gegebene Kantengröße gibt es, ausgehend vom Ursprung der SRS, eine einzige hexagonale Kachelung der Ebene, `Tiling(SRS, Size)`. Diese Funktion beantwortet die Frage, welche Sechsecke in einem gegebenen `Tiling(SRS, Size)` sich mit einer gegebenen Grenze überschneiden.



Die SRS für die Ausgangssehsecke ist die SRS, die durch die Begrenzungsgeometrie bereitgestellt wird.

Durch Verdoppelung oder Verdreifachung der Kantengröße des Sechsecks wird eine neue übergeordnete Kachel erzeugt, die zur ursprünglichen Kachel passt. Leider ist es nicht möglich, übergeordnete Sechseckkacheln zu erzeugen, in die die untergeordneten Kacheln perfekt hineinpassen.





Verfügbarkeit: 2.1.0

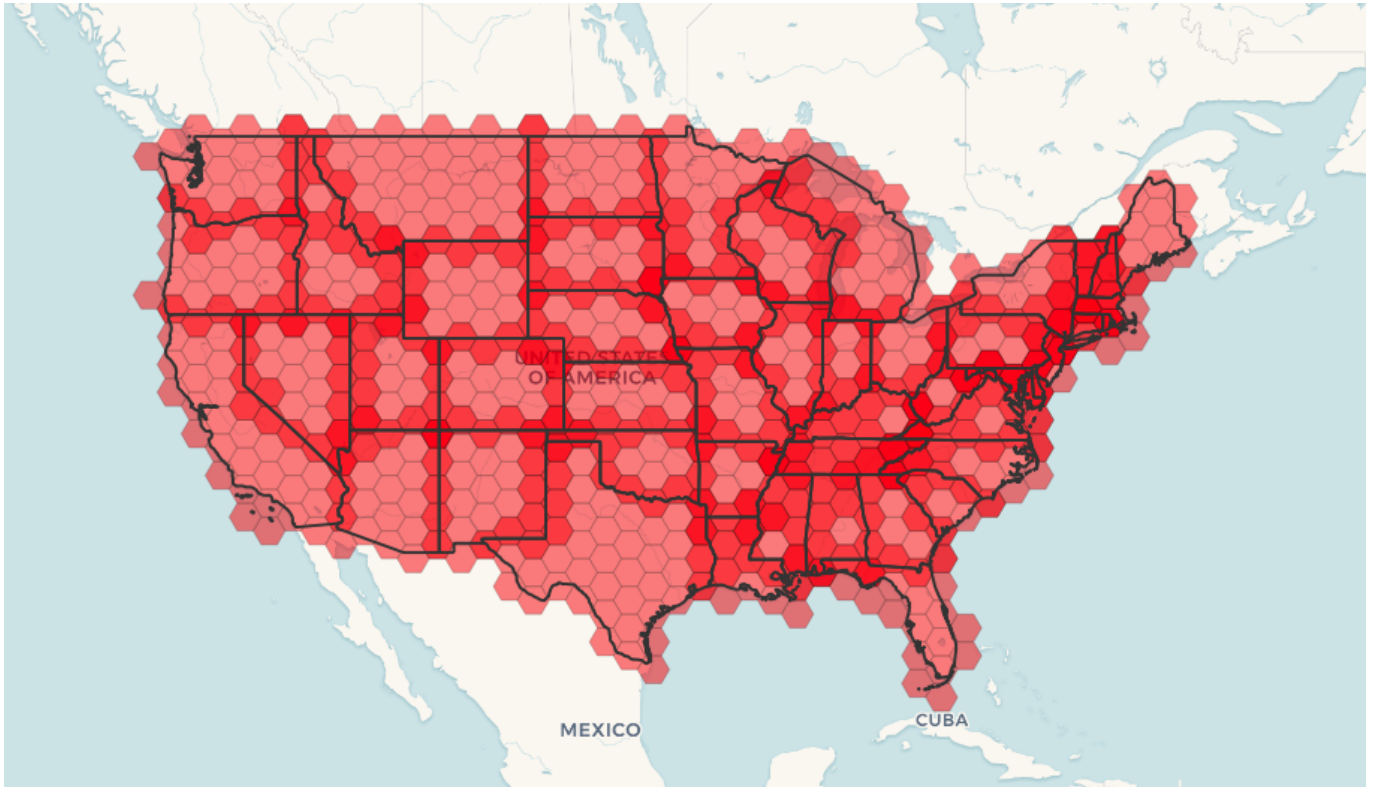
### Beispiele: Verwendung der Feld-Version

Um eine Punktzusammenfassung anhand eines sechseckigen Kachels vorzunehmen, erstellen Sie ein Sechseckgitter, wobei Sie die Ausdehnung der Punkte als Grenzen verwenden, und verbinden Sie es dann räumlich mit diesem Gitter.

```
SELECT COUNT(*), hexes.geom
FROM
  ST_HexagonGrid(
    10000,
    ST_SetSRID(ST_EstimatedExtent('pointtable', 'geom'), 3857)
  ) AS hexes
INNER JOIN
  pointtable AS pts
  ON ST_Intersects(pts.geom, hexes.geom)
GROUP BY hexes.geom;
```

### Beispiel: Ein Umgebungsrechteck Polygon erzeugen

Wenn wir für jede Polygongrenze eine Reihe von Sechsecken erzeugen und diejenigen herausfiltern, die sich nicht mit ihren Sechsecken schneiden, erhalten wir ein Tiling für jedes Polygon.



Die Kachelung von Zuständen führt dazu, dass jeder Zustand von einem Sechseck abgedeckt wird und sich mehrere Sechsecke an den Grenzen zwischen den Zuständen überlappen.

**Note**

Das Schlüsselwort `LATERAL` wird für Funktionen mit Mengenrückgabe impliziert, wenn auf eine vorherige Tabelle in der `FROM`-Liste verwiesen wird. `CROSS JOIN LATERAL`, `CROSS JOIN` oder einfach nur `,` sind also gleichwertige Konstrukte für dieses Beispiel.

```
SELECT admin1.gid, hex.geom
FROM
  admin1
  CROSS JOIN
  ST_HexagonGrid(100000, admin1.geom) AS hex
WHERE
  adm0_a3 = 'USA'
  AND
  ST_Intersects(admin1.geom, hex.geom)
```

**Siehe auch**

[ST\\_EstimatedExtent](#), [ST\\_MakePoint](#), [ST\\_Point](#), [ST\\_SRID](#)

### 7.3.15 ST\_Hexagon

`ST_Hexagon` — Liefert ein einzelnes Sechseck unter Verwendung der angegebenen Kantengröße und Zellkoordinate innerhalb des Sechseck-Gitterraums.

## Synopsis

geometry **ST\_Hexagon**(float8 size, integer cell\_i, integer cell\_j, geometry origin);

## Beschreibung

Verwendet dasselbe Sechseck-Kacheln-Konzept wie **ST\_HexagonGrid**, erzeugt aber nur ein Sechseck an der gewünschten Zellkoordinate. Optional kann die Ursprungsordinate der Kacheln angepasst werden, der Standardursprung liegt bei 0,0.

Sechsecke werden ohne SRID generiert. Verwenden Sie daher **ST\_SetSRID**, um den SRID auf den von Ihnen erwarteten Wert zu setzen.

Verfügbarkeit: 2.1.0

## Beispiel: Erstellen eines Sechsecks im Ursprung

```
SELECT ST_AsText(ST_SetSRID(ST_Hexagon(1.0, 0, 0), 3857));

POLYGON((-1 0,-0.5
         -0.866025403784439,0.5
         -0.866025403784439,1
         0,0.5
         0.866025403784439,-0.5
         0.866025403784439,-1 0))
```

## Siehe auch

[ST\\_TileEnvelope](#), [ST\\_MakePoint](#), [ST\\_SetSRID](#)

## 7.3.16 ST\_SquareGrid

**ST\_SquareGrid** — Gibt eine Menge von Gitterquadraten und Zellindizes zurück, die die Grenzen des Arguments Geometrie vollständig abdecken.

## Synopsis

setof record **ST\_SquareGrid**(float8 size, geometry bounds);

## Beschreibung

Beginnt mit dem Konzept einer quadratischen Kachelung der Ebene. Für eine gegebene planare SRS und eine gegebene Kantenlänge gibt es, ausgehend vom Ursprung der SRS, ein einziges quadratisches Tiling der Ebene, **Tiling(SRS, Size)**. Diese Funktion beantwortet die Frage, welche Gitter in einem gegebenen **Tiling(SRS, Size)** sich mit einer gegebenen Grenze überlappen.

Die SRS für die Ausgangsquadrate ist die SRS, die durch die Begrenzungsgeometrie bereitgestellt wird.

Durch Verdoppelung der Größe des Quadrats oder der Kanten wird eine neue übergeordnete Kachel erzeugt, die perfekt zur ursprünglichen Kachel passt. Standard-Webmap-Kacheln in Mercator sind einfach Zweierpotenzen von quadratischen Gittern in der Mercator-Ebene.

Verfügbarkeit: 2.1.0

### Beispiel: Ein Umgebungsrechteck Polygon erzeugen

Das Raster füllt die gesamten Grenzen des Landes aus. Wenn Sie also nur Quadrate haben wollen, die das Land berühren, müssen Sie anschließend mit `ST_Intersects` filtern.

```
WITH grid AS (  
SELECT (ST_SquareGrid(1, ST_Transform(geom, 4326))).*  
FROM admin0 WHERE name = 'Canada'  
)  
SELECT ST_AsText(geom)  
FROM grid
```

### Beispiel: Zählen von Punkten in Quadraten (unter Verwendung eines einfach geschnittenen Gitters)

Um eine Punktzusammenfassung anhand einer quadratischen Kachel zu erstellen, erzeugen Sie ein quadratisches Gitter, das die Ausdehnung der Punkte als Grenzen verwendet, und verbinden Sie es dann räumlich mit diesem Gitter. Beachten Sie, dass die geschätzte Ausdehnung von der tatsächlichen Ausdehnung abweichen kann, seien Sie also vorsichtig und stellen Sie zumindest sicher, dass Sie Ihre Tabelle analysiert haben.

```
SELECT COUNT(*), squares.geom  
FROM  
pointtable AS pts  
INNER JOIN  
ST_SquareGrid(  
1000,  
ST_SetSRID(ST_EstimatedExtent('pointtable', 'geom'), 3857)  
) AS squares  
ON ST_Intersects(pts.geom, squares.geom)  
GROUP BY squares.geom
```

### Beispiel: Zählen von Punkten in Quadraten unter Verwendung eines Rasters pro Punkt

Dies führt zum gleichen Ergebnis wie das erste Beispiel, ist aber bei einer großen Anzahl von Punkten langsamer

```
SELECT COUNT(*), squares.geom  
FROM  
pointtable AS pts  
INNER JOIN  
ST_SquareGrid(  
1000,  
pts.geom  
) AS squares  
ON ST_Intersects(pts.geom, squares.geom)  
GROUP BY squares.geom
```

### Siehe auch

[ST\\_TileEnvelope](#), [ST\\_Point](#), [ST\\_SetSRID](#), [ST\\_SRID](#)

## 7.3.17 ST\_Square

`ST_Square` — Gibt ein einzelnes Quadrat mit der angegebenen Kantengröße und Zellkoordinate innerhalb des quadratischen Gitterraums zurück.

## Synopsis

geometry **ST\_Square**(float8 size, integer cell\_i, integer cell\_j, geometry origin);

## Beschreibung

Verwendet dasselbe Konzept der quadratischen Kachelung wie **ST\_SquareGrid**, erzeugt aber nur ein Quadrat an der gewünschten Zellkoordinate. Optional kann die Ursprungscoordinate der Kacheln angepasst werden, der Standardursprung liegt bei 0,0.

Es werden Quadrate erzeugt, ohne dass ein SRID gesetzt ist. Verwenden Sie daher **ST\_SetSRID**, um den SRID auf den von Ihnen erwarteten Wert zu setzen.

Verfügbarkeit: 2.1.0

## Beispiel: Erstellen eines Quadrats im Ursprung

```
SELECT ST_AsText(ST_SetSRID(ST_Square(1.0, 0, 0), 3857));  
  
POLYGON((0 0,0 1,1 1,1 0,0 0))
```

## Siehe auch

[ST\\_TileEnvelope](#), [ST\\_MakeLine](#), [ST\\_MakePolygon](#)

## 7.3.18 ST\_Letters

**ST\_Letters** — Gibt die eingegebenen Buchstaben als Geometrie mit einer Standardstartposition am Ursprung und einer Standardtexthöhe von 100 zurück.

## Synopsis

geometry **ST\_Letters**(text letters, json font);

## Beschreibung

Verwendet eine eingebaute Schriftart, um eine Zeichenkette als Multipolygoneometrie darzustellen. Die Standard Texthöhe ist 100.0, der Abstand von der Unterkante einer Unterlänge bis zur Oberkante eines Großbuchstabens. Die Standard-Startposition setzt den Beginn der Grundlinie auf den Ursprung. Um die Schriftart zu überschreiben, muss eine json-Map mit einem Zeichen als Schlüssel und base64-kodierten TWKB für die Schriftform übergeben werden, wobei die Schriftarten eine Höhe von 1000 Einheiten von der Unterkante der Unterlängen bis zur Oberkante der Großbuchstaben haben.

Der Text wird standardmäßig am Ursprung erzeugt. Um den Text neu zu positionieren und in der Größe zu verändern, wenden Sie zuerst die Funktion **ST\_Scale** und dann die Funktion **ST\_Translate** an.

Verfügbarkeit: 3.3.0

## Beispiel: Ein Umgebungsrechteck Polygon erzeugen

```
SELECT ST_AsText(ST_Letters('Yo'), 1);
```



Von `ST_Letters` erzeugte Briefe

#### Beispiel: Wörter skalieren und verschieben

```
SELECT ST_Translate(ST_Scale(ST_Letters('Yo'), 10, 10), 100,100);
```

#### Siehe auch

[ST\\_AsTWKB](#), [ST\\_Scale](#), [ST\\_Translate](#)

## 7.4 Geometrische Zugriffsfunktionen

### 7.4.1 GeometryType

`GeometryType` — Gibt den Geometrietyp des `ST_Geometry` Wertes zurück.

#### Synopsis

```
text GeometryType(geometry geomA);
```

#### Beschreibung

Gibt den Geometrietyp als Zeichenkette zurück. z.B.: 'LINESTRING', 'POLYGON', 'MULTIPOINT', etc.

OGC SPEC s2.1.1.1 - Gibt den Namen des 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.

- ✔ Diese Methode implementiert die [OGC Simple Features Implementation Specification for SQL 1.1](#).
- ✔ Diese Methode unterstützt kreisförmige Strings und Kurven.
- ✔ Diese Funktion unterstützt 3d und lässt den Z-Index nicht fallen.
- ✔ Diese Funktion unterstützt polyedrische Flächen.
- ✔ Diese Funktion unterstützt Dreiecke und dreieckige unregelmäßige Netzoberflächen (TIN).

### Beispiele

```
SELECT GeometryType(ST_GeomFromText('LINESTRING(77.29 29.07,77.42 29.26,77.27 29.31,77.29 29.07)'));
geometrytype
-----
LINESTRING
```

```
SELECT ST_GeometryType(ST_GeomFromEWKT('POLYHEDRALSURFACE( ((0 0 0, 0 0 1, 0 1 1, 0 1 0, 0 0 0)),
((0 0 0, 0 1 0, 1 1 0, 1 0 0, 0 0 0)), ((0 0 0, 1 0 0, 1 0 1, 0 0 1, 0 0 0)
),
((1 1 0, 1 1 1, 1 0 1, 1 0 0, 1 1 0)),
((0 1 0, 0 1 1, 1 1 1, 1 1 0, 0 1 0)), ((0 0 1, 1 0 1, 1 1 1, 0 1 1, 0 0 1)
) )'));
--result
POLYHEDRALSURFACE
```

```
SELECT GeometryType(geom) as result
FROM
  (SELECT
    ST_GeomFromEWKT('TIN (((
      0 0 0,
      0 0 1,
      0 1 0,
      0 0 0
    )), ((
      0 0 0,
      0 1 0,
      1 1 0,
      0 0 0
    ))
  )') AS geom
  ) AS g;
result
-----
TIN
```

### Siehe auch

[ST\\_GeometryType](#)

## 7.4.2 ST\_Boundary

**ST\_Boundary** — Gibt die abgeschlossene Hülle aus der kombinierten Begrenzung der Geometrie zurück.

## Synopsis

geometry **ST\_Boundary**(geometry geomA);

## Beschreibung

Gibt die abgeschlossene Hülle aus der kombinierten Begrenzung der Geometrie zurück. Die Definition der kombinierte Begrenzung ist in Abschnitt 3.12.3.2 der OGC SPEC beschrieben. Da das Ergebnis dieser Funktion eine abgeschlossene Hülle und daher topologisch geschlossen ist, kann die resultierende Begrenzung durch geometrische Primitive, wie in Abschnitt 3.12.2. der OGC SPEC erörtert, dargestellt werden.

Wird durch das GEOS Modul ausgeführt



### Note

Vor 2.0.0 meldete diese Funktion einen Fehler, falls sie auf eine `GEOMETRYCOLLECTION` angewandt wurde. Ab 2.0.0 wird stattdessen `NULL` (nicht unterstützte Eingabe) zurückgegeben.



Diese Methode implementiert die [OGC Simple Features Implementation Specification for SQL 1.1](#). OGC SPEC s2.1.1.1



Diese Methode setzt die SQL/MM-Spezifikation um. SQL-MM IEC 13249-3: 5.1.17



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


Erweiterung: mit 2.1.0 wurde die Unterstützung von Dreiecken eingeführt

Geändert: 3.2.0 Unterstützung für TIN, verwendet keine Geos, linearisiert keine Kurven

## Beispiele

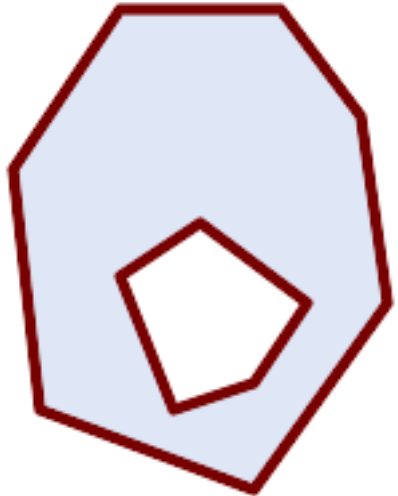
---





**Linienzug mit überlagerten Begrenzungspunkten**

```
SELECT ST_Boundary(geom)
FROM (SELECT 'LINESTRING(100 150,50 60, ←
      70 80, 160 170)>:::geometry As geom) As f;
ST_AsText output
MULTIPOINT((100 150),(160 170))
```



**Polygon mit Lücke und der Abgrenzung/Boundary als Multilinestring**

```
SELECT ST_Boundary(geom)
FROM (SELECT
'POLYGON (( 10 130, 50 190, 110 190, 140 ←
      150, 150 80, 100 10, 20 40, 10 130 ),
      ( 70 40, 100 50, 120 80, 80 110, ←
      50 90, 70 40 ))>:::geometry As geom) As f;
ST_AsText output
MULTILINESTRING((10 130,50 190,110 ←
      190,140 150,150 80,100 10,20 40,10 130),
      (70 40,100 50,120 80,80 110,50 ←
      90,70 40))
```

```
SELECT ST_AsText(ST_Boundary(ST_GeomFromText('LINESTRING(1 1,0 0, -1 1)')));
st_astext
-----
MULTIPOINT((1 1),(-1 1))

SELECT ST_AsText(ST_Boundary(ST_GeomFromText('POLYGON((1 1,0 0, -1 1, 1 1)'))));
st_astext
-----
LINESTRING(1 1,0 0,-1 1,1 1)

--Using a 3d polygon
SELECT ST_AsEWKT(ST_Boundary(ST_GeomFromEWKT('POLYGON((1 1 1,0 0 1, -1 1 1, 1 1 1)'))));
st_asewkt
-----
LINESTRING(1 1 1,0 0 1,-1 1 1,1 1 1)

--Using a 3d multilinestring
SELECT ST_AsEWKT(ST_Boundary(ST_GeomFromEWKT('MULTILINESTRING((1 1 1,0 0 0.5, -1 1 1),(1 1 ←
      0.5,0 0 0.5, -1 1 0.5, 1 1 0.5) )'))));
st_asewkt
-----
```

```
MULTIPOINT((-1 1 1), (1 1 0.75))
```

### Siehe auch

[ST\\_AsText](#), [ST\\_ExteriorRing](#), [ST\\_MakePolygon](#)

## 7.4.3 ST\_BoundingDiagonal

`ST_BoundingDiagonal` — Gibt die Diagonale des 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.

Der `fits` Parameter bestimmt ob die bestmögliche Anpassung notwendig ist. Wenn er `FALSE` ist, so kann auch die Diagonale eines etwas größeren Umgebungsrechtecks akzeptiert werden (dies ist für Geometrien mit vielen Knoten schneller). Auf jeden Fall wird immer die gesamte Eingabegeometrie durch das von der Diagonale bestimmten Umgebungsrechtecks abgedeckt.

Die zurückgegebene Linienzug-Geometrie beinhaltet immer die SRID und die Dimensionalität (Anwesenheit von Z und M) der eingegebenen Geometrie.



#### Note

Bei Spezialfällen (ein einzelner Knoten als Eingabewert) ist der zurückgegebene Linienzug topologisch ungültig (kein Inneres/Interior). Das Ergebnis ist dadurch jedoch nicht semantisch ungültig.

Verfügbarkeit: 2.2.0



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



Diese Funktion unterstützt M-Koordinaten.

### Beispiele

```
-- Get the minimum X in a buffer around a point
SELECT ST_X(ST_StartPoint(ST_BoundingDiagonal(
  ST_Buffer(ST_Point(0,0),10)
)));
 st_x
-----
-10
```

### Siehe auch

[ST\\_StartPoint](#), [ST\\_EndPoint](#), [ST\\_X](#), [ST\\_Y](#), [ST\\_Z](#), [ST\\_M](#), &&&

### 7.4.4 ST\_CoordDim

ST\_CoordDim — Gibt die Dimension der Koordinaten für den Wert von ST\_Geometry zurück.







#### Synopsis

```
integer ST_CoordDim(geometry geomA);
```

#### Beschreibung

Gibt die Dimension der Koordinaten für den Wert von ST\_Geometry zurück.

Dies ist der MM konforme Alias für [ST\\_NDims](#)

-  Diese Methode implementiert die [OGC Simple Features Implementation Specification for SQL 1.1](#).
-  Diese Methode setzt die SQL/MM-Spezifikation um. SQL-MM 3: 5.1.3
-  Diese Methode unterstützt kreisförmige Strings und Kurven.
-  Diese Funktion unterstützt 3d und lässt den Z-Index nicht fallen.
-  Diese Funktion unterstützt polyedrische Flächen.
-  Diese Funktion unterstützt Dreiecke und dreieckige unregelmäßige Netzoberflächen (TIN).

#### Beispiele

```
SELECT ST_CoordDim('CIRCULARSTRING(1 2 3, 1 3 4, 5 6 7, 8 9 10, 11 12 13)');
      ---result---
      3

      SELECT ST_CoordDim(ST_Point(1,2));
      --result--
      2
```

#### Siehe auch

[ST\\_NDims](#)

### 7.4.5 ST\_Dimension

ST\_Dimension — Gibt die Dimension der Koordinaten für den Wert von ST\_Geometry zurück.

#### Synopsis

```
integer ST_Dimension(geometry g);
```

## Beschreibung

Die inhärente Dimension eines geometrischen Objektes, welche kleiner oder gleich der Dimension der Koordinaten sein muss. Nach OGC SPEC s2.1.1.1 wird 0 für POINT, 1 für LINESTRING, 2 für POLYGON, und die größte Dimension der Teile einer GEOMETRYCOLLECTION zurückgegeben. Wenn die Dimension nicht bekannt ist (leereGEOMETRYCOLLECTION) wird 0 zurückgegeben.



Diese Methode setzt die SQL/MM-Spezifikation um. 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.



Diese Funktion unterstützt polyedrische Flächen.



Diese Funktion unterstützt Dreiecke und dreieckige unregelmäßige Netzoberflächen (TIN).

## Beispiele

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

## Siehe auch

[ST\\_NDims](#)

## 7.4.6 ST\_Dump

ST\_Dump — Gibt einen Satz von `geometry_dump` Zeilen für die Komponenten einer Geometrie zurück.

### Synopsis

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

### Beschreibung

Eine Funktion, die eine Menge zurückgibt (SRF), die die Komponenten einer Geometrie extrahiert. Sie gibt einen Satz von `geometry_dump` Zeilen zurück, die jeweils eine Geometrie (`geom` Feld) und eine Reihe von Ganzzahlen (`path` Feld) enthalten.

Für einen atomaren Geometriotyp (POINT, LINESTRING, POLYGON) wird ein einzelner Datensatz mit einem leeren Array `path` und die Eingabegeometrie als `geom` zurückgegeben. Bei einer Sammlung oder Multi-Geometrie wird ein Datensatz für jede der Komponenten der Sammlung zurückgegeben, und der `path` bezeichnet die Position der Komponente innerhalb der Sammlung.

ST\_Dump ist nützlich für die Erweiterung von Geometrien. Es ist die Umkehrung einer `ST_Collect` / GROUP BY, da es neue Zeilen erstellt. Zum Beispiel kann es verwendet werden, um MULTIPOLYGONS in POLYGONS zu erweitern.

Erweiterung: Mit 2.0.0 wurde die Unterstützung für polyedrische Oberflächen, Dreiecke und TIN eingeführt.

Verfügbarkeit: PostGIS 1.0.0RC1. Benötigt PostgreSQL 7.3 oder höher.

**Note**

Vor 1.3.4 ist diese Funktion abgestürzt, wenn die Geometrien CURVES enthalten. Dies wurde mit 1.3.4+ behoben

- ✔ Diese Methode unterstützt kreisförmige Strings und Kurven.
- ✔ Diese Funktion unterstützt polyedrische Flächen.
- ✔ Diese Funktion unterstützt Dreiecke und dreieckige unregelmäßige Netzoberflächen (TIN).
- ✔ Diese Funktion unterstützt 3d und lässt den Z-Index nicht fallen.

**Standard Beispiele**

```
SELECT sometable.field1, sometable.field1,
       (ST_Dump(sometable.geom)).geom AS geom
FROM sometable;

-- Break a compound curve into its constituent linestrings and circularstrings
SELECT ST_AsEWKT(a.geom), ST_HasArc(a.geom)
FROM ( SELECT (ST_Dump(p_geom)).geom AS geom
       FROM (SELECT ST_GeomFromEWKT('COMPOUNDCURVE(CIRCULARSTRING(0 0, 1 1, 1 0),(1 0, 0 1)') AS p_geom) AS b
       ) AS a;
       st_asewkt          | st_hasarc
-----+-----
CIRCULARSTRING(0 0,1 1,1 0) | t
LINESTRING(1 0,0 1)       | f
(2 rows)
```

**Beispiele für polyedrische Oberflächen, TIN und Dreieck**

```
-- Polyhedral surface example
-- Break a Polyhedral surface into its faces
SELECT (a.p_geom).path[1] As path, ST_AsEWKT((a.p_geom).geom) As geom_ewkt
FROM (SELECT ST_Dump(ST_GeomFromEWKT('POLYHEDRALSURFACE(
((0 0 0, 0 0 1, 0 1 1, 0 1 0, 0 0 0)),
((0 0 0, 0 1 0, 1 1 0, 1 0 0, 0 0 0)), ((0 0 0, 1 0 0, 1 0 1, 0 0 1, 0 0 0)), ((1 1 0, 1 1 0, 1 1 1, 1 0 1, 1 0 0, 1 1 0)),
((0 1 0, 0 1 1, 1 1 1, 1 1 0, 0 1 0)), ((0 0 1, 1 0 1, 1 1 1, 0 1 1, 0 0 1))
)') ) AS p_geom ) AS a;
```

path	geom_ewkt
1	POLYGON((0 0 0,0 0 1,0 1 1,0 1 0,0 0 0))
2	POLYGON((0 0 0,0 1 0,1 1 0,1 0 0,0 0 0))
3	POLYGON((0 0 0,1 0 0,1 0 1,0 0 1,0 0 0))
4	POLYGON((1 1 0,1 1 1,1 0 1,1 0 0,1 1 0))
5	POLYGON((0 1 0,0 1 1,1 1 1,1 1 0,0 1 0))
6	POLYGON((0 0 1,1 0 1,1 1 1,0 1 1,0 0 1))

```
-- TIN --
SELECT (g.gdump).path, ST_AsEWKT((g.gdump).geom) as wkt
FROM
(SELECT
```

```

    ST_Dump( ST_GeomFromEWKT('TIN (((
        0 0 0,
        0 0 1,
        0 1 0,
        0 0 0
    )), ((
        0 0 0,
        0 1 0,
        1 1 0,
        0 0 0
    ))
    )') ) AS gdump
) AS g;
-- result --
path |          wkt
-----+-----
{1}  | TRIANGLE((0 0 0,0 0 1,0 1 0,0 0 0))
{2}  | TRIANGLE((0 0 0,0 1 0,1 1 0,0 0 0))

```

**Siehe auch**

[geometry\\_dump](#), [ST\\_GeomFromEWKT](#), [ST\\_Dump](#), [ST\\_GeometryN](#), [ST\\_NumGeometries](#)

**7.4.7 ST\_DumpPoints**

`ST_DumpPoints` — Gibt eine Zusammenfassung des Inhalts einer Geometrie wieder.

**Synopsis**

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

**Beschreibung**

Eine Funktion, die eine Menge zurückliefert (SRF), die die Koordinaten (Eckpunkte) einer Geometrie extrahiert. Sie gibt einen Satz von [geometry\\_dump](#) Zeilen zurück, die jeweils eine Geometrie (*geom* Feld) und eine Reihe von Ganzzahlen (*path* Feld) enthalten.

- das Feld *geom* POINTs stellt die Koordinaten der gelieferten Geometrie dar.
- das Feld *path* (ein `integer[]`) ist ein Index, der die Koordinatenpositionen in den Elementen der gelieferten Geometrie aufzählt. Die Indizes sind 1-basiert. Für einen `LINestring` sind die Pfade beispielsweise `{i}` wobei *i* die *n*-te Koordinate im `LINestring` ist. Für ein `POLYGON` lauten die Pfade `{i, j}`, wobei *i* die Ringnummer ist (1 ist der äußere, die inneren Ringe folgen) und *j* die Koordinatenposition im Ring.

Um eine einzelne Geometrie zu erhalten, die die Koordinaten enthält, verwenden Sie [ST\\_Points](#).

Verbessert: 2.1.0 Höhere Geschwindigkeit. Reimplementiert als natives C.

Erweiterung: Mit 2.0.0 wurde die Unterstützung für polyedrische Oberflächen, Dreiecke und TIN eingeführt.

Verfügbarkeit: 1.5.0



Diese Methode unterstützt kreisförmige Strings und Kurven.



Diese Funktion unterstützt polyedrische Flächen.



Diese Funktion unterstützt Dreiecke und dreieckige unregelmäßige Netzoberflächen (TIN).



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

### Klassisches Auflösen einer Tabelle von LineStrings in Knoten

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

### Standard Beispiele



```
SELECT path, ST_AsText(geom)
FROM (
  SELECT (ST_DumpPoints(g.geom)).*
  FROM
    (SELECT
      'GEOMETRYCOLLECTION(
        POINT ( 0 1 ),
        LINESTRING ( 0 3, 3 4 ),
        POLYGON (( 2 0, 2 3, 0 2, 2 0 )),
        POLYGON (( 3 0, 3 3, 6 3, 6 0, 3 0 ),
          ( 5 1, 4 2, 5 2, 5 1 )),
        MULTIPOLYGON (
          (( 0 5, 0 8, 4 8, 4 5, 0 5 )),
          ( 1 6, 3 6, 2 7, 1 6 )),
          (( 5 4, 5 8, 6 7, 5 4 ))
        )
      ):::geometry AS geom
    ) AS g
  ) j;

path | st_astext
```

```

-----+-----
{1,1} | POINT(0 1)
{2,1} | POINT(0 3)
{2,2} | POINT(3 4)
{3,1,1} | POINT(2 0)
{3,1,2} | POINT(2 3)
{3,1,3} | POINT(0 2)
{3,1,4} | POINT(2 0)
{4,1,1} | POINT(3 0)
{4,1,2} | POINT(3 3)
{4,1,3} | POINT(6 3)
{4,1,4} | POINT(6 0)
{4,1,5} | POINT(3 0)
{4,2,1} | POINT(5 1)
{4,2,2} | POINT(4 2)
{4,2,3} | POINT(5 2)
{4,2,4} | POINT(5 1)
{5,1,1,1} | POINT(0 5)
{5,1,1,2} | POINT(0 8)
{5,1,1,3} | POINT(4 8)
{5,1,1,4} | POINT(4 5)
{5,1,1,5} | POINT(0 5)
{5,1,2,1} | POINT(1 6)
{5,1,2,2} | POINT(3 6)
{5,1,2,3} | POINT(2 7)
{5,1,2,4} | POINT(1 6)
{5,2,1,1} | POINT(5 4)
{5,2,1,2} | POINT(5 8)
{5,2,1,3} | POINT(6 7)
{5,2,1,4} | POINT(5 4)
(29 rows)

```

### Beispiele für polyedrische Oberflächen, TIN und Dreieck

```

-- Polyhedral surface cube --
SELECT (g.gdump).path, ST_AsEWKT((g.gdump).geom) as wkt
FROM
  (SELECT
    ST_DumpPoints(ST_GeomFromEWKT('POLYHEDRALSURFACE( ((0 0 0, 0 0 1, 0 1 1, 0 1 0, 0 0 ←
    0)),
    ((0 0 0, 0 1 0, 1 1 0, 1 0 0, 0 0 0)), ((0 0 0, 1 0 0, 1 0 1, 0 0 1, 0 0 0)),
    ((1 1 0, 1 1 1, 1 0 1, 1 0 0, 1 1 0)),
    ((0 1 0, 0 1 1, 1 1 1, 1 1 0, 0 1 0)), ((0 0 1, 1 0 1, 1 1 1, 0 1 1, 0 0 1)) )' ) AS gdump
  ) AS g;
-- result --
path | wkt
-----+-----
{1,1,1} | POINT(0 0 0)
{1,1,2} | POINT(0 0 1)
{1,1,3} | POINT(0 1 1)
{1,1,4} | POINT(0 1 0)
{1,1,5} | POINT(0 0 0)
{2,1,1} | POINT(0 0 0)
{2,1,2} | POINT(0 1 0)
{2,1,3} | POINT(1 1 0)
{2,1,4} | POINT(1 0 0)
{2,1,5} | POINT(0 0 0)
{3,1,1} | POINT(0 0 0)
{3,1,2} | POINT(1 0 0)
{3,1,3} | POINT(1 0 1)

```



```

{3,1,4} | POINT(0 0 1)
{3,1,5} | POINT(0 0 0)
{4,1,1} | POINT(1 1 0)
{4,1,2} | POINT(1 1 1)
{4,1,3} | POINT(1 0 1)
{4,1,4} | POINT(1 0 0)
{4,1,5} | POINT(1 1 0)
{5,1,1} | POINT(0 1 0)
{5,1,2} | POINT(0 1 1)
{5,1,3} | POINT(1 1 1)
{5,1,4} | POINT(1 1 0)
{5,1,5} | POINT(0 1 0)
{6,1,1} | POINT(0 0 1)
{6,1,2} | POINT(1 0 1)
{6,1,3} | POINT(1 1 1)
{6,1,4} | POINT(0 1 1)
{6,1,5} | POINT(0 0 1)
(30 rows)

```

```

-- Triangle --
SELECT (g.gdump).path, ST_AsText((g.gdump).geom) as wkt
FROM
  (SELECT
    ST_DumpPoints( ST_GeomFromEWKT('TRIANGLE ((
      0 0,
      0 9,
      9 0,
      0 0
    ))') ) AS gdump
  ) AS g;
-- result --
path | wkt
-----+-----
{1} | POINT(0 0)
{2} | POINT(0 9)
{3} | POINT(9 0)
{4} | POINT(0 0)

```

```

-- TIN --
SELECT (g.gdump).path, ST_AsEWKT((g.gdump).geom) as wkt
FROM
  (SELECT
    ST_DumpPoints( ST_GeomFromEWKT('TIN (((
      0 0 0,
      0 0 1,
      0 1 0,
      0 0 0
    )), ((
      0 0 0,
      0 1 0,
      1 1 0,
      0 0 0
    ))
  )') ) AS gdump
  ) AS g;
-- result --
path | wkt
-----+-----
{1,1,1} | POINT(0 0 0)
{1,1,2} | POINT(0 0 1)
{1,1,3} | POINT(0 1 0)

```

```
{1,1,4} | POINT(0 0 0)
{2,1,1} | POINT(0 0 0)
{2,1,2} | POINT(0 1 0)
{2,1,3} | POINT(1 1 0)
{2,1,4} | POINT(0 0 0)
(8 rows)
```

## Siehe auch

[geometry\\_dump](#), [ST\\_GeomFromEWKT](#), [ST\\_Dump](#), [ST\\_GeometryN](#), [ST\\_NumGeometries](#)

## 7.4.8 ST\_DumpSegments

`ST_DumpSegments` — Gibt eine Zusammenfassung des Inhalts einer Geometrie wieder.

### Synopsis

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

### Beschreibung

Eine Funktion, die eine Menge zurückgibt (SRF), die die Segmente einer Geometrie extrahiert. Sie gibt einen Satz von `geometry_dump` Zeilen zurück, die jeweils eine Geometrie (Feld `geom`) und eine Reihe von Ganzzahlen (Feld `path`) enthalten.

- the `geom` field `LINESTRINGS` represent the linear segments of the supplied geometry, while the `CIRCULARSTRINGS` represent the arc segments.
- Das Feld `path` (ein `integer[]`) ist ein Index, der die Positionen der Segmentstartpunkte in den Elementen der gelieferten Geometrie auflistet. Die Indizes sind 1-basiert. Zum Beispiel sind für einen `LINESTRING` die Pfade `{i}` wobei `i` der `n`-te Segmentstartpunkt im `LINESTRING` ist. Für ein `POLYGON` lauten die Pfade `{i, j}`, wobei `i` die Ringnummer ist (1 ist der äußere, die inneren Ringe folgen) und `j` die Position des Segment-Startpunkts im Ring.

Verfügbarkeit: 3.2.0



Diese Funktion unterstützt Dreiecke und dreieckige unregelmäßige Netzoberflächen (TIN).



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

### Standard Beispiele

```
SELECT path, ST_AsText(geom)
FROM (
  SELECT (ST_DumpSegments(g.geom)).*
  FROM (SELECT 'GEOMETRYCOLLECTION(
    LINESTRING(1 1, 3 3, 4 4),
    POLYGON((5 5, 6 6, 7 7, 5 5))
  )'::geometry AS geom
        ) AS g
) j;

 path      &#x2502;          st_astext
-----
{1,1}      &#x2502;  LINESTRING(1 1,3 3)
{1,2}      &#x2502;  LINESTRING(3 3,4 4)
```

```
{2,1,1} &#x2502; LINESTRING(5 5,6 6)
{2,1,2} &#x2502; LINESTRING(6 6,7 7)
{2,1,3} &#x2502; LINESTRING(7 7,5 5)
(5 rows)
```

### Beispiele für polyedrische Oberflächen, TIN und Dreieck

```
-- Triangle --
SELECT path, ST_AsText(geom)
FROM (
  SELECT (ST_DumpSegments(g.geom)).*
  FROM (SELECT 'TRIANGLE((
    0 0,
    0 9,
    9 0,
    0 0
  ))'::geometry AS geom
  ) AS g
) j;

path &#x2502;          st_astext
-----
{1,1} &#x2502; LINESTRING(0 0,0 9)
{1,2} &#x2502; LINESTRING(0 9,9 0)
{1,3} &#x2502; LINESTRING(9 0,0 0)
(3 rows)
```

```
-- TIN --
SELECT path, ST_AsEWKT(geom)
FROM (
  SELECT (ST_DumpSegments(g.geom)).*
  FROM (SELECT 'TIN(((
    0 0 0,
    0 0 1,
    0 1 0,
    0 0 0
  )), ((
    0 0 0,
    0 1 0,
    1 1 0,
    0 0 0
  ))
  )'::geometry AS geom
  ) AS g
) j;

path &#x2502;          st_asewkt
-----
{1,1,1} &#x2502; LINESTRING(0 0 0,0 0 1)
{1,1,2} &#x2502; LINESTRING(0 0 1,0 1 0)
{1,1,3} &#x2502; LINESTRING(0 1 0,0 0 0)
{2,1,1} &#x2502; LINESTRING(0 0 0,0 1 0)
{2,1,2} &#x2502; LINESTRING(0 1 0,1 1 0)
{2,1,3} &#x2502; LINESTRING(1 1 0,0 0 0)
(6 rows)
```

### Siehe auch

[geometry\\_dump](#), [ST\\_Collect](#), [ST\\_Dump](#), [ST\\_NumInteriorRing](#),

## 7.4.9 ST\_DumpRings

ST\_DumpRings — Gibt einen Satz von `geometry_dump` Zeilen für die äußeren und inneren Ringe eines Polygons zurück.

### Synopsis

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

### Beschreibung

Eine Funktion, die eine Menge zurückgibt (SRF), die die Ringe eines Polygons extrahiert. Sie gibt einen Satz von `geometry_dump` Zeilen zurück, die jeweils eine Geometrie (`geom` Feld) und eine Reihe von Ganzzahlen (`path` Feld) enthalten.

Das Feld `geom` enthält jeden Ring als POLYGON. Das Feld `path` ist ein ganzzahliges Feld der Länge 1, das den Polygonringindex enthält. Der äußere Ring (Schale) hat den Index 0. Die inneren Ringe (Löcher) haben Indizes von 1 und höher.



#### Note

Dies funktioniert nicht mit MULTIPOLYGONen. Verwenden Sie die Funktion bitte in Zusammenhang mit ST\_Dump um sie auf MULTIPOLYGONE anzuwenden.

Verfügbarkeit: PostGIS 1.1.3. Benötigt PostgreSQL 7.3 oder höher.



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

### Beispiele

Allgemeine Form der Abfrage.

```
SELECT polyTable.field1, polyTable.field1,
       (ST_DumpRings(polyTable.geom)).geom As geom
FROM polyTable;
```

Ein Polygon mit einem einzigen Loch.

```
SELECT path, ST_AsEWKT(geom) As geom
FROM ST_DumpRings(
    ST_GeomFromEWKT('POLYGON((-8149064 5133092 1,-8149064 5132986 1,-8148996 ↵
        5132839 1,-8148972 5132767 1,-8148958 5132508 1,-8148941 5132466 ↵
        1,-8148924 5132394 1,
        -8148903 5132210 1,-8148930 5131967 1,-8148992 5131978 1,-8149237 5132093 ↵
        1,-8149404 5132211 1,-8149647 5132310 1,-8149757 5132394 1,
        -8150305 5132788 1,-8149064 5133092 1),
        (-8149362 5132394 1,-8149446 5132501 1,-8149548 5132597 1,-8149695 5132675 ↵
        1,-8149362 5132394 1))')
) as foo;
```

path	geom
{0}	POLYGON((-8149064 5133092 1,-8149064 5132986 1,-8148996 5132839 1,-8148972 5132767 ↵ 1,-8148958 5132508 1,   -8148941 5132466 1,-8148924 5132394 1,   -8148903 5132210 1,-8148930 5131967 1,   -8148992 5131978 1,-8149237 5132093 1,   -8149404 5132211 1,-8149647 5132310 1,-8149757 5132394 1,-8150305 ↵ 5132788 1,-8149064 5133092 1))
{1}	POLYGON((-8149362 5132394 1,-8149446 5132501 1,   -8149548 5132597 1,-8149695 5132675 1,-8149362 5132394 1))

**Siehe auch**

[geometry\\_dump](#), [ST\\_GeomFromEWKT](#), [ST\\_Dump](#), [ST\\_GeometryN](#), [ST\\_NumGeometries](#)

**7.4.10 ST\_EndPoint**

`ST_EndPoint` — Gibt die Anzahl der Stützpunkte eines `ST_LineString` oder eines `ST_CircularString` zurück.

**Synopsis**

```
geometry ST_EndPoint(geometry g);
```

**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.



Diese Methode setzt die SQL/MM-Spezifikation um. SQL-MM 3: 7.1.4



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



Diese Methode unterstützt kreisförmige Strings und Kurven.

**Note**

Änderung: 2.0.0 unterstützt die Verarbeitung von `MultiLineString`'s die nur aus einer einzelnen Geometrie bestehen, nicht mehr. In früheren Versionen von PostGIS gab die Funktion bei einem aus einer einzelnen Linie bestehender `MultiLineString` den Anfangspunkt zurück. Ab 2.0.0 gibt sie nur `NULL` zurück, so wie bei jedem anderen `MultiLineString`. Die alte Verhaltensweise war undokumentiert, aber Anwender, die annahmen, dass Sie Ihre Daten als `LINESTRING` vorliegen haben, könnten in 2.0 dieses zurückgegebene `NULL` bemerken.

**Beispiele**

Einhüllende von Punkt und Linienzug.

```
postgis=# SELECT ST_AsText(ST_EndPoint('LINESTRING(1 1, 2 2, 3 3)::geometry));
st_astext
-----
POINT(3 3)
```

Endpunkt eines Nicht-LineString ist NULL

```
SELECT ST_EndPoint('POINT(1 1)::geometry') IS NULL AS is_null;
is_null
-----
t
```

Einhüllende von Punkt und Linienzug.

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

Gibt die Anzahl der Stützpunkte eines `ST_LineString` oder eines `ST_CircularString` zurück.

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

## Siehe auch

[ST\\_PointN](#), [ST\\_StartPoint](#)

## 7.4.11 ST\_Envelope

**ST\_Envelope** — Gibt eine Geometrie in doppelter Genauigkeit (float8) zurück, welche das Umgebungsrechteck der beigestellten Geometrie darstellt.

### Synopsis

geometry **ST\_Envelope**(geometry g1);

### Beschreibung

Gibt das kleinstmögliche Umgebungsrechteck der bereitgestellten Geometrie als Geometrie im Float8-Format zurück. Das Polygon wird durch die Eckpunkte des Umgebungsrechteckes beschrieben ((MINX, MINY), (MINX, MAXY), (MAXX, MAXY), (MAXX, MINY), (MINX, MINY)). (PostGIS fügt auch die ZMIN/ZMAX Koordinaten hinzu).

Spezialfälle (vertikale Linien, Punkte) geben eine Geometrie geringerer Dimension zurück als POLYGON, insbesondere POINT oder LINESTRING.

Verfügbarkeit: 1.5.0 Änderung der Verhaltensweise insofern, das die Ausgabe in Double Precision anstelle von Float4 erfolgt



Diese Methode implementiert die [OGC Simple Features Implementation Specification for SQL 1.1](#). s2.1.1.1



Diese Methode setzt die SQL/MM-Spezifikation um. SQL-MM 3: 5.1.19

### Beispiele

```
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 10000123333334.34545678, 1.0000001 1, 1.0000001 0, 0 0))'::geometry As geom) As foo;
```



*Einhüllende von Punkt und Linienzug.*

```
SELECT ST_AsText(ST_Envelope(
    ST_Collect(
        ST_GeomFromText('LINESTRING(55 75,125 150)'),
        ST_Point(20, 80)
    ) As wktenv);
wktenv
-----
POLYGON((20 75,20 150,125 150,125 75,20 75))
```

#### Siehe auch

[Box2D](#), [Box3D](#), [ST\\_OrientedEnvelope](#)

#### 7.4.12 ST\_ExteriorRing

`ST_ExteriorRing` — Gibt die Anzahl der inneren Ringe einer Polygongeometrie aus.

#### Synopsis

geometry `ST_ExteriorRing`(geometry a\_polygon);

## Beschreibung

Gibt einen Linienzug zurück, welcher den äußeren Ring der POLYGON Geometrie darstellt. Gibt NULL zurück wenn es sich bei der Geometrie um kein Polygon handelt.



### Note

Dies funktioniert nicht mit MULTIPOLYGONen. Verwenden Sie die Funktion bitte in Zusammenhang mit ST\_Dump um sie auf MULTIPOLYGONE anzuwenden.



Diese Methode implementiert die [OGC Simple Features Implementation Specification for SQL 1.1](#). 2.1.5.1



Diese Methode setzt die SQL/MM-Spezifikation um. SQL-MM 3: 8.2.3, 8.3.3



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

## Beispiele

```
--If you have a table of polygons
SELECT gid, ST_ExteriorRing(geom) AS ering
FROM sometable;

--If you have a table of MULTIPOLYGONs
--and want to return a MULTILINESTRING composed of the exterior rings of each polygon
SELECT gid, ST_Collect(ST_ExteriorRing(geom)) AS erings
      FROM (SELECT gid, (ST_Dump(geom)).geom As geom
            FROM sometable) As foo
GROUP BY gid;

--3d Example
SELECT ST_AsEWKT(
  ST_ExteriorRing(
    ST_GeomFromEWKT('POLYGON((0 0 1, 1 1 1, 1 2 1, 1 1 1, 0 0 1))')
  )
);

st_asewkt
-----
LINESTRING(0 0 1,1 1 1,1 2 1,1 1 1,0 0 1)
```

## Siehe auch

[ST\\_InteriorRingN](#), [ST\\_Boundary](#), [ST\\_NumInteriorRings](#)

### 7.4.13 ST\_GeometryN

ST\_GeometryN — Gibt den Geometrietyp des ST\_Geometry Wertes zurück.

#### Synopsis

geometry **ST\_GeometryN**(geometry geomA, integer n);



## Beschreibung

Gibt die auf 1-basierende n-te Geometrie zurück, wenn es sich bei der Geometrie um eine GEOMETRYCOLLECTION, (MULTI)POINT, (MULTI)LINestring, MULTICURVE oder (MULTI)POLYGON, POLYHEDRALSURFACE handelt. Anderenfalls wird NULL zurückgegeben.



### Note

Seit Version 0.8.0 basiert der Index auf 1, so wie in der OGC Spezifikation. Vorhergegangene Versionen waren 0-basiert.



### Note

Falls Sie alle Geometrien einer Geometrie entnehmen wollen, so ist ST\_Dump wesentlich leistungsfähiger und es funktioniert auch mit Einzelgeometrien.

Erweiterung: Mit 2.0.0 wurde die Unterstützung für polyedrische Oberflächen, Dreiecke und TIN eingeführt.

Änderung: 2.0.0 Vorangegangene Versionen geben bei Einzelgeometrien NULL zurück. Dies wurde geändert um die Geometrie für den ST\_GeometrieN(..,1) Fall zurückzugeben.



Diese Methode implementiert die [OGC Simple Features Implementation Specification for SQL 1.1](#).



Diese Methode setzt die SQL/MM-Spezifikation um. SQL-MM 3: 9.1.5



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



Diese Methode unterstützt kreisförmige Strings und Kurven.



Diese Funktion unterstützt polyedrische Flächen.



Diese Funktion unterstützt Dreiecke und dreieckige unregelmäßige Netzoberflächen (TIN).

## Standard Beispiele

```
--Extracting a subset of points from a 3d multipoint
SELECT n, ST_AsEWKT(ST_GeometryN(geom, n)) As geomewkt
FROM (
VALUES (ST_GeomFromEWKT('MULTIPOINT((1 2 7), (3 4 7), (5 6 7), (8 9 10))') ),
( ST_GeomFromEWKT('MULTICURVE(CIRCULARSTRING(2.5 2.5,4.5 2.5, 3.5 3.5), (10 11, 12 11))') )
)As foo(geom)
CROSS JOIN generate_series(1,100) n
WHERE n <= ST_NumGeometries(geom);
```

```
n |          geomewkt
---+-----
1 | POINT(1 2 7)
2 | POINT(3 4 7)
3 | POINT(5 6 7)
4 | POINT(8 9 10)
1 | CIRCULARSTRING(2.5 2.5,4.5 2.5,3.5 3.5)
2 | LINestring(10 11,12 11)
```

```
--Extracting all geometries (useful when you want to assign an id)
SELECT gid, n, ST_GeometryN(geom, n)
FROM sometable CROSS JOIN generate_series(1,100) n
WHERE n <= ST_NumGeometries(geom);
```

## Beispiele für polyedrische Oberflächen, TIN und Dreieck

```
-- Polyhedral surface example
-- Break a Polyhedral surface into its faces
SELECT ST_AsEWKT(ST_GeometryN(p_geom,3)) As geom_ewkt
FROM (SELECT ST_GeomFromEWKT('POLYHEDRALSURFACE(
((0 0 0, 0 0 1, 0 1 1, 0 1 0, 0 0 0)),
((0 0 0, 0 1 0, 1 1 0, 1 0 0, 0 0 0)),
((0 0 0, 1 0 0, 1 0 1, 0 0 1, 0 0 0)),
((1 1 0, 1 1 1, 1 0 1, 1 0 0, 1 1 0)),
((0 1 0, 0 1 1, 1 1 1, 1 1 0, 0 1 0)),
((0 0 1, 1 0 1, 1 1 1, 0 1 1, 0 0 1))
)') AS p_geom ) AS a;

          geom_ewkt
-----
POLYGON((0 0 0,1 0 0,1 0 1,0 0 1,0 0 0))
```

```
-- TIN --
SELECT ST_AsEWKT(ST_GeometryN(geom,2)) as wkt
FROM
  (SELECT
    ST_GeomFromEWKT('TIN (((
      0 0 0,
      0 0 1,
      0 1 0,
      0 0 0
    )), ((
      0 0 0,
      0 1 0,
      1 1 0,
      0 0 0
    ))
  )') AS geom
  ) AS g;
-- result --

          wkt
-----
TRIANGLE((0 0 0,0 1 0,1 1 0,0 0 0))
```

### Siehe auch

[ST\\_Dump](#), [ST\\_NumGeometries](#)

## 7.4.14 ST\_GeometryType

ST\_GeometryType — Gibt den Geometrietyp des ST\_Geometry Wertes zurück.

### Synopsis

```
text ST_GeometryType(geometry g1);
```

### Beschreibung

Gibt den Geometrietyp als Zeichenkette zurück. Z.B.: 'ST\_LineString', 'ST\_Polygon', 'ST\_MultiPolygon' etc. Diese Funktion unterscheidet sich von GeometryType(geometry) durch den Präfix ST\_ und dadurch, dass nicht angezeigt wird, ob die Geometrie eine Maßzahl besitzt.

Erweiterung: Mit 2.0.0 wurde die Unterstützung für polyedrische Oberflächen eingeführt.



Diese Methode setzt die SQL/MM-Spezifikation um. SQL-MM 3: 5.1.4



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



Diese Funktion unterstützt polyedrische Flächen.

## Beispiele

```
SELECT ST_GeometryType(ST_GeomFromText('LINESTRING(77.29 29.07,77.42 29.26,77.27 29.31,77.29 29.07)'));
--result
ST_LineString
```

```
SELECT ST_GeometryType(ST_GeomFromEWKT('POLYHEDRALSURFACE( ((0 0 0, 0 0 1, 0 1 1, 0 1 0, 0 0 0)),
((0 0 0, 0 1 0, 1 1 0, 1 0 0, 0 0 0)), ((0 0 0, 1 0 0, 1 0 1, 0 0 1, 0 0 0)),
((1 1 0, 1 1 1, 1 0 1, 1 0 0, 1 1 0)),
((0 1 0, 0 1 1, 1 1 1, 1 1 0, 0 1 0)), ((0 0 1, 1 0 1, 1 1 1, 0 1 1, 0 0 1))
) )'));
--result
ST_PolyhedralSurface
```

```
SELECT ST_GeometryType(ST_GeomFromEWKT('POLYHEDRALSURFACE( ((0 0 0, 0 0 1, 0 1 1, 0 1 0, 0 0 0)),
((0 0 0, 0 1 0, 1 1 0, 1 0 0, 0 0 0)), ((0 0 0, 1 0 0, 1 0 1, 0 0 1, 0 0 0)),
((1 1 0, 1 1 1, 1 0 1, 1 0 0, 1 1 0)),
((0 1 0, 0 1 1, 1 1 1, 1 1 0, 0 1 0)), ((0 0 1, 1 0 1, 1 1 1, 0 1 1, 0 0 1))
) )'));
--result
ST_PolyhedralSurface
```

```
SELECT ST_GeometryType(geom) as result
FROM
  (SELECT
    ST_GeomFromEWKT('TIN (((
      0 0 0,
      0 0 1,
      0 1 0,
      0 0 0
    )), ((
      0 0 0,
      0 1 0,
      1 1 0,
      0 0 0
    ))
  )') AS geom
) AS g;
result
-----
ST_Tin
```

## Siehe auch

[GeometryType](#)

### 7.4.15 ST\_HasArc

ST\_HasArc — Prüft, ob eine Geometrie einen Kreisbogen enthält

#### Synopsis

boolean **ST\_HasArc**(geometry geomA);

#### Beschreibung

Gibt den Wert TRUE zurück, falls es sich bei der Geometrie um eine leere GeometryCollection, Polygon, Point etc. handelt.

Verfügbarkeit: 1.2.2



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



Diese Methode unterstützt kreisförmige Strings und Kurven.

#### Beispiele

```
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\\_PointN](#)

### 7.4.16 ST\_InteriorRingN

ST\_InteriorRingN — Gibt die Anzahl der inneren Ringe einer Polygoneometrie aus.

#### Synopsis

geometry **ST\_InteriorRingN**(geometry a\_polygon, integer n);

#### Beschreibung

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.



#### Note

Dies funktioniert nicht mit MULTIPOLYGONen. Verwenden Sie die Funktion bitte in Zusammenhang mit ST\_Dump um sie auf MULTIPOLYGONe anzuwenden.



Diese Methode implementiert die [OGC Simple Features Implementation Specification for SQL 1.1](#).



Diese Methode setzt die SQL/MM-Spezifikation um. SQL-MM 3: 8.2.6, 8.3.5



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

## Beispiele

```
SELECT ST_AsText(ST_InteriorRingN(geom, 1)) As geom
FROM (SELECT ST_BuildArea(
        ST_Collect(ST_Buffer(ST_Point(1,2), 20,3),
                ST_Buffer(ST_Point(1, 2), 10,3))) As geom
      ) as foo;
```

## Siehe auch

[ST\\_ExteriorRing](#), [ST\\_M](#), [ST\\_X](#), [ST\\_Y](#), [ST\\_ZMax](#), [ST\\_ZMin](#)

## 7.4.17 ST\_NumCurves

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

### Synopsis

integer `ST_NumCurves`(geometry a\_compoundcurve);

### Beschreibung

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



Diese Methode setzt die SQL/MM-Spezifikation um. SQL-MM 3: 8.2.6, 8.3.5



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

## Beispiele

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

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

## Siehe auch

[ST\\_CurveN](#), [ST\\_Dump](#), [ST\\_ExteriorRing](#), [ST\\_NumInteriorRings](#), [ST\\_NumGeometries](#)

## 7.4.18 ST\_CurveN

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

### Synopsis

geometry `ST_CurveN`(geometry a\_compoundcurve, integer index);

## Beschreibung

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



Diese Methode setzt die SQL/MM-Spezifikation um. SQL-MM 3: 8.2.6, 8.3.5



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

## Beispiele

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

## Siehe auch

[ST\\_NumCurves](#), [ST\\_Dump](#), [ST\\_ExteriorRing](#), [ST\\_NumInteriorRings](#), [ST\\_NumGeometries](#)

### 7.4.19 ST\_IsClosed

**ST\_IsClosed** — Gibt den Wert TRUE zurück, wenn die Anfangs- und Endpunkte des LINESTRING's zusammenfallen. Bei polyedrischen Oberflächen, wenn sie geschlossen (volumetrisch) sind.

## Synopsis

boolean **ST\_IsClosed**(geometry g);

## Beschreibung

Gibt den Wert TRUE zurück, wenn die Anfangs- und Endpunkte des LINESTRING's zusammenfallen. Bei polyedrischen Oberflächen wird angezeigt, ob die Oberfläche eine Fläche (offen) oder ein Volumen (geschlossen) beschreibt.



Diese Methode implementiert die [OGC Simple Features Implementation Specification for SQL 1.1](#).



Diese Methode setzt die SQL/MM-Spezifikation um. SQL-MM 3: 7.1.5, 9.3.3



### Note

SQL-MM definiert das Ergebnis von `ST_IsClosed(NULL)` als 0, während PostGIS NULL zurückgibt.



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



Diese Methode unterstützt kreisförmige Strings und Kurven.

Erweiterung: Mit 2.0.0 wurde die Unterstützung für polyedrische Oberflächen eingeführt.



Diese Funktion unterstützt polyedrische Flächen.

**Beispiele für Linienzüge und Punkte**

```

postgis=# SELECT ST_IsClosed('LINESTRING(0 0, 1 1)::geometry);
 st_isclosed
-----
f
(1 row)

postgis=# SELECT ST_IsClosed('LINESTRING(0 0, 0 1, 1 1, 0 0)::geometry);
 st_isclosed
-----
t
(1 row)

postgis=# SELECT ST_IsClosed('MULTILINESTRING((0 0, 0 1, 1 1, 0 0),(0 0, 1 1))::geometry);
 st_isclosed
-----
f
(1 row)

postgis=# SELECT ST_IsClosed('POINT(0 0)::geometry);
 st_isclosed
-----
t
(1 row)

postgis=# SELECT ST_IsClosed('MULTIPOINT((0 0), (1 1))::geometry);
 st_isclosed
-----
t
(1 row)

```

**Beispiel für eine polyedrische Oberfläche**

```

-- A cube --
SELECT ST_IsClosed(ST_GeomFromEWKT('POLYHEDRALSURFACE( ((0 0 0, 0 0 1, 0 1 ←
1, 0 1 0, 0 0 0)),
((0 0 0, 0 1 0, 1 1 0, 1 0 0, 0 0 0)), ((0 0 0, 1 0 0, 1 0 1, 0 0 1, 0 0 0) ←
),
((1 1 0, 1 1 1, 1 0 1, 1 0 0, 1 1 0)),
((0 1 0, 0 1 1, 1 1 1, 1 1 0, 0 1 0)), ((0 0 1, 1 0 1, 1 1 1, 0 1 1, 0 0 1) ←
) )'));

 st_isclosed
-----
t

-- Same as cube but missing a side --
SELECT ST_IsClosed(ST_GeomFromEWKT('POLYHEDRALSURFACE( ((0 0 0, 0 0 1, 0 1 1, 0 1 0, 0 0 ←
0)),
((0 0 0, 0 1 0, 1 1 0, 1 0 0, 0 0 0)), ((0 0 0, 1 0 0, 1 0 1, 0 0 1, 0 0 0) ←
),
((1 1 0, 1 1 1, 1 0 1, 1 0 0, 1 1 0)),
((0 1 0, 0 1 1, 1 1 1, 1 1 0, 0 1 0)) '));

 st_isclosed
-----
f

```

**Siehe auch**[ST\\_IsRing](#)**7.4.20 ST\_IsCollection**

`ST_IsCollection` — Gibt den Wert `TRUE` zurück, falls es sich bei der Geometrie um eine leere `GeometryCollection`, `Polygon`, `Point` etc. handelt.

**Synopsis**

```
boolean ST_IsCollection(geometry g);
```

**Beschreibung**

Gibt den Wert `TRUE` zurück, wenn der Geometrietyp einer der folgenden 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.



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



Diese Methode unterstützt kreisförmige Strings und Kurven.

**Beispiele**

```
postgis=# SELECT ST_IsCollection('LINESTRING(0 0, 1 1)::geometry);
 st_iscollection
-----
 f
(1 row)

postgis=# SELECT ST_IsCollection('MULTIPOINT EMPTY)::geometry);
 st_iscollection
-----
 t
(1 row)

postgis=# SELECT ST_IsCollection('MULTIPOINT((0 0))::geometry);
 st_iscollection
-----
 t
(1 row)

postgis=# SELECT ST_IsCollection('MULTIPOINT((0 0), (42 42))::geometry);
 st_iscollection
-----
```



```
t
(1 row)

postgis=# SELECT ST_IsCollection('GEOMETRYCOLLECTION(POINT(0 0))'::geometry);
 st_iscollection
-----
t
(1 row)
```

**Siehe auch**[ST\\_NumGeometries](#)**7.4.21 ST\_IsEmpty**

ST\_IsEmpty — Prüft, ob eine Geometrie leer ist.

**Synopsis**

boolean **ST\_IsEmpty**(geometry geomA);

**Beschreibung**

Gibt den Wert TRUE zurück, wenn es sich um eine leere Geometrie handelt. Falls TRUE, dann repräsentiert diese Geometrie eine leere GeometryCollection, Polygon, Point etc.

**Note**

SQL-MM gibt vor, daß das Ergebnis von ST\_IsEmpty(NULL) der Wert 0 ist, während PostGIS den Wert NULL zurückgibt.



Diese Methode implementiert die [OGC Simple Features Implementation Specification for SQL 1.1](#). s2.1.1.1



Diese Methode setzt die SQL/MM-Spezifikation um. SQL-MM 3: 5.1.7



Diese Methode unterstützt kreisförmige Strings und Kurven.

**Warning**

Änderung: 2.0.0 - In Vorgängerversionen von PostGIS war ST\_GeomFromText('GEOMETRYCOLLECTION(EMPTY)') erlaubt. Um eine bessere Übereinstimmung mit der SQL/MM Norm zu erreichen, ist dies nun nicht mehr gestattet.

**Beispiele**

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

SELECT ST_IsEmpty(ST_GeomFromText('POLYGON EMPTY'));
```

```

st_isempty
-----
t
(1 row)

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

st_isempty
-----
f
(1 row)

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

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

```

## 7.4.22 ST\_IsPolygonCCW

**ST\_IsPolygonCCW** — Gibt TRUE zurück, wenn alle äußeren Ringe gegen den Uhrzeigersinn orientiert sind und alle inneren Ringe im Uhrzeigersinn ausgerichtet sind.

### Synopsis

boolean **ST\_IsPolygonCCW** ( geometry geom );

### Beschreibung

Gibt TRUE zurück, wenn für alle Bestandteile der angegebenen Geometrie gilt: die äußeren Ringe sind gegen den Uhrzeigersinn und die inneren Ringe im Uhrzeigersinn ausgerichtet.

Gibt TRUE zurück, wenn die Geometrie keine Polygonbestandteile aufweist.



#### 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.

Verfügbarkeit: 2.4.0



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



Diese Funktion unterstützt M-Koordinaten.

**Siehe auch**

[ST\\_ForcePolygonCW](#) , [ST\\_ForcePolygonCCW](#) , [ST\\_IsPolygonCW](#)

**7.4.23 ST\_IsPolygonCW**

`ST_IsPolygonCW` — Gibt den Wert TRUE zurück, wenn alle äußeren Ringe im Uhrzeigersinn und alle inneren Ringe gegen den Uhrzeigersinn ausgerichtet sind.

**Synopsis**

boolean `ST_IsPolygonCW` ( geometry geom );

**Beschreibung**

Gibt den Wert TRUE zurück, wenn für alle Polygonbestandteile der eingegebenen Geometrie gilt: die äußeren Ringe sind im Uhrzeigersinn orientiert, die inneren Ringe entgegen dem Uhrzeigersinn.

Gibt TRUE zurück, wenn die Geometrie keine Polygonbestandteile aufweist.

**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.

Verfügbarkeit: 2.4.0



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



Diese Funktion unterstützt M-Koordinaten.

**Siehe auch**

[ST\\_ForcePolygonCW](#) , [ST\\_ForcePolygonCCW](#) , [ST\\_IsPolygonCW](#)

**7.4.24 ST\_IsRing**

`ST_IsRing` — Prüft, ob ein LineString geschlossen und einfach ist.

**Synopsis**

boolean `ST_IsRing`(geometry g);

## Beschreibung

Liefert TRUE wenn dieser LINESTRING sowohl **ST\_IsClosed** (`ST_StartPoint(g) ~= ST_Endpoint(g)`) als auch **ST\_IsSimple** (schneidet sich nicht selbst) ist.



Diese Methode implementiert die **OGC Simple Features Implementation Specification for SQL 1.1**. 2.1.5.1



Diese Methode setzt die SQL/MM-Spezifikation um. SQL-MM 3: 7.1.6



### Note

SQL-MM gibt vor, daß das Ergebnis von `ST_IsRing(NULL)` der Wert 0 sein soll, während PostGIS den Wert NULL zurückgibt.

## Beispiele

```
SELECT ST_IsRing(geom), ST_IsClosed(geom), ST_IsSimple(geom)
FROM (SELECT 'LINESTRING(0 0, 0 1, 1 1, 1 0, 0 0)::geometry AS geom) AS foo;
  st_isring | st_isclosed | st_issimple
-----+-----+-----
t          | t          | t
(1 row)
```

```
SELECT ST_IsRing(geom), ST_IsClosed(geom), ST_IsSimple(geom)
FROM (SELECT 'LINESTRING(0 0, 0 1, 1 0, 1 1, 0 0)::geometry AS geom) AS foo;
  st_isring | st_isclosed | st_issimple
-----+-----+-----
f          | t          | f
(1 row)
```

## Siehe auch

[ST\\_IsClosed](#), [ST\\_IsSimple](#), [ST\\_StartPoint](#), [ST\\_EndPoint](#)

### 7.4.25 ST\_IsSimple

**ST\_IsSimple** — Gibt den Wert (TRUE) zurück, wenn die Geometrie keine irregulären Stellen, wie Selbstüberschneidungen oder Selbstberührungen, aufweist.

## Synopsis

boolean **ST\_IsSimple**(geometry geomA);

## Beschreibung

Gibt TRUE zurück, wenn keine regelwidrigen geometrischen Merkmale, wie Geometrien die sich selbst kreuzen oder berühren, auftreten. Für weiterführende Information zur OGC-Definition von Simplität und Gültigkeit von Geometrien, siehe "[Ensuring OpenGIS compliance of geometries](#)"



### Note

SQL-MM definiert das Ergebnis von `ST_IsSimple(NULL)` als 0, während PostGIS NULL zurückgibt.

- ✔ Diese Methode implementiert die [OGC Simple Features Implementation Specification for SQL 1.1](#). s2.1.1.1
- ✔ Diese Methode setzt die SQL/MM-Spezifikation um. SQL-MM 3: 5.1.8
- ✔ Diese Funktion unterstützt 3d und lässt den Z-Index nicht fallen.

### Beispiele

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

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

### Siehe auch

[ST\\_IsValid](#)

## 7.4.26 ST\_M

ST\_M — Gibt die M-Koordinate eines Punktes zurück.

### Synopsis

```
float ST_M(geometry a_point);
```

### Beschreibung

Gibt die M-Koordinate des Punktes zurück, oder NULL wenn keine vorhanden ist. Der Einabewert muss ein Punkt sein.



#### Note

Dies ist (noch) kein Teil der OGC Spezifikation, wird aber hier aufgeführt um die Liste von Funktionen zum Auslesen von Punktkoordinaten zu vervollständigen.

- ✔ Diese Methode implementiert die [OGC Simple Features Implementation Specification for SQL 1.1](#).
- ✔ Diese Methode setzt die SQL/MM-Spezifikation um.
- ✔ Diese Funktion unterstützt 3d und lässt den Z-Index nicht fallen.

**Beispiele**

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

**Siehe auch**

[ST\\_GeomFromEWKT](#), [ST\\_X](#), [ST\\_Y](#), [ST\\_Z](#)

**7.4.27 ST\_MemSize**

ST\_MemSize — Gibt den Geometrietyp des ST\_Geometry Wertes zurück.

**Synopsis**

integer **ST\_MemSize**(geometry geomA);

**Beschreibung**

Gibt den Geometrietyp des ST\_Geometry Wertes zurück.

Dies ergänzt die in PostgreSQL eingebauten [Datenbankobjektfunktionen](#) `pg_column_size`, `pg_size_pretty`, `pg_relation_size`, `pg_total_relation_size`.

**Note**

`pg_relation_size`, das die Bytegröße einer Tabelle angibt, kann eine geringere Bytegröße als `ST_MemSize` zurückgeben. Der Grund dafür ist, dass `pg_relation_size` den Beitrag von TOAST-Tabellen nicht berücksichtigt und große Geometrien in TOAST-Tabellen gespeichert werden.

`pg_total_relation_size` - schließt die Tabelle, die TOAST-Tabellen und die Indizes mit ein.

`pg_column_size` gibt zurück, wie viel Platz eine Geometrie in einer Spalte unter Berücksichtigung der Komprimierung einnehmen würde, kann also niedriger sein als `ST_MemSize`



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



Diese Methode unterstützt kreisförmige Strings und Kurven.



Diese Funktion unterstützt polyedrische Flächen.



Diese Funktion unterstützt Dreiecke und dreieckige unregelmäßige Netzoberflächen (TIN).

Geändert: 2.2.0 Name geändert in `ST_MemSize`, um der Namenskonvention zu folgen.

**Beispiele**

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

totgeomsum	bossum	perbos
1522 kB	30 kB	1.99

```
SELECT ST_MemSize(ST_GeomFromText('CIRCULARSTRING(220268 150415,220227 150505,220227 150406)'));
---
73

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

## 7.4.28 ST\_NDims

ST\_NDims — Gibt die Dimension der Koordinaten für den Wert von ST\_Geometry zurück.

### Synopsis

```
integer ST_NDims(geometry g1);
```

### Beschreibung

Gibt die Koordinatendimension der Geometrie zurück. PostGIS unterstützt 2- (x,y), 3- (x,y,z) oder 2D mit Kilometrierung - x,y,m, und 4- dimensionalen Raum - 3D mit Kilometrierung x,y,z,m .



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

### Beispiele

```
SELECT ST_NDims(ST_GeomFromText('POINT(1 1)')) As d2point,
ST_NDims(ST_GeomFromEWKT('POINT(1 1 2)')) As d3point,
ST_NDims(ST_GeomFromEWKT('POINTM(1 1 0.5)')) As d2pointm;
```

d2point	d3point	d2pointm
2	3	3

### Siehe auch

[ST\\_CoordDim](#), [ST\\_Dimension](#), [ST\\_GeomFromEWKT](#)

## 7.4.29 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



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



Diese Methode unterstützt kreisförmige Strings und Kurven.



Diese Funktion unterstützt polyedrische Flächen.

### Beispiele

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

--Polygon in 3D space
SELECT ST_NPoints(ST_GeomFromEWKT('LINESTRING(77.29 29.07 1,77.42 29.26 0,77.27 29.31 ↵
    -1,77.29 29.07 3)'));
--result
4
```

### Siehe auch

[ST\\_NumPoints](#)

## 7.4.30 ST\_NRings

ST\_NRings — Gibt die Anzahl der inneren Ringe einer Polygoneometrie aus.

### Synopsis

```
integer ST_NRings(geometry geomA);
```



## Beschreibung

Wenn es sich bei der Geometrie um ein Polygon oder um ein MultiPolygon handelt, wird die Anzahl der Ringe zurückgegeben. Anders als NumInteriorRings werden auch die äußeren Ringe gezählt.



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



Diese Methode unterstützt kreisförmige Strings und Kurven.

## Beispiele

```
SELECT ST_NRings(geom) As Nrings, ST_NumInteriorRings(geom) As ninterrings
      FROM (SELECT ST_GeomFromText('POLYGON((1 2, 3 4, 5 6, 1 2))') As geom) As foo;
```

nrings	ninterrings
1	0

(1 row)

## Siehe auch

[ST\\_NumInteriorRings](#)

### 7.4.31 ST\_NumGeometries

ST\_NumGeometries — Gibt die Anzahl der Punkte einer Geometrie zurück. Funktioniert für alle Geometrien.

## Synopsis

integer **ST\_NumGeometries**(geometry geom);

## Beschreibung

Gibt die Anzahl der Elemente in einer Geometriesammlung (GEOMETRYCOLLECTION oder MULTI\*) zurück. Für nicht leere atomare Geometrien wird 1 zurückgegeben. Für leere Geometrien wird 0 zurückgegeben.

Erweiterung: Mit 2.0.0 wurde die Unterstützung für polyedrische Oberflächen, Dreiecke und TIN eingeführt.

Änderung: 2.0.0 Bei früheren Versionen wurde NULL zurückgegeben, wenn die Geometrie nicht vom Typ GEOMETRYCOLLECTION/MULTI war. 2.0.0+ gibt nun 1 für Einzelgeometrien, wie POLYGON, LINESTRING, POINT zurück.



Diese Methode setzt die SQL/MM-Spezifikation um. SQL-MM 3: 9.1.4



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



Diese Funktion unterstützt polyedrische Flächen.



Diese Funktion unterstützt Dreiecke und dreieckige unregelmäßige Netzoberflächen (TIN).

**Beispiele**

```

--Prior versions would have returned NULL for this -- in 2.0.0 this returns 1
SELECT ST_NumGeometries(ST_GeomFromText('LINESTRING(77.29 29.07,77.42 29.26,77.27 ↔
    29.31,77.29 29.07)'));
--result
1

--Geometry Collection Example - multis count as one geom in a collection
SELECT ST_NumGeometries(ST_GeomFromEWKT('GEOMETRYCOLLECTION(MULTIPOINT((-2 3),(-2 2)),
LINESTRING(5 5 ,10 10),
POLYGON((-7 4.2,-7.1 5,-7.1 4.3,-7 4.2))'));
--result
3

```

**Siehe auch**

[ST\\_GeometryN](#), [ST\\_Multi](#)

**7.4.32 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.



Diese Methode setzt die SQL/MM-Spezifikation um. 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**

```

--If you have a regular polygon
SELECT gid, field1, field2, ST_NumInteriorRings(geom) AS numholes
FROM sometable;

--If you have multipolygons
--And you want to know the total number of interior rings in the MULTIPOLYGON
SELECT gid, field1, field2, SUM(ST_NumInteriorRings(geom)) AS numholes
FROM (SELECT gid, field1, field2, (ST_Dump(geom)).geom As geom
      FROM sometable) As foo
GROUP BY gid, field1,field2;

```

**Siehe auch**

[ST\\_NumInteriorRing](#), [ST\\_PointN](#)

### 7.4.33 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](#), [ST\\_PointN](#)

### 7.4.34 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



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



Diese Methode implementiert die [OGC Simple Features Implementation Specification for SQL 1.1](#).



Diese Methode setzt die SQL/MM-Spezifikation um. SQL-MM ISO/IEC 13249-3: 8.5



Diese Funktion unterstützt polyedrische Flächen.

#### Beispiele

```
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](#)

### 7.4.35 ST\_NumPoints

ST\_NumPoints — Gibt die Anzahl der Stützpunkte eines ST\_LineString oder eines ST\_CircularString zurück.

#### Synopsis

```
integer ST_NumPoints(geometry g1);
```

#### Beschreibung

Gibt die Anzahl der Stützpunkte eines ST\_LineString oder eines ST\_CircularString zurück. Vor 1.4 funktionierte dies nur mit ST\_LineString, wie von der Spezifikation festgelegt. Ab 1.4 aufwärts handelt es sich um einen Alias für ST\_NPoints, das die Anzahl der Knoten nicht nur für Linienzüge ausgibt. Erwägen Sie stattdessen die Verwendung von ST\_NPoints, das vielseitig ist und mit vielen Geometrietypen funktioniert.



Diese Methode implementiert die [OGC Simple Features Implementation Specification for SQL 1.1](#).



Diese Methode setzt die SQL/MM-Spezifikation um. SQL-MM 3: 7.2.4

#### Beispiele

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

#### Siehe auch

[ST\\_NPoints](#)

### 7.4.36 ST\_PatchN

ST\_PatchN — Gibt den Geometrietyp des ST\_Geometry Wertes zurück.

#### Synopsis

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

#### Beschreibung

>Gibt die auf 1-basierende n-te Geometrie (Masche) zurück, wenn es sich bei der Geometrie um ein POLYHEDRALSURFACE, oder ein POLYHEDRALSURFACEM handelt. Anderenfalls wird NULL zurückgegeben. Gibt bei polyedrischen Oberflächen das selbe Ergebnis wie ST\_GeometryN. Die Verwendung von ST\_GeometryN ist schneller.



#### Note

Der Index ist auf 1 basiert.

**Note**

Falls Sie alle Geometrien einer Geometrie entnehmen wollen, so ist `ST_Dump` wesentlich leistungsfähiger.

Verfügbarkeit: 2.0.0



Diese Methode setzt die SQL/MM-Spezifikation um. SQL-MM ISO/IEC 13249-3: 8.5



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



Diese Funktion unterstützt polyedrische Flächen.

**Beispiele**

```
--Extract the 2nd face of the polyhedral surface
SELECT ST_AsEWKT(ST_PatchN(geom, 2)) As geomewkt
FROM (
VALUES (ST_GeomFromEWKT('POLYHEDRALSURFACE( ((0 0 0, 0 0 1, 0 1 1, 0 1 0, 0 0 0)),
      ((0 0 0, 0 1 0, 1 1 0, 1 0 0, 0 0 0)), ((0 0 0, 1 0 0, 1 0 1, 0 0 1, 0 0 0)),
      ((1 1 0, 1 1 1, 1 0 1, 1 0 0, 1 1 0)),
      ((0 1 0, 0 1 1, 1 1 1, 1 1 0, 0 1 0)), ((0 0 1, 1 0 1, 1 1 1, 0 1 1, 0 0 1)) )' ) ) ←
      As foo(geom);

      geomewkt
-----+-----
POLYGON((0 0 0,0 1 0,1 1 0,1 0 0,0 0 0))
```

**Siehe auch**

[ST\\_AsEWKT](#), [ST\\_GeomFromEWKT](#), [ST\\_Dump](#), [ST\\_GeometryN](#), [ST\\_NumGeometries](#)

**7.4.37 ST\_PointN**

`ST_PointN` — Gibt die Anzahl der Stützpunkte eines `ST_LineString` oder eines `ST_CircularString` zurück.

**Synopsis**

geometry **ST\_PointN**(geometry a\_linestring, integer n);

**Beschreibung**

Gibt den n-ten Punkt des ersten `LineString`'s oder des kreisförmigen `LineStrings`'s einer Geometrie zurück. Negative Werte werden rückwärts, vom Ende des `LineString`'s her gezählt, sodass -1 der Endpunkt ist. Gibt NULL aus, wenn die Geometrie keinen `LineString` enthält.

**Note**

Seit Version 0.8.0 ist der Index 1-basiert, so wie in der OGC Spezifikation. Rückwärtiges Indizieren (negativer Index) findet sich nicht in der OGC Spezifikation. Vorhergegangene Versionen waren 0-basiert.

**Note**

Falls Sie den n-ten Punkt eines jeden LineString's in einem MultiLineString wollen, nutzen Sie diese Funktion gemeinsam mit ST\_Dump.



Diese Methode implementiert die [OGC Simple Features Implementation Specification for SQL 1.1](#).



Diese Methode setzt die SQL/MM-Spezifikation um. SQL-MM 3: 7.2.5, 7.3.5



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



Diese Methode unterstützt kreisförmige Strings und Kurven.

**Note**

Änderung: 2.0.0 arbeitet nicht mehr mit MultiLineString's, die nur eine einzelne Geometrie enthalten. In früheren Versionen von PostGIS gab die Funktion bei einem, aus einer einzelnen Linie bestehender MultiLineString, den Anfangspunkt zurück. Ab 2.0.0 wird, so wie bei jedem anderen MultiLineString auch, NULL zurückgegeben.

Änderung: 2.3.0 : negatives Indizieren verfügbar (-1 entspricht dem Endpunkt)

**Beispiele**

```
-- Extract all POINTs from a LINESTRING
SELECT ST_AsText(
  ST_PointN(
    column1,
    generate_series(1, ST_NPoints(column1))
  ))
FROM ( VALUES ('LINESTRING(0 0, 1 1, 2 2)::geometry') ) AS foo;

st_astext
-----
POINT(0 0)
POINT(1 1)
POINT(2 2)
(3 rows)

--Example circular string
SELECT ST_AsText(ST_PointN(ST_GeomFromText('CIRCULARSTRING(1 2, 3 2, 1 2)'), 2));

st_astext
-----
POINT(3 2)
(1 row)

SELECT ST_AsText(f)
FROM ST_GeomFromText('LINESTRING(0 0 0, 1 1 1, 2 2 2)') AS g
,ST_PointN(g, -2) AS f; -- 1 based index

st_astext
-----
POINT Z (1 1 1)
(1 row)
```

**Siehe auch**[ST\\_NPoints](#)**7.4.38 ST\_Points**

`ST_Points` — Gibt einen `MultiPoint` zurück, welcher alle Koordinaten einer Geometrie enthält.

**Synopsis**

```
geometry ST_Points( geometry geom );
```

**Beschreibung**

Gibt einen `MultiPoint` zurück, der alle Koordinaten einer Geometrie enthält. Doppelte Punkte werden beibehalten, einschließlich der Start- und Endpunkte von Ringgeometrien. (Falls gewünscht, können doppelte Punkte durch den Aufruf von [ST\\_RemoveRepeatedPoints](#) für das Ergebnis entfernt werden).

Um Informationen über die Position der einzelnen Koordinaten in der übergeordneten Geometrie zu erhalten, verwenden Sie [ST\\_DumpPoints](#).

M- und Z-Koordinaten werden beibehalten, falls vorhanden.



Diese Methode unterstützt kreisförmige Strings und Kurven.



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

Verfügbarkeit: 2.3.0

**Beispiele**

```
SELECT ST_AsText(ST_Points('POLYGON Z ((30 10 4,10 30 5,40 40 6, 30 10))'));  
  
--result  
MULTIPOINT Z ((30 10 4),(10 30 5),(40 40 6),(30 10 4))
```

**Siehe auch**[ST\\_RemoveRepeatedPoints](#), [ST\\_PointN](#)**7.4.39 ST\_StartPoint**

`ST_StartPoint` — Gibt den ersten Punkt eines `LineString` 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.



Diese Methode setzt die SQL/MM-Spezifikation um. SQL-MM 3: 7.1.3



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



Diese Methode unterstützt kreisförmige Strings und Kurven.

### Note

Verbessert: 3.2.0 gibt einen Punkt für alle Geometrien zurück. Vorheriges Verhalten gibt NULLs zurück, wenn die Eingabe kein LineString war.



Änderung: 2.0.0 unterstützt die Verarbeitung von MultiLineString's die nur aus einer einzelnen Geometrie bestehen, nicht mehr. In früheren Versionen von PostGIS gab die Funktion bei einem aus einer einzelnen Linie bestehender MultiLineString den Anfangspunkt zurück. Ab 2.0.0 gibt sie nur NULL zurück, so wie bei jedem anderen MultiLineString. Die alte Verhaltensweise war undokumentiert, aber Anwender, die annahmen, dass Sie Ihre Daten als `LINESTRING` vorliegen haben, könnten in 2.0 dieses zurückgegebene NULL bemerken.

## Beispiele

### Startpunkt eines LineString

```
SELECT ST_AsText(ST_StartPoint('LINESTRING(0 1, 0 2)::geometry'));
 st_astext
-----
POINT(0 1)
```

### Startpunkt eines Nicht-LineString ist NULL

```
SELECT ST_StartPoint('POINT(0 1)::geometry') IS NULL AS is_null;
 is_null
-----
t
```

### Startpunkt einer 3D-LinieString

```
SELECT ST_AsEWKT(ST_StartPoint('LINESTRING(0 1 1, 0 2 2)::geometry'));
 st_asewkt
-----
POINT(0 1 1)
```

Gibt die Anzahl der Stützpunkte eines `ST_LineString` oder eines `ST_CircularString` zurück.

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

## Siehe auch

[ST\\_EndPoint](#), [ST\\_PointN](#)



## 7.4.40 ST\_Summary

ST\_Summary — Gibt eine Zusammenfassung des Inhalts einer Geometrie wieder.

### Synopsis

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

### Beschreibung

Gibt eine Zusammenfassung des Inhalts einer Geometrie wieder.

Die Bedeutung der Flags, welche in eckigen Klammern hinter dem Geometrietyp angezeigt werden, ist wie folgt:

- M: besitzt eine M-Ordinate
- Z: besitzt eine Z-Ordinate
- B: besitzt ein zwischengespeichertes Umgebungsrechteck
- G: ist geodätisch (Geographie)
- S: besitzt ein räumliches Koordinatenreferenzsystem



Diese Methode unterstützt kreisförmige Strings und Kurven.



Diese Funktion unterstützt polyedrische Flächen.



Diese Funktion unterstützt Dreiecke und dreieckige unregelmäßige Netzoberflächen (TIN).

Verfügbarkeit: 1.2.2

Erweiterung: 2.0.0 Unterstützung für geographische Koordinaten hinzugefügt

Erweiterung: 2.1.0 S-Flag, diese zeigt an ob das Koordinatenreferenzsystem bekannt ist

Erweiterung: 2.2.0 Unterstützung für TIN und Kurven

### Beispiele

```
=# SELECT ST_Summary(ST_GeomFromText('LINESTRING(0 0, 1 1)')) as geom,
          ST_Summary(ST_GeogFromText('POLYGON((0 0, 1 1, 1 2, 1 1, 0 0))')) geog;
          geom                |                geog
-----+-----
 LineString[B] with 2 points | Polygon[BGS] with 1 rings
                               | ring 0 has 5 points
                               :
(1 row)

=# SELECT ST_Summary(ST_GeogFromText('LINESTRING(0 0 1, 1 1 1)')) As geog_line,
          ST_Summary(ST_GeomFromText('SRID=4326;POLYGON((0 0 1, 1 1 2, 1 2 3, 1 1 1, 0 0 1)) ←
          ') As geom_poly;
;
          geog_line                |                geom_poly
-----+-----
 LineString[ZBGS] with 2 points | Polygon[ZBS] with 1 rings
                               :   ring 0 has 5 points
                               :
(1 row)
```

**Siehe auch**

[PostGIS\\_DropBBox](#), [PostGIS\\_AddBBox](#), [ST\\_Force3DM](#), [ST\\_Force3DZ](#), [ST\\_Force2D](#), [geography](#)  
[ST\\_IsValid](#), [ST\\_IsValidReason](#), [ST\\_IsValidDetail](#)

**7.4.41 ST\_X**

`ST_X` — Gibt die X-Koordinate eines Punktes zurück.

**Synopsis**

```
float ST_X(geometry a_point);
```

**Beschreibung**

Gibt die X-Koordinate eines Punktes, oder NULL wenn diese nicht vorhanden ist, zurück. Die Eingabe muss ein Punkt sein.

**Note**

Um den minimalen und maximalen X-Wert der Geometriekoordinaten zu ermitteln, verwenden Sie die Funktionen [ST\\_XMin](#) und [ST\\_XMax](#).



Diese Methode setzt die SQL/MM-Spezifikation um. SQL-MM 3: 6.1.3



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

**Beispiele**

```
SELECT ST_X(ST_GeomFromEWKT('POINT(1 2 3 4)'));
 st_x
-----
      1
(1 row)

SELECT ST_Y(ST_Centroid(ST_GeomFromEWKT('LINESTRING(1 2 3 4, 1 1 1 1)')));
 st_y
-----
   1.5
(1 row)
```

**Siehe auch**

[ST\\_Centroid](#), [ST\\_GeomFromEWKT](#), [ST\\_M](#), [ST\\_XMax](#), [ST\\_XMin](#), [ST\\_Y](#), [ST\\_Z](#)

**7.4.42 ST\_Y**

`ST_Y` — Gibt die Y-Koordinate eines Punktes zurück.

## Synopsis

```
float ST_Y(geometry a_point);
```

## Beschreibung

Gibt die Y-Koordinate eines Punktes, oder NULL wenn diese nicht vorhanden ist, zurück. Die Eingabe muss ein Punkt sein.



### Note

Um den minimalen und maximalen Y-Wert der Geometriekoordinaten zu ermitteln, verwenden Sie die Funktionen [ST\\_YMin](#) und [ST\\_YMax](#).



Diese Methode implementiert die [OGC Simple Features Implementation Specification for SQL 1.1](#).



Diese Methode setzt die SQL/MM-Spezifikation um. SQL-MM 3: 6.1.4



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

## Beispiele

```
SELECT ST_Y(ST_GeomFromEWKT('POINT(1 2 3 4)'));
 st_y
-----
      2
(1 row)

SELECT ST_Y(ST_Centroid(ST_GeomFromEWKT('LINESTRING(1 2 3 4, 1 1 1 1)')));
 st_y
-----
   1.5
(1 row)
```

## Siehe auch

[ST\\_Centroid](#), [ST\\_GeomFromEWKT](#), [ST\\_M](#), [ST\\_X](#), [ST\\_YMax](#), [ST\\_YMin](#), [ST\\_Z](#)

### 7.4.43 ST\_Z

**ST\_Z** — Gibt die Z-Koordinate eines Punktes zurück.

## Synopsis

```
float ST_Z(geometry a_point);
```

## Beschreibung

Gibt die Z-Koordinate eines Punktes, oder NULL wenn diese nicht vorhanden ist, zurück. Die Eingabe muss ein Punkt sein.



### Note

Um den minimalen und maximalen Z-Wert der Geometriekoordinaten zu ermitteln, verwenden Sie die Funktionen [ST\\_ZMin](#) und [ST\\_ZMax](#).



Diese Methode setzt die SQL/MM-Spezifikation um.



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

## Beispiele

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

## Siehe auch

[ST\\_GeomFromEWKT](#), [ST\\_M](#), [ST\\_X](#), [ST\\_Y](#), [ST\\_ZMax](#), [ST\\_ZMin](#)

## 7.4.44 ST\_Zmflag

ST\_Zmflag — Gibt die Dimension der Koordinaten von ST\_Geometry zurück.

### Synopsis

smallint **ST\_Zmflag**(geometry geomA);

### Beschreibung

Gibt die Dimension der Koordinaten für den Wert von ST\_Geometry zurück.

Die Werte sind: 0 = 2D, 1 = 3D-M, 2 = 3D-Z, 3 = 4D.



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



Diese Methode unterstützt kreisförmige Strings und Kurven.

## Beispiele

```
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
-----
          3
```

```

st_zmflag
-----
                1
SELECT ST_Zmflag(ST_GeomFromEWKT('CIRCULARSTRING(1 2 3, 3 4 3, 5 6 3)'));
st_zmflag
-----
                2
SELECT ST_Zmflag(ST_GeomFromEWKT('POINT(1 2 3 4)'));
st_zmflag
-----
                3

```

**Siehe auch**

[ST\\_CoordDim](#), [ST\\_NDims](#), [ST\\_Dimension](#)

**7.4.45 ST\_HasZ**

ST\_HasZ — Prüft, ob eine Geometrie eine Z-Dimension hat.

**Synopsis**

boolean **ST\_HasZ**(geometry geom);

**Beschreibung**

Prüft, ob die Eingabegeometrie eine Z-Dimension hat und gibt einen booleschen Wert zurück. Wenn die Geometrie eine Z-Dimension hat, wird true zurückgegeben, andernfalls false.

Geometrieobjekte mit einer Z-Dimension stellen typischerweise dreidimensionale (3D) Geometrien dar, während solche ohne Z-Dimension zweidimensionale (2D) Geometrien sind.

Diese Funktion ist nützlich, um festzustellen, ob eine Geometrie Höhen- oder Breiteninformationen enthält.

Verfügbarkeit: 3.5.0



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



Diese Funktion unterstützt M-Koordinaten.

**Beispiele**

```

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

```

```

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

```

**Siehe auch**

[ST\\_Zmflag](#)

[ST\\_HasM](#)

## 7.4.46 ST\_HasM

ST\_HasM — Prüft, ob eine Geometrie eine M-Dimension (Maß) hat.

### Synopsis

```
boolean ST_HasM(geometry geom);
```

### Beschreibung

Prüft, ob die Eingabegeometrie eine M-Dimension (Maß) hat und gibt einen booleschen Wert zurück. Wenn die Geometrie ein M-Maß hat, wird true zurückgegeben, andernfalls false.

Geometrieobjekte mit einer M-Dimension stellen in der Regel Messungen oder zusätzliche Daten dar, die mit räumlichen Merkmalen verbunden sind.

Diese Funktion ist nützlich, um festzustellen, ob eine Geometrie Messinformationen enthält.

Verfügbarkeit: 3.5.0



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



Diese Funktion unterstützt M-Koordinaten.

### Beispiele

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

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

### Siehe auch

[ST\\_Zmflag](#)

[ST\\_HasZ](#)

## 7.5 Geometrische Editoren

### 7.5.1 ST\_AddPoint

ST\_AddPoint — Fügt einem Linienzug einen Punkt hinzu.

### Synopsis

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

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

## Beschreibung

Fügt einen Punkt zu einem LineString vor dem Index *Position* hinzu (unter Verwendung eines 0-basierten Index). Wenn der Parameter *position* weggelassen wird oder -1 ist, wird der Punkt an das Ende des LineString angehängt.

Verfügbarkeit: 1.1.0



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

## Beispiele

Hinzufügen eines Punktes am Ende einer 3D-Linie

```
SELECT ST_AsEWKT(ST_AddPoint('LINESTRING(0 0 1, 1 1 1)', ST_MakePoint(1, 2, 3)));

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

Garantiert, dass alle Zeilen einer Tabelle geschlossen sind, indem der Anfangspunkt jeder Zeile nur bei den nicht geschlossenen Zeilen an das Zeilenende angefügt wird.

```
UPDATE sometable
SET geom = ST_AddPoint(geom, ST_StartPoint(geom))
FROM sometable
WHERE ST_IsClosed(geom) = false;
```

## Siehe auch

[ST\\_RemovePoint](#), [ST\\_SetPoint](#)

## 7.5.2 ST\_CollectionExtract

`ST_CollectionExtract` — Gibt bei einer Geometriesammlung eine Multi-Geometrie zurück, die nur Elemente eines bestimmten Typs enthält.

### Synopsis

```
geometry ST_CollectionExtract(geometry collection);
geometry ST_CollectionExtract(geometry collection, integer type);
```

### Beschreibung

Gibt bei einer Geometriesammlung eine homogene Multi-Geometrie zurück.

Wenn der Typ nicht angegeben ist, wird eine Multi-Geometrie zurückgegeben, die nur Geometrien der höchsten Dimension enthält. Polygone werden also gegenüber Linien bevorzugt, die wiederum gegenüber Punkten bevorzugt werden.

Wenn der Typ angegeben ist, wird eine Multi-Geometrie zurückgegeben, die nur diesen Typ enthält. Gibt es keine Untergeometrien des richtigen Typs, wird eine EMPTY-Geometrie zurückgegeben. Es werden nur Punkte, Linien und Polygone unterstützt. Die Typnummern sind:

- 1 == PUNKT
- 2 == ZEILENUMBRUCH

- 3 == POLYGON

Bei atomaren Geometrieangaben wird die Geometrie unverändert wiedergegeben, wenn der Eingabetyp mit dem angeforderten Typ übereinstimmt. Andernfalls ist das Ergebnis eine LEERE Geometrie des angegebenen Typs. Falls erforderlich, können diese mit [ST\\_Multi](#) in Multi-Geometrien umgewandelt werden.



#### Warning

MultiPolygon-Ergebnisse werden nicht auf ihre Gültigkeit geprüft. Wenn die Polygonkomponenten nebeneinander liegen oder sich überlappen, ist das Ergebnis ungültig. (Dies kann z.B. bei der Anwendung dieser Funktion auf ein [ST\\_Split](#) Ergebnis auftreten.) Diese Situation kann mit [ST\\_IsValid](#) überprüft und mit [ST\\_MakeValid](#) behoben werden.

Verfügbarkeit: 1.5.0



#### Note

Vor 1.5.3 gab diese Funktion atomare Eingaben unverändert zurück, unabhängig vom Typ. In 1.5.3 lieferten nicht übereinstimmende Einzelgeometrien ein NULL-Ergebnis. In 2.0.0 geben nicht übereinstimmende Einzelgeometrien ein LEERES Ergebnis des angeforderten Typs zurück.

## Beispiele

Extrahieren Sie den Typ mit der höchsten Dimension:

```
SELECT ST_AsText(ST_CollectionExtract(
    'GEOMETRYCOLLECTION( POINT(0 0), LINESTRING(1 1, 2 2) )');
    st_astext
    -----
MULTILINESTRING((1 1, 2 2))
```

Punkte extrahieren (Typ 1 == POINT):

```
SELECT ST_AsText(ST_CollectionExtract(
    'GEOMETRYCOLLECTION(GEOMETRYCOLLECTION(POINT(0 0))),
    1 ));
    st_astext
    -----
MULTIPOINT((0 0))
```

Zeilen extrahieren (Typ 2 == LINESTRING):

```
SELECT ST_AsText(ST_CollectionExtract(
    'GEOMETRYCOLLECTION(GEOMETRYCOLLECTION(LINESTRING(0 0, 1 1)),LINESTRING(2 2, 3 3)) ←
    ',
    2 ));
    st_astext
    -----
MULTILINESTRING((0 0, 1 1), (2 2, 3 3))
```

## Siehe auch

[ST\\_CollectionHomogenize](#), [ST\\_Multi](#), [ST\\_IsValid](#), [ST\\_MakeValid](#)

## 7.5.3 ST\_CollectionHomogenize

[ST\\_CollectionHomogenize](#) — Gibt die einfachste Darstellung einer Geometriesammlung zurück.



## Synopsis

```
geometry ST_CollectionHomogenize(geometry collection);
```

## Beschreibung

Gibt bei einer Geometriesammlung die "einfachste" Darstellung des Inhalts zurück.

- Homogene (einheitliche) Sammlungen werden als die entsprechende Multi-Geometrie zurückgegeben.
- Heterogene (gemischte) Sammlungen werden zu einer einzigen GeometryCollection zusammengeführt.
- Sammlungen, die ein einzelnes atomares Element enthalten, werden als dieses Element zurückgegeben.
- Atomare Geometrien werden unverändert zurückgegeben. Bei Bedarf können diese mit **ST\_Multi** in eine Multi-Geometrie umgewandelt werden.



### Warning

Diese Funktion stellt nicht sicher, dass das Ergebnis gültig ist. Insbesondere wird eine Sammlung, die benachbarte oder überlappende Polygone enthält, ein ungültiges MultiPolygon erzeugen. Diese Situation kann mit **ST\_IsValid** überprüft und mit **ST\_MakeValid** behoben werden.

Verfügbarkeit: 2.0.0

## Beispiele

### In eine atomare Geometrie umgewandelte Einzelementsammlung

```
SELECT ST_AsText(ST_CollectionHomogenize('GEOMETRYCOLLECTION(POINT(0 0))'));

 st_astext
-----
POINT(0 0)
```

### Verschachtelte Einzelementsammlung, die in eine atomare Geometrie umgewandelt wurde:

```
SELECT ST_AsText(ST_CollectionHomogenize('GEOMETRYCOLLECTION(MULTIPOINT((0 0)))'));

 st_astext
-----
POINT(0 0)
```

### Sammlung in eine Multi-Geometrie umgewandelt:

```
SELECT ST_AsText(ST_CollectionHomogenize('GEOMETRYCOLLECTION(POINT(0 0),POINT(1 1))'));

 st_astext
-----
MULTIPOINT((0 0),(1 1))
```

### Verschachtelte heterogene Sammlung, die zu einer GeometryCollection abgeflacht wurde:

```
SELECT ST_AsText(ST_CollectionHomogenize('GEOMETRYCOLLECTION(POINT(0 0), GEOMETRYCOLLECTION ←
 ( LINESTRING(1 1, 2 2))')));

 st_astext
-----
GEOMETRYCOLLECTION(POINT(0 0),LINESTRING(1 1,2 2))
```

Sammlung von Polygonen, die in ein (ungültiges) MultiPolygon umgewandelt wurden:

```
SELECT ST_AsText(ST_CollectionHomogenize('GEOMETRYCOLLECTION (POLYGON ((10 50, 50 50, 50
  10, 10 10, 10 50)), POLYGON ((90 50, 90 10, 50 10, 50 50, 90 50)))'));

st_astext
-----
MULTIPOLYGON(((10 50,50 50,50 10,10 10,10 50)),((90 50,90 10,50 10,50 50,90 50)))
```

**Siehe auch**

[ST\\_CollectionExtract](#), [ST\\_Multi](#), [ST\\_IsValid](#), [ST\\_MakeValid](#)

## 7.5.4 ST\_CurveToLine

ST\_CurveToLine — Konvertiert eine Geometrie mit Kurven in eine lineare Geometrie.

### Synopsis

geometry **ST\_CurveToLine**(geometry curveGeom, float tolerance, integer tolerance\_type, integer flags);

### Beschreibung

Konvertiert einen CIRCULAR STRING in einen normalen LINESTRING, ein CURVEPOLYGON in ein POLYGON und ein MULTISURFACE in ein MULTIPOLYGON. Ist nützlich zur Ausgabe an Geräten, die keine Kreisbögen unterstützen.

Wandelt eine gegebene Geometrie in eine lineare Geometrie um. Jede Kurvengeometrie und jedes Kurvensegment wird in linearer Näherung mit der gegebenen Toleranz und Optionen konvertiert ( Standardmäßig 32 Segmenten pro Viertelkreis und keine Optionen).

Der Übergabewert 'tolerance\_type' gibt den Toleranztyp an. Er kann die folgenden Werte annehmen:

- 0 (default): die Toleranz wird über die maximale Anzahl der Segmente pro Viertelkreis angegeben.
- 1: Die Toleranz wird als maximale Abweichung der Linie von der Kurve in der Einheit der Herkunftsdaten angegeben.
- 2: Die Toleranz entspricht dem maximalen Winkel zwischen zwei erzeugten Radien.

Der Parameter 'flags' ist ein Bitfeld mit dem Standardwert 0. Es werden folgende Bits unterstützt:

- 1: Symmetrische (orientierungsunabhängige) Ausgabe.
- 2: Erhält den Winkel, vermeidet die Winkel (Segmentlängen) bei der symmetrischen Ausgabe zu reduzieren. Hat keine Auswirkung, wenn die Symmetrie-Flag nicht aktiviert ist.

Verfügbarkeit: 1.3.0

Erweiterung: ab 2.4.0 kann die Toleranz über die 'maximale Abweichung' und den 'maximalen Winkel' angegeben werden. Die symmetrische Ausgabe wurde hinzugefügt.

Erweiterung: 3.0.0 führte eine minimale Anzahl an Segmenten pro linearisierten Bogen ein, um einem topologischen Kollaps vorzubeugen.



Diese Methode implementiert die [OGC Simple Features Implementation Specification for SQL 1.1](#).



Diese Methode setzt die SQL/MM-Spezifikation um. SQL-MM 3: 7.1.7



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



Diese Methode unterstützt kreisförmige Strings und Kurven.

**Beispiele**

```
SELECT ST_AsText(ST_CurveToLine(ST_GeomFromText('CIRCULARSTRING(220268 150415,220227 150505,220227 150406)')));

--Result --
LINESTRING(220268 150415,220269.95064912 150416.539364228,220271.823415575 150418.17258804,220273.613787707 150419.895736857,
220275.317452352 150421.704659462,220276.930305234 150423.594998003,220278.448460847 150425.562198489,
220279.868261823 150427.60152176,220281.186287736 150429.708054909,220282.399363347 150431.876723113,
220283.50456625 150434.10230186,220284.499233914 150436.379429536,220285.380970099 150438.702620341,220286.147650624 150441.066277505,
220286.797428488 150443.464706771,220287.328738321 150445.892130112,220287.740300149 150448.342699654,
220288.031122486 150450.810511759,220288.200504713 150453.289621251,220288.248038775 150455.77405574,
220288.173610157 150458.257830005,220287.977398166 150460.734960415,220287.659875492 150463.199479347,
220287.221807076 150465.64544956,220286.664248262 150468.066978495,220285.988542259 150470.458232479,220285.196316903 150472.81345077,
220284.289480732 150475.126959442,220283.270218395 150477.39318505,220282.140985384 150479.606668057,
220280.90450212 150481.762075989,220279.5637474 150483.85421628,220278.12195122 150485.87804878,
220276.582586992 150487.828697901,220274.949363179 150489.701464356,220273.226214362 150491.491836488,
220271.417291757 150493.195501133,220269.526953216 150494.808354014,220267.559752731 150496.326509628,
220265.520429459 150497.746310603,220263.41389631 150499.064336517,220261.245228106 150500.277412127,
220259.019649359 150501.38261503,220256.742521683 150502.377282695,220254.419330878 150503.259018879,
220252.055673714 150504.025699404,220249.657244448 150504.675477269,220247.229821107 150505.206787101,
220244.779251566 150505.61834893,220242.311439461 150505.909171266,220239.832329968 150506.078553494,
220237.347895479 150506.126087555,220234.864121215 150506.051658938,220232.386990804 150505.855446946,
220229.922471872 150505.537924272,220227.47650166 150505.099855856,220225.054972724 150504.542297043,
220222.663718741 150503.86659104,220220.308500449 150503.074365683,
220217.994991777 150502.167529512,220215.72876617 150501.148267175,
220213.515283163 150500.019034164,220211.35987523 150498.7825509,
220209.267734939 150497.441796181,220207.243902439 150496,
220205.293253319 150494.460635772,220203.420486864 150492.82741196,220201.630114732 150491.104263143,
220199.926450087 150489.295340538,220198.313597205 150487.405001997,220196.795441592 150485.437801511,
220195.375640616 150483.39847824,220194.057614703 150481.291945091,220192.844539092 150479.123276887,220191.739336189 150476.89769814,
220190.744668525 150474.620570464,220189.86293234 150472.297379659,220189.096251815 150469.933722495,
220188.446473951 150467.535293229,220187.915164118 150465.107869888,220187.50360229 150462.657300346,
220187.212779953 150460.189488241,220187.043397726 150457.710378749,220186.995863664 150455.22594426,
220187.070292282 150452.742169995,220187.266504273 150450.265039585,220187.584026947 150447.800520653,
220188.022095363 150445.35455044,220188.579654177 150442.933021505,220189.25536018 150440.541767521,
```

```

220190.047585536 150438.18654923,220190.954421707 150435.873040558,220191.973684044 ↔
  150433.60681495,
220193.102917055 150431.393331943,220194.339400319 150429.237924011,220195.680155039 ↔
  150427.14578372,220197.12195122 150425.12195122,
220198.661315447 150423.171302099,220200.29453926 150421.298535644,220202.017688077 ↔
  150419.508163512,220203.826610682 150417.804498867,
220205.716949223 150416.191645986,220207.684149708 150414.673490372,220209.72347298 ↔
  150413.253689397,220211.830006129 150411.935663483,
220213.998674333 150410.722587873,220216.22425308 150409.61738497,220218.501380756 ↔
  150408.622717305,220220.824571561 150407.740981121,
220223.188228725 150406.974300596,220225.586657991 150406.324522731,220227 150406)

--3d example
SELECT ST_AsEWKT(ST_CurveToLine(ST_GeomFromEWKT('CIRCULARSTRING(220268 150415 1,220227 ↔
  150505 2,220227 150406 3)')));
Output
-----
LINESTRING(220268 150415 1,220269.95064912 150416.539364228 1.0181172856673,
220271.823415575 150418.17258804 1.03623457133459,220273.613787707 150419.895736857 ↔
  1.05435185700189,...AD INFINITUM ....
  220225.586657991 150406.324522731 1.32611114201132,220227 150406 3)

--use only 2 segments to approximate quarter circle
SELECT ST_AsText(ST_CurveToLine(ST_GeomFromText('CIRCULARSTRING(220268 150415,220227 ↔
  150505,220227 150406)'),2));
st_astext
-----
LINESTRING(220268 150415,220287.740300149 150448.342699654,220278.12195122 ↔
  150485.87804878,
220244.779251566 150505.61834893,220207.243902439 150496,220187.50360229 150462.657300346,
220197.12195122 150425.12195122,220227 150406)

-- Ensure approximated line is no further than 20 units away from
-- original curve, and make the result direction-neutral
SELECT ST_AsText(ST_CurveToLine(
  'CIRCULARSTRING(0 0,100 -100,200 0) '::geometry,
  20, -- Tolerance
  1, -- Above is max distance between curve and line
  1 -- Symmetric flag
));
st_astext
-----
LINESTRING(0 0,50 -86.6025403784438,150 -86.6025403784439,200 -1.1331077795296e-13,200 0)

```

**Siehe auch**[ST\\_LineToCurve](#)**7.5.5 ST\_Scroll**

ST\_Scroll — Startpunkt eines geschlossenen LineStrings ändern.

**Synopsis**geometry **ST\_Scroll**(geometry linestring, geometry point);

**Beschreibung**

Ändert den Start-/Endpunkt eines geschlossenen LineStrings auf den angegebenen Scheitelpunkt .

Verfügbarkeit: 3.2.0



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



Diese Funktion unterstützt M-Koordinaten.

**Beispiele**

Eine geschlossene Linie soll an ihrem 3. Scheitelpunkt beginnen.

```
SELECT ST_AseWKT(ST_Scroll('SRID=4326;LINESTRING(0 0 0 1, 10 0 2 0, 5 5 4 2,0 0 0 1)', ' ←
    POINT(5 5 4 2)'));
```

```
st_asewkt
```

```
-----
```

```
SRID=4326;LINESTRING(5 5 4 2,0 0 0 1,10 0 2 0,5 5 4 2)
```

**Siehe auch**

[ST\\_Normalize](#)

**7.5.6 ST\_FlipCoordinates**

ST\_FlipCoordinates — Gibt eine Version einer Geometrie mit gespiegelter X- und Y-Achse zurück.

**Synopsis**

```
geometry ST_FlipCoordinates(geometry geom);
```

**Beschreibung**

Liefert eine Version der angegebenen Geometrie mit gespiegelter X- und Y-Achse. Nützlich zum Fixieren von Geometrien, die Koordinaten enthalten, die als Breitengrad/Längengrad (Y,X) ausgedrückt sind.

Verfügbarkeit: 2.0.0



Diese Methode unterstützt kreisförmige Strings und Kurven.



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



Diese Funktion unterstützt M-Koordinaten.



Diese Funktion unterstützt polyedrische Flächen.



Diese Funktion unterstützt Dreiecke und dreieckige unregelmäßige Netzoberflächen (TIN).

**Beispiel**

```
SELECT ST_AseWKT(ST_FlipCoordinates(GeomFromEWKT('POINT(1 2)')));
```

```
st_asewkt
```

```
-----
```

```
POINT(2 1)
```

**Siehe auch**[ST\\_SwapOrdinates](#)**7.5.7 ST\_Force2D**

ST\_Force2D — Die Geometrien in einen "2-dimensionalen Modus" zwingen.

**Synopsis**

```
geometry ST_Force2D(geometry geomA);
```

**Beschreibung**

Zwingt die Geometrien in einen "2-dimensionalen Modus", sodass in der Ausgabe nur die X- und Y-Koordinaten dargestellt werden. Nützlich um eine OGC-konforme Ausgabe zu erhalten (da OGC nur 2-D Geometrien spezifiziert).

Erweiterung: Mit 2.0.0 wurde die Unterstützung für polyedrische Oberflächen eingeführt.

Änderung: 2.1.0. Bis zu 2.0.x wurde diese Funktion mit ST\_Force\_2D bezeichnet.



Diese Methode unterstützt kreisförmige Strings und Kurven.



Diese Funktion unterstützt polyedrische Flächen.



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

**Beispiele**

```
SELECT ST_AsEWKT(ST_Force2D(ST_GeomFromEWKT('CIRCULARSTRING(1 1 2, 2 3 2, 4 5 2, 6 7 2, 5 6 2)')));
           st_asewkt
-----
CIRCULARSTRING(1 1,2 3,4 5,6 7,5 6)

SELECT ST_AsEWKT(ST_Force2D('POLYGON((0 0 2,0 5 2,5 0 2,0 0 2),(1 1 2,3 1 2,1 3 2,1 1 2))'));
           st_asewkt
-----
POLYGON((0 0,0 5,5 0,0 0),(1 1,3 1,1 3,1 1))
```

**Siehe auch**[ST\\_Force3D](#)**7.5.8 ST\_Force3D**

ST\_Force3D — Zwingt die Geometrien in einen XYZ Modus. Dies ist ein Alias für ST\_Force3DZ.

**Synopsis**

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

## Beschreibung

Zwingt die Geometrien in den XYZ-Modus. Dies ist ein Alias für `ST_Force3DZ`. Wenn eine Geometrie keine Z-Komponente hat, wird ein *z-Wert* Z-Koordinate angehängt.

Erweiterung: Mit 2.0.0 wurde die Unterstützung für polyedrische Oberflächen eingeführt.

Änderung: 2.1.0. Bis zu 2.0.x wurde diese Funktion mit `ST_Force_3D` bezeichnet.

Geändert: 3.1.0. Unterstützung für die Angabe eines Z-Wertes ungleich Null wurde zugefügt.



Diese Funktion unterstützt polyedrische Flächen.



Diese Methode unterstützt kreisförmige Strings und Kurven.



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

## Beispiele

```
--Nothing happens to an already 3D geometry
SELECT ST_AsEWKT(ST_Force3D(ST_GeomFromEWKT('CIRCULARSTRING(1 1 2, 2 3 2, 4 5 2, 6 7 2, 5 6 2)')));
          st_asewkt
-----
CIRCULARSTRING(1 1 2,2 3 2,4 5 2,6 7 2,5 6 2)

SELECT ST_AsEWKT(ST_Force3D('POLYGON((0 0,0 5,5 0,0 0),(1 1,3 1,1 3,1 1))'));
          st_asewkt
-----
POLYGON((0 0 0,0 5 0,5 0 0,0 0 0),(1 1 0,3 1 0,1 3 0,1 1 0))
```

## Siehe auch

[ST\\_AsEWKT](#), [ST\\_Force2D](#), [ST\\_Force3DM](#), [ST\\_Force3DZ](#)

## 7.5.9 ST\_Force3DZ

`ST_Force3DZ` — Zwingt die Geometrien in einen XYZ Modus.

### Synopsis

geometry **ST\_Force3DZ**(geometry geomA, float Zvalue = 0.0);

### Beschreibung

Zwingt die Geometrien in den XYZ-Modus. Wenn eine Geometrie keine Z-Komponente hat, wird ein *z-Wert* Z-Koordinate angehängt.



Erweiterung: Mit 2.0.0 wurde die Unterstützung für polyedrische Oberflächen eingeführt.

Änderung: 2.1.0. Bis zu 2.0.x wurde diese Funktion mit `ST_Force_3DZ` bezeichnet.

Geändert: 3.1.0. Unterstützung für die Angabe eines Z-Wertes ungleich Null wurde zugefügt.



Diese Funktion unterstützt polyedrische Flächen.

-  Diese Funktion unterstützt 3d und lässt den Z-Index nicht fallen.
-  Diese Methode unterstützt kreisförmige Strings und Kurven.

### Beispiele

```
--Nothing happens to an already 3D geometry
SELECT ST_AsEWKT(ST_Force3DZ(ST_GeomFromEWKT('CIRCULARSTRING(1 1 2, 2 3 2, 4 5 2, 6 7 2, 5 6 2)')));

```

	st_asewkt
CIRCULARSTRING(1 1 2,2 3 2,4 5 2,6 7 2,5 6 2)	

```
SELECT ST_AsEWKT(ST_Force3DZ('POLYGON((0 0,0 5,5 0,0 0),(1 1,3 1,1 3,1 1))'));

```

	st_asewkt
POLYGON((0 0 0,0 5 0,5 0 0,0 0 0),(1 1 0,3 1 0,1 3 0,1 1 0))	

### Siehe auch

[ST\\_AsEWKT](#), [ST\\_Force2D](#), [ST\\_Force3DM](#), [ST\\_Force3D](#)

## 7.5.10 ST\_Force3DM

ST\_Force3DM — Zwingt die Geometrien in einen XYM Modus.

### Synopsis

geometry **ST\_Force3DM**(geometry geomA, float Mvalue = 0.0);

### Beschreibung

Zwingt die Geometrien in den XYM-Modus. Wenn eine Geometrie keine M-Komponente hat, wird ein *M-Wert* M-Koordinate angehängt. Wenn sie eine Z-Komponente hat, wird Z entfernt.

Änderung: 2.1.0. Bis zu 2.0.x wurde diese Funktion mit ST\_Force\_3DM bezeichnet.

Geändert: 3.1.0. Unterstützung für die Angabe eines M-Wertes ungleich Null wurde hinzugefügt.

-  Diese Methode unterstützt kreisförmige Strings und Kurven.

### Beispiele

```
--Nothing happens to an already 3D geometry
SELECT ST_AsEWKT(ST_Force3DM(ST_GeomFromEWKT('CIRCULARSTRING(1 1 2, 2 3 2, 4 5 2, 6 7 2, 5 6 2)')));

```

	st_asewkt
CIRCULARSTRINGM(1 1 0,2 3 0,4 5 0,6 7 0,5 6 0)	



```
SELECT ST_AsEWKT(ST_Force3DM('POLYGON((0 0 1,0 5 1,5 0 1,0 0 1),(1 1 1,3 1 1,1 3 1,1 1 1)) ←
  '));

```

	st_asewkt
-----	
	POLYGONM((0 0 0,0 5 0,5 0 0,0 0 0),(1 1 0,3 1 0,1 3 0,1 1 0))

**Siehe auch**

[ST\\_AsEWKT](#), [ST\\_Force2D](#), [ST\\_Force3DM](#), [ST\\_Force3D](#), [ST\\_GeomFromEWKT](#)

**7.5.11 ST\_Force4D**

`ST_Force4D` — Zwingt die Geometrien in einen XYZM Modus.

**Synopsis**

geometry **ST\_Force4D**(geometry geomA, float Zvalue = 0.0, float Mvalue = 0.0);

**Beschreibung**

Zwingt die Geometrien in den XYZM-Modus. *Zvalue* und *Mvalue* wird für fehlende Z- bzw. M-Dimensionen angehängt.

Änderung: 2.1.0. Bis zu 2.0.x wurde diese Funktion mit `ST_Force_4D` bezeichnet.

Geändert: 3.1.0. Unterstützung für die Angabe von Z- und M-Werten ungleich Null wurde hinzugefügt.



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



Diese Methode unterstützt kreisförmige Strings und Kurven.

**Beispiele**

```
--Nothing happens to an already 3D geometry
SELECT ST_AsEWKT(ST_Force4D(ST_GeomFromEWKT('CIRCULARSTRING(1 1 2, 2 3 2, 4 5 2, 6 7 2, 5 6 ←
  2)'))));

```

	st_asewkt
-----	
	CIRCULARSTRING(1 1 2 0,2 3 2 0,4 5 2 0,6 7 2 0,5 6 2 0)

```
SELECT ST_AsEWKT(ST_Force4D('MULTILINESTRINGM((0 0 1,0 5 2,5 0 3,0 0 4),(1 1 1,3 1 1,1 3 ←
  1,1 1 1))'));

```

	st_asewkt
-----	
	MULTILINESTRING((0 0 0 1,0 5 0 2,5 0 0 3,0 0 0 4),(1 1 0 1,3 1 0 1,1 3 0 1,1 1 0 1))

**Siehe auch**

[ST\\_AsEWKT](#), [ST\\_Force2D](#), [ST\\_Force3DM](#), [ST\\_Force3D](#)

## 7.5.12 ST\_ForceCollection

ST\_ForceCollection — Wandelt eine Geometrie in eine GEOMETRYCOLLECTION um.

### Synopsis

geometry **ST\_ForceCollection**(geometry geomA);

### Beschreibung

Wandelt eine Geometrie in eine GEOMETRYCOLLECTION um. Nützlich um eine WKB-Darstellung zu vereinfachen.

Erweiterung: Mit 2.0.0 wurde die Unterstützung für polyedrische Oberflächen eingeführt.

Verfügbarkeit: 1.2.2, Vor 1.3.4 ist diese Funktion bei CURVES abgestürzt. Dies wurde mit 1.3.4+ behoben

Änderung: 2.1.0. Bis zu 2.0.x wurde diese Funktion mit ST\_Force\_Collection bezeichnet.



Diese Funktion unterstützt polyedrische Flächen.



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



Diese Methode unterstützt kreisförmige Strings und Kurven.

### Beispiele

```
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 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 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)
```

```
-- POLYHEDRAL example --
SELECT ST_AsEWKT(ST_ForceCollection('POLYHEDRALSURFACE(((0 0 0,0 0 1,0 1 1,0 1 0,0 0 0)),
((0 0 0,0 1 0,1 1 0,1 0 0,0 0 0)),
((0 0 0,1 0 0,1 0 1,0 0 1,0 0 0)),
((1 1 0,1 1 1,1 0 1,1 0 0,1 1 0)),
((0 1 0,0 1 1,1 1 1,1 1 0,0 1 0)),
((0 0 1,1 0 1,1 1 1,0 1 1,0 0 1)))'));

```

st\_asewkt

---

```
GEOMETRYCOLLECTION (
  POLYGON((0 0 0,0 0 1,0 1 1,0 1 0,0 0 0)),
  POLYGON((0 0 0,0 1 0,1 1 0,1 0 0,0 0 0)),
  POLYGON((0 0 0,1 0 0,1 0 1,0 0 1,0 0 0)),
  POLYGON((1 1 0,1 1 1,1 0 1,1 0 0,1 1 0)),
  POLYGON((0 1 0,0 1 1,1 1 1,1 1 0,0 1 0)),
  POLYGON((0 0 1,1 0 1,1 1 1,0 1 1,0 0 1))
)
```

**Siehe auch**

[ST\\_AsEWKT](#), [ST\\_Force2D](#), [ST\\_Force3DM](#), [ST\\_Force3D](#), [ST\\_GeomFromEWKT](#)

**7.5.13 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



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



Diese Methode unterstützt kreisförmige Strings und Kurven.

**Beispiele**

```
SELECT ST_AsText (
  ST_ForceCurve (
    'POLYGON((0 0 2, 5 0 2, 0 5 2, 0 0 2),(1 1 2, 1 3 2, 3 1 2, 1 1 2))'::geometry
  )
);
----- st_astext -----
CURVEPOLYGON Z ((0 0 2,5 0 2,0 5 2,0 0 2),(1 1 2,1 3 2,3 1 2,1 1 2))
(1 row)
```

**Siehe auch**

[ST\\_LineToCurve](#)

**7.5.14 ST\_ForcePolygonCCW**

`ST_ForcePolygonCCW` — Richtet alle äußeren Ringe gegen den Uhrzeigersinn und alle inneren Ringe mit dem Uhrzeigersinn aus.

**Synopsis**

```
geometry ST_ForcePolygonCCW ( geometry geom );
```

## Beschreibung

Zwingt (Multi)Polygone, den äusseren Ring gegen den Uhrzeigersinn und die inneren Ringe im Uhrzeigersinn zu orientieren. Andere Geometrien werden unverändert zurückgegeben.

Verfügbarkeit: 2.4.0



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



Diese Funktion unterstützt M-Koordinaten.

## Siehe auch

[ST\\_ForcePolygonCW](#) , [ST\\_IsPolygonCCW](#) , [ST\\_IsPolygonCW](#)

### 7.5.15 ST\_ForcePolygonCW

`ST_ForcePolygonCW` — Richtet alle äußeren Ringe im Uhrzeigersinn und alle inneren Ringe gegen den Uhrzeigersinn aus.

## Synopsis

```
geometry ST_ForcePolygonCW ( geometry geom );
```

## Beschreibung

Zwingt (Multi)Polygone, den äusseren Ring im Uhrzeigersinn und die inneren Ringe gegen den Uhrzeigersinn zu orientieren. Andere Geometrien werden unverändert zurückgegeben.

Verfügbarkeit: 2.4.0



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



Diese Funktion unterstützt M-Koordinaten.

## Siehe auch

[ST\\_ForcePolygonCCW](#) , [ST\\_IsPolygonCCW](#) , [ST\\_IsPolygonCW](#)

### 7.5.16 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**

- ✔ Diese Funktion unterstützt polyedrische Flächen.
- ✔ Diese Funktion unterstützt Dreiecke und dreieckige unregelmäßige Netzoberflächen (TIN).
- ✔ Diese Methode unterstützt kreisförmige Strings und Kurven.
- ✔ Diese Funktion unterstützt 3d und lässt den Z-Index nicht fallen.

**7.5.17 ST\_ForceRHR**

ST\_ForceRHR — Orientiert die Knoten in einem Polygon so, dass sie der Drei-Finger-Regel folgen.

**Synopsis**

geometry **ST\_ForceRHR**(geometry g);

**Beschreibung**

Orientiert die Knoten in einem Polygon so, dass sie der Drei-Finger-Regel folgen. Dadurch kommt die durch das Polygon begrenzte Fläche auf der rechten Seite der Begrenzung zu liegen. Insbesondere sind der äussere Ring im Uhrzeigersinn und die inneren Ringe gegen den Uhrzeigersinn orientiert. Diese Funktion ist ein Synonym für [ST\\_ForcePolygonCW](#)

**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.

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

**Beispiele**

```
SELECT ST_AseWKT (
  ST_ForceRHR (
    'POLYGON((0 0 2, 5 0 2, 0 5 2, 0 0 2), (1 1 2, 1 3 2, 3 1 2, 1 1 2))'
  )
);
```

	st_asewkt
POLYGON((0 0 2,0 5 2,5 0 2,0 0 2), (1 1 2,3 1 2,1 3 2,1 1 2))	
(1 row)	

**Siehe auch**

[ST\\_ForcePolygonCCW](#) , [ST\\_ForcePolygonCW](#) , [ST\\_IsPolygonCCW](#) , [ST\\_IsPolygonCW](#) , [ST\\_BuildArea](#), [ST\\_Polygonize](#), [ST\\_Reverse](#)

### 7.5.18 ST\_LineExtend

ST\_LineExtend — Gibt eine Linie zurück, die um die angegebenen Abstände vorwärts und rückwärts verlängert wurde.

#### Synopsis

geometry **ST\_LineExtend**(geometry line, float distance\_forward, float distance\_backward=0.0);

#### Beschreibung

Gibt eine vorwärts und rückwärts verlängerte Linie zurück, indem neue Start- (und End-) Punkte in den angegebenen Abständen hinzugefügt werden. Bei einem Abstand von Null wird kein Punkt hinzugefügt. Nur nicht-negative Abstände sind erlaubt. Die Richtung des/der hinzugefügten Punkte(s) wird durch die ersten (und letzten) zwei unterschiedlichen Punkte der Linie bestimmt. Doppelte Punkte werden ignoriert.

Verfügbarkeit: 3.4.0

#### Beispiel: Erweitert eine Linie um 5 Einheiten vorwärts und 6 Einheiten rückwärts

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

#### Siehe auch

[ST\\_LineSubstring](#), [ST\\_LocateAlong](#), [ST\\_Project](#)

### 7.5.19 ST\_LineToCurve

ST\_LineToCurve — Konvertiert eine lineare Geometrie in eine gekrümmte Geometrie.

#### Synopsis

geometry **ST\_LineToCurve**(geometry geomANoncircular);

#### Beschreibung

Wandelt einen einfachen LineString/Polygon in Kreisbögen/CIRCULARSTRINGs und Kurvenpolygone um. Beachten Sie, dass wesentlich weniger Punkte zur Beschreibung des Kurvenäquivalents benötigt werden.



#### Note

Wenn der gegebene LINESTRING/POLYGON nicht genug gekrümmt ist um eine deutliche Kurve zu repräsentieren, wird die Geometrie von der Funktion unverändert zurückgegeben.

Verfügbarkeit: 1.3.0



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



Diese Methode unterstützt kreisförmige Strings und Kurven.

**Beispiele**

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

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,-0.14805029   0.228361402466137,   -0.666710699058802 ↵ 0.505591163092361,-1.12132034   0.878679656440353,   -1.49440883690763 ↵ 1.33328930094119,-1.77163859   1.85194970290472   --ETC-- ↵ ,3.94235584120969 ↵ 3.58527096604839,4 ↵ 3))	

```
--3D example
SELECT ST_AsText(ST_LineToCurve(geom)) As curved, ST_AsText(geom) AS not_curved
FROM (SELECT ST_Translate(ST_Force3D(ST_Boundary(ST_Buffer(ST_Point(1,3), 2,2))),0,0,3) AS ↵
    geom) AS foo;
```

curved	not_curved
CIRCULARSTRING Z (3 3 3,-1 2.999999999999999 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,   0.999999999999999 5 ↵ 3,2.41421356237309 4.4142135623731 ↵ 3,3 3 3)	

(1 row)

**Siehe auch**

[ST\\_CurveToLine](#)

## 7.5.20 ST\_Multi

ST\_Multi — Gibt die Geometrie als MULTI\* Geometrie zurück.

### Synopsis

```
geometry ST_Multi(geometry geom);
```

### Beschreibung

Gibt die Geometrie als MULTI\*-Geometriesammlung zurück. Wenn die Geometrie bereits eine Sammlung ist, wird sie unverändert zurückgegeben.

### Beispiele

```
SELECT ST_AsText(ST_Multi('POLYGON ((10 30, 30 30, 30 10, 10 10, 10 30))'));
           st_astext
-----
MULTIPOLYGON(((10 30,30 30,30 10,10 10,10 30)))
```

### Siehe auch

[ST\\_AsText](#)

## 7.5.21 ST\_Normalize

ST\_Normalize — Gibt die Geometrie in Normalform zurück.

### Synopsis

```
geometry ST_Normalize(geometry geom);
```

### Beschreibung

Gibt die Geometrie in Normalform aus. Möglicherweise werden die Knoten der Polygonringe, die Ringe eines Polygons oder die Elemente eines Komplexes von Mehrfachgeometrien neu gereiht.

Hauptsächlich für Testzwecke sinnvoll (zum Vergleich von erwarteten und erhaltenen Ergebnissen).

Verfügbarkeit: 2.3.0

### Beispiele

```
SELECT ST_AsText(ST_Normalize(ST_GeomFromText (
  'GEOMETRYCOLLECTION (
    POINT(2 3),
    MULTILINESTRING((0 0, 1 1),(2 2, 3 3)),
    POLYGON (
      (0 10,0 0,10 0,10 10,0 10),
      (4 2,2 2,2 4,4 4,4 2),
      (6 8,8 8,8 6,6 6,6 8)
    )
  )'
))
```



```

))) ;
-----
GEOMETRYCOLLECTION(POLYGON((0 0,0 10,10 10,10 0,0 0),(6 6,8 6,8 8,6 8,6 6),(2 2,4 2,4 4,2 4,2 2)),MULTILINESTRING((2 2,3 3),(0 0,1 1)),POINT(2 3))
(1 row)

```

## Siehe auch

[ST\\_Equals](#),

## 7.5.22 ST\_Project

**ST\_Project** — Gibt einen Punkt zurück, der von einem Startpunkt um eine bestimmte Entfernung und Peilung (Azimut) projiziert wird.

### Synopsis

```

geometry ST_Project(geometry g1, float distance, float azimuth);
geometry ST_Project(geometry g1, geometry g2, float distance);
geography ST_Project(geography g1, float distance, float azimuth);
geography ST_Project(geography g1, geography g2, float distance);

```

### Beschreibung

Gibt einen Punkt zurück, der von einem Punkt entlang einer Geodäte projiziert wird, wobei eine bestimmte Entfernung und ein bestimmter Azimut (Peilung) verwendet werden. Dies ist bekannt als das direkte geodätische Problem.

Bei der Zweipunktversion wird der Weg vom ersten zum zweiten Punkt verwendet, um implizit den Azimut zu definieren, und die Entfernung wird wie zuvor verwendet.

Die Entfernung wird in Metern angegeben. Negative Werte werden unterstützt.

Der Azimut (auch als Kurs oder Peilung bezeichnet) wird im Bogenmaß angegeben. Er wird im Uhrzeigersinn vom geografischen Norden aus gemessen.

- Norden ist Azimut Null (0 Grad)
- Osten ist Azimut  $\pi/2$  (90 Grad)
- Süden ist Azimut  $\pi$  (180 Grad)
- West ist Azimut  $3\pi/2$  (270 Grad)

Negative Azimutwerte und Werte größer als  $2\pi$  (360 Grad) werden unterstützt.

Verfügbarkeit: 2.0.0

Verbessert: 2.4.0 Erlaubt negative Entfernungen und nicht-normierte Azimute.

Verbessert: 3.4.0 Erlaubt Geometrieargumente und Zweipunktform ohne Azimut.

### Beispiel: Projizierter Punkt auf 100.000 Meter und Peilung 45 Grad

```

SELECT ST_AsText(ST_Project('POINT(0 0)::geography, 100000, radians(45.0)));
-----
POINT(0.635231029125537 0.639472334729198)

```

**Siehe auch**

[ST\\_Azimuth](#), [ST\\_Distance](#), [PostgreSQL-Funktion radians\(\)](#)

**7.5.23 ST\_QuantizeCoordinates**

`ST_QuantizeCoordinates` — Setzt die niedrigwertigsten Bits der Koordinaten auf Null

**Synopsis**

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

**Beschreibung**

`ST_QuantizeCoordinates` ermittelt die Anzahl der Bits (N), die erforderlich sind, um einen Koordinatenwert mit einer bestimmten Anzahl von Nachkommastellen darzustellen, und setzt dann alle Bits bis auf die N höchstwertigen auf Null. Der resultierende Koordinatenwert wird immer noch auf den ursprünglichen Wert gerundet, hat aber eine verbesserte Kompressibilität. Dies kann zu einer erheblichen Verringerung der Festplattennutzung führen, vorausgesetzt, die Geometriespalte verwendet einen **komprimierbaren Speichertyp**. Die Funktion erlaubt die Angabe einer unterschiedlichen Anzahl von Nachkommastellen in jeder Dimension; bei nicht spezifizierten Dimensionen wird die Genauigkeit der x Dimension angenommen. Negative Ziffern werden so interpretiert, dass sie sich auf Ziffern links vom Dezimalpunkt beziehen (d.h. `prec_x=-2` erhält Koordinatenwerte auf die nächsten 100).

Die von `ST_QuantizeCoordinates` erzeugten Koordinaten sind unabhängig von der Geometrie, die diese Koordinaten und die relative Position dieser Koordinaten in der Geometrie enthält. Daher sind vorhandene topologische Beziehungen zwischen Geometrien durch die Verwendung dieser Funktion nicht betroffen. Die Funktion erzeugt möglicherweise ungültige Geometrie, wenn sie mit einer Anzahl von Stellen aufgerufen wird, die Koordinaten innerhalb der Geometrie zusammenfallen lassen.

Verfügbarkeit: 2.5.0

**Technischer Hintergrund**

PostGIS speichert alle Koordinatenwerte als Gleitkommazahlen mit doppelter Genauigkeit, die 15 signifikante Stellen zuverlässig darstellen können. PostGIS kann jedoch verwendet werden, um Daten zu verwalten, die weniger als 15 signifikante Ziffern enthalten. Ein Beispiel sind TIGER-Daten, die als geografische Koordinaten mit sechs Nachkommastellen zur Verfügung gestellt werden (so dass nur neun signifikante Ziffern des Längengrads und acht signifikante Breitengrade erforderlich sind).

Wenn 15 signifikante Ziffern verfügbar sind, gibt es viele mögliche Darstellungen einer Zahl mit 9 signifikanten Ziffern. Eine Gleitkommazahl mit doppelter Genauigkeit verwendet 52 explizite Bits, um den Mantisse der Koordinate darzustellen. Nur 30 Bits werden benötigt, um eine Mantisse mit 9 signifikanten Ziffern darzustellen, wobei 22 unbedeutende Bits übrig bleiben; Wir können ihren Wert auf alles setzen, was wir wollen, und erhalten trotzdem eine zum Eingabewert passende Zahl. Beispielsweise kann der Wert 100.123456 durch die nächstliegenden Zahlen 100.123456000000, 100.123456000001 und 100.123456432199 dargestellt werden. Alle sind gleichermaßen gültig, da `ST_AsText (geom, 6)` bei allen dieser Eingaben das gleiche Ergebnis liefert. Da wir diese Bits auf einen beliebigen Wert setzen können, setzt `ST_QuantizeCoordinates` die 22 nicht signifikanten Bits auf Null. Für eine lange Koordinatensequenz wird dadurch ein Muster aus Blöcken von aufeinanderfolgenden Nullen erzeugt, das von PostgreSQL effizienter komprimiert wird.

**Note**

Von `ST_QuantizeCoordinates` ist möglicherweise nur die Größe der Geometrie auf der Festplatte betroffen. **ST\_MemSize**, das die speicherinterne Verwendung der Geometrie meldet, gibt unabhängig vom von einer Geometrie belegten Speicherplatz den gleichen Wert zurück.

**Beispiele**

```
SELECT ST_AsText(ST_QuantizeCoordinates('POINT (100.123456 0)::geometry, 4));
st_astext
-----
POINT(100.123455047607 0)
```

```
WITH test AS (SELECT 'POINT (123.456789123456 123.456789123456)::geometry AS geom)
SELECT
  digits,
  encode(ST_QuantizeCoordinates(geom, digits), 'hex'),
  ST_AsText(ST_QuantizeCoordinates(geom, digits))
FROM test, generate_series(15, -15, -1) AS digits;
```

digits	encode	st_astext
15	01010000005f9a72083cdd5e405f9a72083cdd5e40	POINT(123.456789123456 123.456789123456) ←
14	01010000005f9a72083cdd5e405f9a72083cdd5e40	POINT(123.456789123456 123.456789123456) ←
13	01010000005f9a72083cdd5e405f9a72083cdd5e40	POINT(123.456789123456 123.456789123456) ←
12	01010000005c9a72083cdd5e405c9a72083cdd5e40	POINT(123.456789123456 123.456789123456) ←
11	0101000000409a72083cdd5e40409a72083cdd5e40	POINT(123.456789123456 123.456789123456) ←
10	0101000000009a72083cdd5e40009a72083cdd5e40	POINT(123.456789123455 123.456789123455) ←
9	0101000000009072083cdd5e40009072083cdd5e40	POINT(123.456789123418 123.456789123418) ←
8	0101000000008072083cdd5e40008072083cdd5e40	POINT(123.45678912336 123.45678912336) ←
7	0101000000000070083cdd5e40000070083cdd5e40	POINT(123.456789121032 123.456789121032) ←
6	0101000000000040083cdd5e40000040083cdd5e40	POINT(123.456789076328 123.456789076328) ←
5	0101000000000000083cdd5e400000000083cdd5e40	POINT(123.456789016724 123.456789016724) ←
4	01010000000000000003cdd5e400000000003cdd5e40	POINT(123.456787109375 123.456787109375) ←
3	0101000000000000000003cdd5e400000000003cdd5e40	POINT(123.456787109375 123.456787109375) ←
2	01010000000000000000038dd5e4000000000038dd5e40	POINT(123.45654296875 123.45654296875) ←
1	0101000000000000000000dd5e400000000000dd5e40	POINT(123.453125 123.453125) ←
0	0101000000000000000000dc5e400000000000dc5e40	POINT(123.4375 123.4375) ←
-1	0101000000000000000000c05e400000000000c05e40	POINT(123 123) ←
-2	010100000000000000000005e400000000000005e40	POINT(120 120) ←
-3	0101000000000000000000058400000000000005840	POINT(96 96) ←
-4	0101000000000000000000058400000000000005840	POINT(96 96) ←
-5	0101000000000000000000058400000000000005840	POINT(96 96) ←
-6	0101000000000000000000058400000000000005840	POINT(96 96) ←
-7	0101000000000000000000058400000000000005840	POINT(96 96) ←
-8	0101000000000000000000058400000000000005840	POINT(96 96) ←
-9	0101000000000000000000058400000000000005840	POINT(96 96) ←
-10	0101000000000000000000058400000000000005840	POINT(96 96) ←
-11	0101000000000000000000058400000000000005840	POINT(96 96) ←
-12	0101000000000000000000058400000000000005840	POINT(96 96) ←
-13	0101000000000000000000058400000000000005840	POINT(96 96) ←
-14	0101000000000000000000058400000000000005840	POINT(96 96) ←

```
-15 | 010100000000000000000000058400000000000005840 | POINT(96 96)
```

### Siehe auch

[ST\\_SnapToGrid](#)

## 7.5.24 ST\_RemovePoint

`ST_RemovePoint` — Einen Punkt aus einem Linienzug entfernen.

### Synopsis

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

### Beschreibung

Entfernt einen Punkt aus einem `LineString` unter Angabe seines Index (0-basiert). Nützlich, um eine geschlossene Linie (Ring) in einen offenen `LineString` zu verwandeln.

Verbessert: 3.2.0

Verfügbarkeit: 1.1.0



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

### Beispiele

Garantiert, dass keine Linien geschlossen sind, indem der Endpunkt von geschlossenen Linien (Ring) entfernt wird. Setzt voraus, dass `geom` vom Typ `LINESTRING` ist

```
UPDATE sometable
   SET geom = ST_RemovePoint(geom, ST_NPoints(geom) - 1)
 FROM sometable
 WHERE ST_IsClosed(geom);
```

### Siehe auch

[ST\\_AddPoint](#), [ST\\_NPoints](#), [ST\\_NumPoints](#)

## 7.5.25 ST\_RemoveRepeatedPoints

`ST_RemoveRepeatedPoints` — Gibt eine Version einer Geometrie zurück, bei der doppelte Punkte entfernt wurden.

### Synopsis

```
geometry ST_RemoveRepeatedPoints(geometry geom, float8 tolerance);
```

## Beschreibung

Gibt eine Version der angegebenen Geometrie zurück, bei der doppelte aufeinanderfolgende Punkte entfernt wurden. Die Funktion verarbeitet nur (Multi)LineStrings, (Multi)Polygone und MultiPoints, kann aber mit jeder Art von Geometrie aufgerufen werden. Elemente von GeometryCollections werden einzeln verarbeitet. Die Endpunkte von LineStrings werden beibehalten.

Wenn der Parameter *tolerance* angegeben wird, werden Scheitelpunkte, die innerhalb des Toleranzabstands zueinander liegen, als Duplikate betrachtet.

Verbessert: 3.2.0

Verfügbarkeit: 2.2.0



Diese Funktion unterstützt polyedrische Flächen.



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

## Beispiele

```
SELECT ST_AsText( ST_RemoveRepeatedPoints( 'MULTIPOINT ((1 1), (2 2), (3 3), (2 2))' ));
-----
MULTIPOINT(1 1,2 2,3 3)
```

```
SELECT ST_AsText( ST_RemoveRepeatedPoints( 'LINESTRING (0 0, 0 0, 1 1, 0 0, 1 1, 2 2)' ));
-----
LINESTRING(0 0,1 1,0 0,1 1,2 2)
```

**Beispiel:** Sammlungselemente werden einzeln bearbeitet.

```
SELECT ST_AsText( ST_RemoveRepeatedPoints( 'GEOMETRYCOLLECTION (LINESTRING (1 1, 2 2, 2 2, ←
3 3), POINT (4 4), POINT (4 4), POINT (5 5))' ));
-----
GEOMETRYCOLLECTION(LINESTRING(1 1,2 2,3 3),POINT(4 4),POINT(4 4),POINT(5 5))
```

**Beispiel:** Wiederholte Punktentfernung mit einer Abstandstoleranz.

```
SELECT ST_AsText( ST_RemoveRepeatedPoints( 'LINESTRING (0 0, 0 0, 1 1, 5 5, 1 1, 2 2)', 2) ←
;
-----
LINESTRING(0 0,5 5,2 2)
```

## Siehe auch

[ST\\_Simplify](#)

### 7.5.26 ST\_RemoveIrrelevantPointsForView

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

#### Synopsis

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

## Beschreibung

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

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

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

In contrast to `ST_ClipByBox2D()` this function

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

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

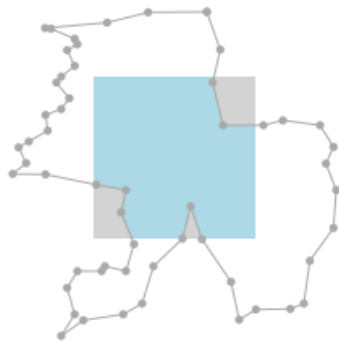


### Warning

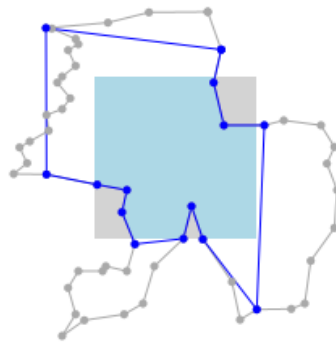
For polygons, this function does currently not ensure that the result is valid. This situation can be checked with `ST_IsValid` and repaired with `ST_MakeValid`.

---

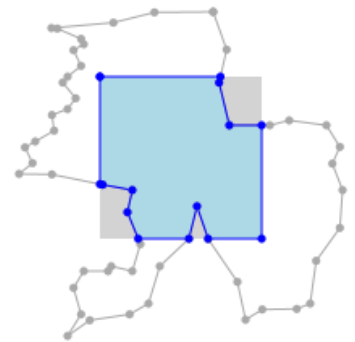
original  
55 points



`ST_RemoveIrrelevantPointsForView(geom, bbox)`  
15 points

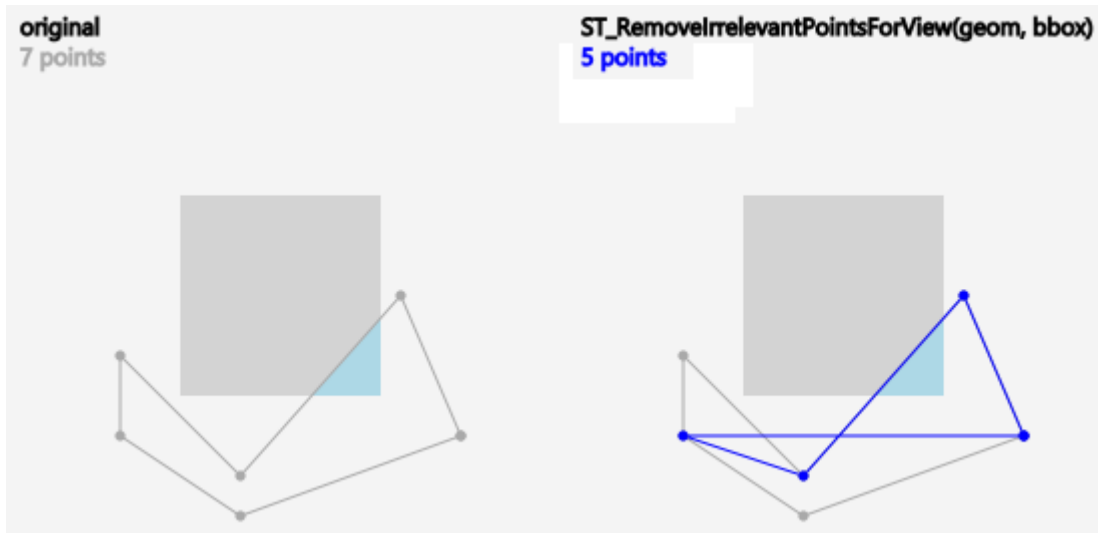


`ST_ClipByBox2D(geom, bbox)`  
15 points



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

---



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

Verfügbarkeit: 3.5.0

**Beispiele**

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

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

```
SELECT ST_AsText (
    ST_RemoveIrrelevantPointsForView(
        ST_GeomFromText ('MULTILINESTRING((0 0, 10 0,20 0,30 0), (0 15, 5 15, 10 15, 15 15, 20 15, 25 15, 30 15, 40 15), (13 13,15 15,17 17))'),
        ST_MakeEnvelope(12,12,18,18), true));

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

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

st_astext
-----
LINESTRING EMPTY
```

```
SELECT ST_AsText (
    ST_RemoveIrrelevantPointsForView(
```

```
ST_GeomFromText('POLYGON((0 30, 15 30, 30 30, 30 0, 0 0, 0 30))'),
ST_MakeEnvelope(12,12,18,18), true));
```

```
st_astext
-----
POLYGON((15 30,30 0,0 0,15 30))
```

```
SELECT ST_AsText (
    ST_RemoveIrrelevantPointsForView(
    ST_GeomFromText('POLYGON((0 30, 15 30, 30 30, 30 0, 0 0, 0 30))'),
    ST_MakeEnvelope(12,12,18,18)));
```

```
st_astext
-----
POLYGON((0 30,30 30,30 0,0 0,0 30))
```

### Siehe auch

[ST\\_ClipByBox2D](#), [ST\\_Intersection](#)

## 7.5.27 ST\_RemoveSmallParts

`ST_RemoveSmallParts` — Removes small parts (polygon rings or linestrings) of a geometry.

### Synopsis

geometry **ST\_RemoveSmallParts**(geometry geom, double precision minSizeX, double precision minSizeY);

### Beschreibung

Returns a **geometry** without small parts (exterior or interior polygon rings, or linestrings).

This function can be used as preprocessing step for creating simplified maps, e. g. to remove small islands or holes.

It evaluates only geometries of type (MULTI)POLYGON and (MULTI)LINESTRING. Other geometries remain unchanged.

If *minSizeX* is greater than 0, parts are sorted out if their width is smaller than *minSizeX*.

If *minSizeY* is greater than 0, parts are sorted out if their height is smaller than *minSizeY*.

Both *minSizeX* and *minSizeY* are measured in coordinate system units of the geometry.

For polygon types, evaluation is done separately for each ring which can lead to one of the following results:

- the original geometry,
- a POLYGON with all rings with less vertices,
- a POLYGON with a reduced number of interior rings (having possibly less vertices),
- a POLYGON EMPTY, or
- a MULTIPOLYGON with a reduced number of polygons (having possibly less interior rings or vertices), or
- a MULTIPOLYGON EMPTY.

For linestring types, evaluation is done for each linestring which can lead to one of the following results:

- the original geometry,



- a LINESRING with a reduced number of vertices,
- a LINESRING EMPTY,
- a MULTILINESTRING with a reduced number of linestrings (having possibly less vertices), or
- a MULTILINESTRING EMPTY.



Example: `ST_RemoveSmallParts()` applied to a multi-polygon. Blue parts remain.

Verfügbarkeit: 3.5.0

## Beispiele

```
SELECT ST_AsText (
    ST_RemoveSmallParts (
        ST_GeomFromText ('MULTIPOLYGON (
            ((60 160, 120 160, 120 220, 60 220, 60 160), (70 170, 70 210, 110 210, 110 170, 70 170)),
            ((85 75, 155 75, 155 145, 85 145, 85 75)),
            ((50 110, 70 110, 70 130, 50 130, 50 110)))'),
            50, 50));

st_astext
-----
MULTIPOLYGON(((60 160,120 160,120 220,60 220,60 160)),((85 75,155 75,155 145,85 145,85 75)))
```

```
SELECT ST_AsText (
    ST_RemoveSmallParts (
        ST_GeomFromText ('LINESRING(10 10, 20 20)'),
        50, 50));

st_astext
-----
LINESRING EMPTY
```

## 7.5.28 ST\_Reverse

`ST_Reverse` — Gibt die Geometrie in umgekehrter Knotenreihenfolge zurück.

## Synopsis

```
geometry ST_Reverse(geometry g1);
```

## Beschreibung

Kann mit jedem geometrischen Datentyp verwendet werden; kehrt die Reihenfolge der Knoten um

Erweiterung: mit 2.4.0 wurde die Unterstützung für Kurven eingeführt.



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



Diese Funktion unterstützt polyedrische Flächen.

## Beispiele

```
SELECT ST_AsText(geom) as line, ST_AsText(ST_Reverse(geom)) As reverseline
FROM
(SELECT ST_MakeLine(ST_Point(1,2),
                   ST_Point(1,10)) As geom) as foo;
--result
           line           |           reverseline
-----+-----
LINESTRING(1 2,1 10) | LINESTRING(1 10,1 2)
```

## 7.5.29 ST\_Segmentize

**ST\_Segmentize** — Gibt eine geänderte Geometrie/Geografie zurück, bei der kein Segment länger als eine bestimmte Entfernung ist.

### Synopsis

```
geometry ST_Segmentize(geometry geom, float max_segment_length);
geography ST_Segmentize(geography geog, float max_segment_length);
```

### Beschreibung

Gibt eine geänderte Geometrie/Geografie zurück, bei der kein Segment länger ist als `max_segment_length`. Die Länge wird in 2D berechnet. Segmente werden immer in Untersegmente gleicher Länge aufgeteilt.

- In der Geometrie wird die maximale Länge in den Einheiten des räumlichen Bezugssystems angegeben.
- In der Geografie wird die maximale Länge in Metern angegeben. Die Entfernungen werden auf der Kugel berechnet. Hinzugefügte Scheitelpunkte werden entlang der sphärischen Großkreisbögen erstellt, die durch die Endpunkte der Segmente definiert sind.



#### Note

Dadurch werden nur lange Segmente verkürzt. Segmente, die kürzer als die maximale Länge sind, werden nicht verlängert.

**Warning**

Bei Eingaben, die lange Segmente enthalten, kann die Angabe einer relativ kurzen `max_segment_length` dazu führen, dass eine sehr große Anzahl von Scheitelpunkten hinzugefügt wird. Dies kann unbeabsichtigt geschehen, wenn das Argument versehentlich als Anzahl der Segmente und nicht als maximale Länge angegeben wird.

Verfügbarkeit: 1.2.2

Verbessert: 3.0.0 Segmentize-Geometrie erzeugt jetzt Teilsegmente gleicher Länge

Verbessert: 2.3.0 Die Segmentierung der Geografie erzeugt nun Teilsegmente gleicher Länge

Erweiterung: mit 2.1.0 wurde die Unterstützung des geographischen Datentyps eingeführt.

Geändert: 2.1.0 Infolge der Einführung der Geographie-Unterstützung verursacht die Verwendung `ST_Segmentize('LINESTRING(2, 3 4)', 0.5)` einen mehrdeutigen Funktionsfehler. Die Eingabe muss korrekt als Geometrie oder Geografie eingegeben werden. Verwenden Sie `ST_GeomFromText`, `ST_GeogFromText` oder eine Umwandlung in den erforderlichen Typ (z.B. `ST_Segmentize('LINESTRING(2, 3 4)::geometry, 0.5)` )

**Beispiele**

Segmentierung einer Zeile. Lange Segmente werden gleichmäßig aufgeteilt, kurze Segmente werden nicht aufgeteilt.

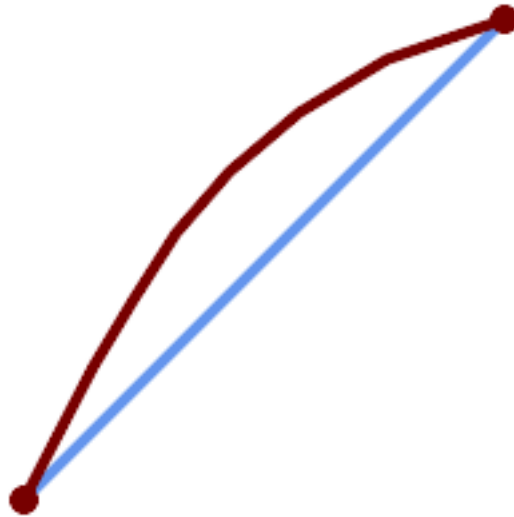
```
SELECT ST_AsText(ST_Segmentize(
  'MULTILINESTRING((0 0, 0 1, 0 9),(1 10, 1 18))'::geometry,
  5 ));
-----
MULTILINESTRING((0 0,0 1,0 5,0 9),(1 10,1 14,1 18))
```

Segmentierung eines Polygons:

```
SELECT ST_AsText(
  ST_Segmentize(('POLYGON((0 0, 0 8, 30 0, 0 0))'::geometry), 10));
-----
POLYGON((0 0,0 8,7.5 6,15 4,22.5 2,30 0,20 0,10 0,0 0))
```

Segmentierung einer geografischen Linie unter Verwendung einer maximalen Segmentlänge von 2000 Kilometern. Scheitelpunkte werden entlang des Großkreisbogens hinzugefügt, der die Endpunkte verbindet.

```
SELECT ST_AsText(
  ST_Segmentize(('LINESTRING (0 0, 60 60)'::geography), 2000000));
-----
LINESTRING(0 0,4.252632294621186 8.43596525986862,8.69579947419404 ↔
  16.824093489701564,13.550465473227048 25.107950473646188,19.1066053508691 ↔
  33.21091076089908,25.779290201459894 41.01711439406505,34.188839517966954 ↔
  48.337222885886,45.238153936612264 54.84733442373889,60 60)
```



*Eine geografische Linie, die entlang eines Großkreisbogens segmentiert ist*

#### Siehe auch

[ST\\_LineSubstring](#)

### 7.5.30 ST\_SetPoint

ST\_SetPoint — Einen Punkt eines Linienzuges durch einen gegebenen Punkt ersetzen.

#### Synopsis

geometry **ST\_SetPoint**(geometry linestring, integer zerobasedposition, geometry point);

#### Beschreibung

Ersetzt den Punkt N eines Linienzuges mit dem gegebenen Punkt. Der Index beginnt mit 0. Negative Indizes werden rückwärts gezählt, 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



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

#### Beispiele

```
--Change first point in line string from -1 3 to -1 1
SELECT ST_AsText(ST_SetPoint('LINESTRING(-1 2,-1 3)', 0, 'POINT(-1 1)'));
          st_astext
-----
LINESTRING(-1 1,-1 3)

---Change last point in a line string (lets play with 3d linestring this time)
SELECT ST_AsEWKT(ST_SetPoint(foo.geom, ST_NumPoints(foo.geom) - 1, ST_GeomFromEWKT('POINT ↵
(-1 1 3)')))
```

```

FROM (SELECT ST_GeomFromEWKT('LINESTRING(-1 2 3,-1 3 4, 5 6 7)') As geom) As foo;
-----
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;
-----
LINESTRING(0 0,1 1,0 0,3 3,4 4)

```

**Siehe auch**

[ST\\_AddPoint](#), [ST\\_NPoints](#), [ST\\_NumPoints](#), [ST\\_PointN](#), [ST\\_RemovePoint](#)

**7.5.31 ST\_ShiftLongitude**

ST\_ShiftLongitude — Verschiebt die Längenkoordinaten einer Geometrie zwischen -180..180 und 0..360.

**Synopsis**

geometry **ST\_ShiftLongitude**(geometry geom);

**Beschreibung**

Liest jeden Punkt/Vertex in einer Geometrie und verschiebt seine Längenkoordinate von -180..0 auf 180..360 und umgekehrt, wenn sie zwischen diesen Bereichen liegt. Diese Funktion ist symmetrisch, so dass das Ergebnis eine 0..360-Darstellung von -180..180-Daten und eine -180..180-Darstellung von 0..360-Daten ist.

**Note**

Dies ist nur für Daten mit Koordinaten in geografischer Länge/Breite nützlich, z. B. SRID 4326 (WGS 84 geografisch)

**Warning**

Pre-1.3.4 Aufgrund eines Bugs funktionierte dies für MULTIPOINT nicht. Mit 1.3.4+ funktioniert es auch mit MULTIPOINT.



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

Erweiterung: Mit 2.0.0 wurde die Unterstützung für polyedrische Oberflächen und TIN eingeführt.

Anmerkung: Vor 2.2.0 hieß diese Funktion "ST\_Shift\_Longitude"



Diese Funktion unterstützt polyedrische Flächen.



Diese Funktion unterstützt Dreiecke und dreieckige unregelmäßige Netzoberflächen (TIN).

## Beispiele

```
--single point forward transformation
SELECT ST_AsText(ST_ShiftLongitude('SRID=4326;POINT(270 0)::geometry))

st_astext
-----
POINT(-90 0)

--single point reverse transformation
SELECT ST_AsText(ST_ShiftLongitude('SRID=4326;POINT(-90 0)::geometry))

st_astext
-----
POINT(270 0)

--for linestrings the functions affects only to the sufficient coordinates
SELECT ST_AsText(ST_ShiftLongitude('SRID=4326;LINESTRING(174 12, 182 13)::geometry))

st_astext
-----
LINESTRING(174 12,-178 13)
```

## Siehe auch

[ST\\_WrapX](#)

### 7.5.32 ST\_WrapX

ST\_WrapX — Versammelt eine Geometrie um einen X-Wert

#### Synopsis

```
geometry ST_WrapX(geometry geom, float8 wrap, float8 move);
```

#### Beschreibung

Diese Funktion teilt die eingegebenen Geometrien auf und verschiebt dann jede resultierende Komponente, die rechts (bei negativem 'move') oder links (bei positivem 'move') von der gegebenen 'wrap'-Linie fällt, in die durch den 'move'-Parameter angegebene Richtung und fügt die Teile schließlich wieder zusammen.



#### Note

Nützlich, um eine Eingabe in Länge und Breite neu zu zentrieren, damit die wesentlichen Geobjekte nicht von einer Seite bis zur anderen abgebildet werden.

Verfügbarkeit: 2.3.0 erfordert GEOS



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

## Beispiele

```
-- Move all components of the given geometries whose bounding box
-- falls completely on the left of x=0 to +360
select ST_WrapX(geom, 0, 360);

-- Move all components of the given geometries whose bounding box
-- falls completely on the left of x=-30 to +360
select ST_WrapX(geom, -30, 360);
```

## Siehe auch

[ST\\_ShiftLongitude](#)

### 7.5.33 ST\_SnapToGrid

ST\_SnapToGrid — Fängt alle Punkte der Eingabegeometrie auf einem regelmäßigen Gitter.

#### Synopsis

```
geometry ST_SnapToGrid(geometry geomA, float originX, float originY, float sizeX, float sizeY);
geometry ST_SnapToGrid(geometry geomA, float sizeX, float sizeY);
geometry ST_SnapToGrid(geometry geomA, float size);
geometry ST_SnapToGrid(geometry geomA, geometry pointOrigin, float sizeX, float sizeY, float sizeZ, float sizeM);
```

#### Beschreibung

Variante 1, 2 und 3: Fängt alle Punkte der Eingabegeometrie auf den Gitterpunkten, die durch Ursprung und Gitterkästchengröße festgelegt sind. Aufeinanderfolgende Punkte, die in dasselbe Gitterkästchen fallen, werden gelöscht, wobei NULL zurückgegeben wird, wenn nicht mehr genug Punkte für den jeweiligen geometrischen Datentyp vorhanden sind. Collapsed geometries in a collection are stripped from it. Kollabierte Geometrien einer Kollektion werden von dieser entfernt. Nützlich um die Genauigkeit zu verringern.

Variante 4: wurde mit 1.1.0 eingeführt - Fängt alle Punkte der Eingabegeometrie auf den Gitterpunkten, welche durch den Ursprung des Gitters (der zweite Übergabewert muss ein Punkt sein) und die Gitterkästchengröße bestimmt sind. Geben Sie 0 als Größe für jene Dimension an, die nicht auf den Gitterpunkten gefangen werden soll.



#### Note

Die zurückgegebene Geometrie kann ihre Simplität 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 Eingabegeometrie, wobei höhere Dimensionen unangetastet bleiben. Verwenden Sie die Version, welche einen zweiten geometrischen Übergabewert annimmt, um sämtliche Grid-Dimensionen zu bestimmen.

Verfügbarkeit: 1.0.0RC1

Verfügbarkeit: 1.1.0, Unterstützung für Z und M



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

## Beispiele

```
--Snap your geometries to a precision grid of 10^-3
UPDATE mytable
  SET geom = ST_SnapToGrid(geom, 0.001);

SELECT ST_AsText(ST_SnapToGrid(
  ST_GeomFromText('LINESTRING(1.1115678 2.123, 4.111111 3.2374897, ←
    4.11112 3.23748667)'),
  0.001)
  );
          st_astext
-----
LINESTRING(1.112 2.123,4.111 3.237)
--Snap a 4d geometry
SELECT ST_AsEWKT(ST_SnapToGrid(
  ST_GeomFromEWKT('LINESTRING(-1.1115678 2.123 2.3456 1.11111,
    4.111111 3.2374897 3.1234 1.1111, -1.11111112 2.123 2.3456 1.111112)'),
  ST_GeomFromEWKT('POINT(1.12 2.22 3.2 4.4444)'),
  0.1, 0.1, 0.1, 0.01) );
                                     st_asewkt
-----
LINESTRING(-1.08 2.12 2.3 1.1144,4.12 3.22 3.1 1.1144,-1.08 2.12 2.3 1.1144)

--With a 4d geometry - the ST_SnapToGrid(geom,size) only touches x and y coords but keeps m ←
  and z the same
SELECT ST_AsEWKT(ST_SnapToGrid(ST_GeomFromEWKT('LINESTRING(-1.1115678 2.123 3 2.3456,
    4.111111 3.2374897 3.1234 1.1111)'),
  0.01) );
                                     st_asewkt
-----
LINESTRING(-1.11 2.12 3 2.3456,4.11 3.24 3.1234 1.1111)
```

## Siehe auch

[ST\\_Snap](#), [ST\\_AsEWKT](#), [ST\\_AsText](#), [ST\\_GeomFromText](#), [ST\\_GeomFromEWKT](#), [ST\\_Simplify](#)

### 7.5.34 ST\_Snap

**ST\_Snap** — Fängt die Segmente und Knoten einer Eingabegeometrie an den Knoten einer Referenzgeometrie.

#### Synopsis

geometry **ST\_Snap**(geometry input, geometry reference, float tolerance);

#### Beschreibung

Fängt die Knoten und Segmente einer Geometrie an den Knoten einer anderen Geometrie. Eine Entfernungstoleranz bestimmt, wo das Fangen durchgeführt wird. Die Ergebnisgeometrie ist die Eingabegeometrie mit gefangenen Knoten. Wenn kein Fangen auftritt, wird die Eingabegeometrie unverändert ausgegeben..

Eine Geometrie an einer anderen zu fangen, kann die Robustheit von Überlagerungs-Operationen verbessern, indem nahe zusammenfallende Kanten beseitigt werden (diese verursachen Probleme bei der Knoten- und Verschneidungsberechnung).

Übermäßiges Fangen kann zu einer invaliden Topologie führen. Die Anzahl und der Ort an dem Knoten sicher gefangen werden können wird mittels Heuristik bestimmt. Dies kann allerdings dazu führen, dass einige potentielle Knoten nicht gefangen werden.

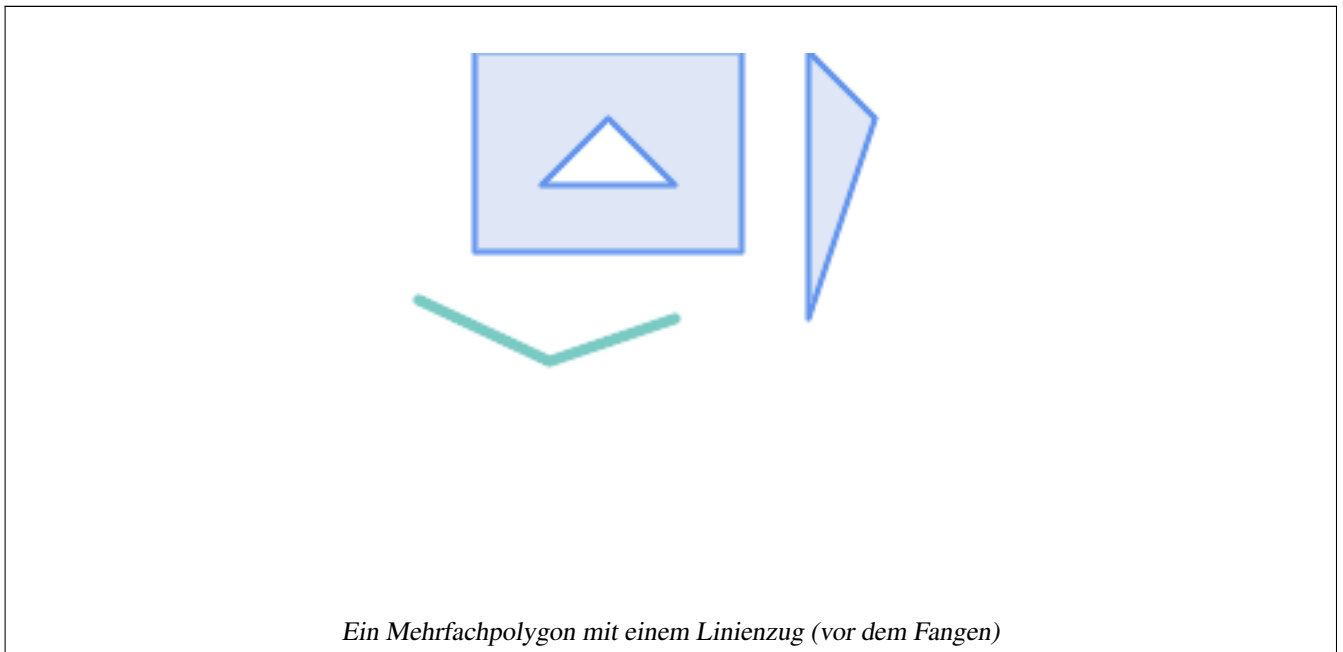


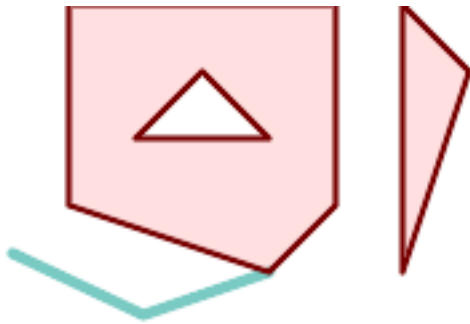
**Note**

Die zurückgegebene Geometrie kann ihre Simplizität (see [ST\\_IsSimple](#)) und Validität (see [ST\\_IsValid](#)) verlieren.

Wird vom GEOS Modul ausgeführt

Verfügbarkeit: 2.0.0

**Beispiele**

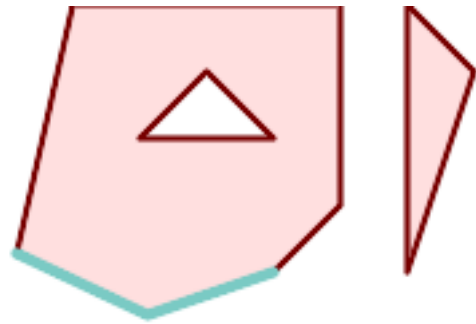


*Ein Mehrfachpolygon das an einem Linienzug gefangen wird; die Toleranz beträgt 1.01 der Entfernung. Das neue Mehrfachpolygon wird mit dem betreffenden Linienzug angezeigt.*

```
SELECT ST_AsText(ST_Snap(poly,line, ←
    ST_Distance(poly,line)*1.01)) AS polysnapped
FROM (SELECT
    ST_GeomFromText('MULTIPOLYGON(
        ((26 125, 26 200, 126 200, 126 125, ←
        26 125 ),
        ( 51 150, 101 150, 76 175, 51 150 ) ←
        ),
        (( 151 100, 151 200, 176 175, 151 ←
        100 )))') As poly,
    ST_GeomFromText('LINESTRING (5 ←
    107, 54 84, 101 100)') As line
    ) As foo;

                                polysnapped
```

```
MULTIPOLYGON(((26 125,26 200,126 200,126 ←
    125,101 100,26 125),
(51 150,101 150,76 175,51 150)),((151 ←
    100,151 200,176 175,151 100)))
```

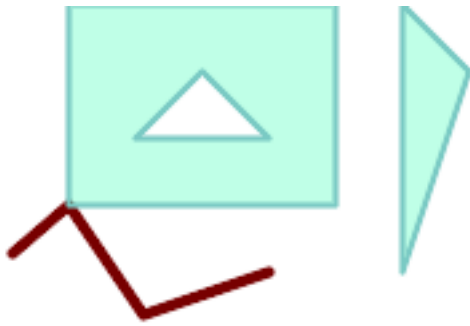


*Ein Mehrfachpolygon das an einem Linienzug gefangen wird; die Toleranz beträgt 1.25 der Entfernung. Das neue Mehrfachpolygon wird mit dem betreffenden Linienzug angezeigt.*

```
SELECT ST_AsText (
    ST_Snap(poly,line, ST_Distance(poly, ←
    line)*1.25)
    ) AS polysnapped
FROM (SELECT
    ST_GeomFromText('MULTIPOLYGON(
        (( 26 125, 26 200, 126 200, 126 125, ←
        26 125 ),
        ( 51 150, 101 150, 76 175, 51 150 ) ←
        ),
        (( 151 100, 151 200, 176 175, 151 ←
        100 )))') As poly,
    ST_GeomFromText('LINESTRING (5 ←
    107, 54 84, 101 100)') As line
    ) As foo;

                                ← polysnapped
```

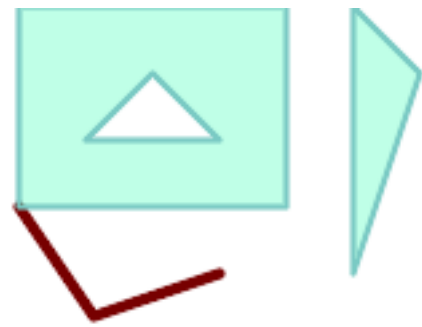
```
MULTIPOLYGON(((5 107,26 200,126 200,126 ←
    125,101 100,54 84,5 107),
(51 150,101 150,76 175,51 150)),((151 ←
    100,151 200,176 175,151 100)))
```



*Ein Linienzug der an dem ursprünglichen Mehrfachpolygon gefangen wird; die Toleranz beträgt 1.01 der Entfernung. Das neue Linienzug wird mit dem betreffenden Mehrfachpolygon angezeigt.*

```
SELECT ST_AsText(
  ST_Snap(line, poly, ST_Distance(poly, ↵
    line)*1.01)
) AS linesnapped
FROM (SELECT
  ST_GeomFromText('MULTIPOLYGON(
    ((26 125, 26 200, 126 200, 126 125, ↵
    26 125),
    (51 150, 101 150, 76 175, 51 150 )) ↵
  ',
  ((151 100, 151 200, 176 175, 151 ↵
  100)))') As poly,
  ST_GeomFromText('LINESTRING (5 ↵
  107, 54 84, 101 100)') As line
) As foo;

                linesnapped
-----
LINESTRING(5 107,26 125,54 84,101 100)
```



*Ein Linienzug der an dem ursprünglichen Mehrfachpolygon gefangen wird; die Toleranz beträgt 1.25 der Entfernung. Das neue Linienzug wird mit dem betreffenden Mehrfachpolygon angezeigt.*

```
SELECT ST_AsText(
  ST_Snap(line, poly, ST_Distance(poly, ↵
    line)*1.25)
) AS linesnapped
FROM (SELECT
  ST_GeomFromText('MULTIPOLYGON(
    (( 26 125, 26 200, 126 200, 126 125, ↵
    26 125 ),
    (51 150, 101 150, 76 175, 51 150 )) ↵
  ',
  ((151 100, 151 200, 176 175, 151 ↵
  100 )))') As poly,
  ST_GeomFromText('LINESTRING (5 ↵
  107, 54 84, 101 100)') As line
) As foo;

                linesnapped
-----
LINESTRING(26 125,54 84,101 100)
```

#### Siehe auch

[ST\\_SnapToGrid](#)

### 7.5.35 ST\_SwapOrdinates

`ST_SwapOrdinates` — Gibt eine Version der Ausgangsgeometrie zurück, in der die angegebenen Ordinatenwerte ausgetauscht werden.

#### Synopsis






geometry `ST_SwapOrdinates`(geometry geom, cstring ords);

## Beschreibung

Gibt eine Version der Ausgangsgeometrie zurück, in der die angegebenen Ordinaten ausgetauscht werden.

Der `ords` Parameter ist eine Zeichenkette aus 2 Zeichen, welche die Ordinate benennt die getauscht werden soll. Gültige Bezeichnungen sind: x,y,z und m.

Verfügbarkeit: 2.2.0

-  Diese Methode unterstützt kreisförmige Strings und Kurven.
-  Diese Funktion unterstützt 3d und lässt den Z-Index nicht fallen.
-  Diese Funktion unterstützt M-Koordinaten.
-  Diese Funktion unterstützt polyedrische Flächen.
-  Diese Funktion unterstützt Dreiecke und dreieckige unregelmäßige Netzoberflächen (TIN).

## Beispiel

```
-- 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;
-----
POINT ZM (0 0 0 4)
```

## Siehe auch

[ST\\_FlipCoordinates](#)

## 7.6 Geometrieverifizierung

### 7.6.1 ST\_IsValid

`ST_IsValid` — Prüft, ob eine Geometrie in 2D wohlgeformt ist.

#### Synopsis

```
boolean ST_IsValid(geometry g);
boolean ST_IsValid(geometry g, integer flags);
```

## Beschreibung

Prüft, ob ein `ST_Geometry`-Wert wohlgeformt und in 2D gemäß den OGC-Regeln gültig ist. Bei Geometrien mit 3 und 4 Dimensionen wird die Gültigkeit weiterhin nur in 2 Dimensionen geprüft. Für Geometrien, die ungültig sind, wird ein PostgreSQL-NOTICE ausgegeben, der die Gründe für die Ungültigkeit angibt.

Für die Version mit dem Parameter `flags` sind die unterstützten Werte in `ST_IsValidDetail` dokumentiert. Diese Version druckt keinen HINWEIS, der die Ungültigkeit erklärt.

Weitere Informationen über die Definition der Geometriegültigkeit finden Sie unter Section 4.4



### Note

SQL-MM definiert das Ergebnis von `ST_IsValid(NULL)` als 0, während PostGIS NULL zurückgibt.

Wird vom GEOS Modul ausgeführt

Die Version, die Flaggen akzeptiert, ist ab 2.0.0 verfügbar.



Diese Methode implementiert die [OGC Simple Features Implementation Specification for SQL 1.1](#).



Diese Methode setzt die SQL/MM-Spezifikation um. SQL-MM 3: 5.1.9



### Note

Weder die OGC-SFS- noch die SQL-MM-Spezifikationen enthalten ein Flag-Argument für `ST_IsValid`. Das 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](#),

## 7.6.2 ST\_IsValidDetail

`ST_IsValidDetail` — Gibt eine Zeile `valid_detail` zurück, die angibt, ob eine Geometrie gültig ist oder, falls nicht, einen Grund und einen Ort.

## Synopsis

`valid_detail` `ST_IsValidDetail`(geometry geom, integer flags);

## Beschreibung

Gibt eine Zeile `valid_detail` zurück, die einen booleschen Wert (`valid`) enthält, der angibt, ob eine Geometrie gültig ist, einen varchar-Wert (`reason`), der einen Grund angibt, warum sie ungültig ist, und eine Geometrie (`location`), die angibt, wo sie ungültig ist.

Nützlich zur Verbesserung der Kombination von `ST_IsValid` und `ST_IsValidReason`, um einen detaillierten Bericht über ungültige Geometrien zu erstellen.

Der optionale Parameter `flags` ist ein Bitfeld. Er kann die folgenden Werte haben:

- 0: Verwendung der üblichen OGC SFS-Gültigkeitssemantik.
- 1: Bestimmte Arten von sich selbst berührenden Ringen (umgekehrte Schalen und umgekehrte Löcher) als gültig betrachten. Dies ist auch als "ESRI-Flagge" bekannt, da dies das von diesen Werkzeugen verwendete Gültigkeitsmodell ist. Beachten Sie, dass dies nach dem OGC-Modell ungültig ist.

Wird vom GEOS Modul ausgeführt

Verfügbarkeit: 2.0.0

## Beispiele

```
--First 3 Rejects from a successful quintuplet experiment
SELECT gid, reason(ST_IsValidDetail(geom)), ST_AsText(location(ST_IsValidDetail(geom))) as location
FROM
(SELECT ST_MakePolygon(ST_ExteriorRing(e.buff), array_agg(f.line)) As geom, gid
FROM (SELECT ST_Buffer(ST_Point(x1*10,y1), z1) As buff, x1*10 + y1*100 + z1*1000 As gid
      FROM generate_series(-4,6) x1
      CROSS JOIN generate_series(2,5) y1
      CROSS JOIN generate_series(1,8) z1
      WHERE x1
> y1*0.5 AND z1 < x1*y1) As e
      INNER JOIN (SELECT ST_Translate(ST_ExteriorRing(ST_Buffer(ST_Point(x1*10,y1), z1)),
      y1*1, z1*2) As line
      FROM generate_series(-3,6) x1
      CROSS JOIN generate_series(2,5) y1
      CROSS JOIN generate_series(1,10) z1
      WHERE x1
> y1*0.75 AND z1 < x1*y1) As f
ON (ST_Area(e.buff)
> 78 AND ST_Contains(e.buff, f.line))
GROUP BY gid, e.buff) As quintuplet_experiment
WHERE ST_IsValid(geom) = false
ORDER BY gid
LIMIT 3;
```

gid	reason	location
5330	Self-intersection	POINT(32 5)
5340	Self-intersection	POINT(42 5)
5350	Self-intersection	POINT(52 5)

```
--simple example
SELECT * FROM ST_IsValidDetail('LINESTRING(220227 150406,2220227 150407,222020 150410)');
```

valid	reason	location
t		

**Siehe auch**

[ST\\_IsValid](#), [ST\\_IsValidReason](#)

**7.6.3 ST\_IsValidReason**

`ST_IsValidReason` — Gibt einen Text zurück, der angibt, ob eine Geometrie gültig ist, oder einen Grund für die Ungültigkeit.

**Synopsis**

```
text ST_IsValidReason(geometry geomA);
text ST_IsValidReason(geometry geomA, integer flags);
```

**Beschreibung**

Gibt einen Text zurück, der angibt, ob eine Geometrie gültig ist, oder, falls ungültig, einen Grund dafür.

Nützlich in Kombination mit [ST\\_IsValid](#), um einen detaillierten Bericht über ungültige Geometrien und Gründe zu erstellen.

Erlaubte Flaggen sind in [ST\\_IsValidDetail](#) dokumentiert.

Wird vom GEOS Modul ausgeführt

Verfügbarkeit: 1.4

Verfügbarkeit: Version 2.0 mit Flaggen.

**Beispiele**

```
-- invalid bow-tie polygon
SELECT ST_IsValidReason(
    'POLYGON ((100 200, 100 100, 200 200,
              200 100, 100 200))'::geometry) as validity_info;
validity_info
-----
Self-intersection[150 150]
```

```
--First 3 Rejects from a successful quintuplet experiment
SELECT gid, ST_IsValidReason(geom) as validity_info
FROM
  (SELECT ST_MakePolygon(ST_ExteriorRing(e.buff), array_agg(f.line)) As geom, gid
  FROM (SELECT ST_Buffer(ST_Point(x1*10,y1), z1) As buff, x1*10 + y1*100 + z1*1000 As gid
        FROM generate_series(-4,6) x1
        CROSS JOIN generate_series(2,5) y1
        CROSS JOIN generate_series(1,8) z1
        WHERE x1
  > y1*0.5 AND z1 < x1*y1) As e
  INNER JOIN (SELECT ST_Translate(ST_ExteriorRing(ST_Buffer(ST_Point(x1*10,y1), z1)), ←
    y1*1, z1*2) As line
  FROM generate_series(-3,6) x1
  CROSS JOIN generate_series(2,5) y1
  CROSS JOIN generate_series(1,10) z1
  WHERE x1
```

```

> y1*0.75 AND z1 < x1*y1) As f
ON (ST_Area(e.buff)
> 78 AND ST_Contains(e.buff, f.line))
GROUP BY gid, e.buff) As quintuplet_experiment
WHERE ST_IsValid(geom) = false
ORDER BY gid
LIMIT 3;

gid |          validity_info
-----+-----
5330 | Self-intersection [32 5]
5340 | Self-intersection [42 5]
5350 | Self-intersection [52 5]

--simple example
SELECT ST_IsValidReason('LINESTRING(220227 150406,2220227 150407,222020 150410)');

st_isvalidreason
-----
Valid Geometry

```

**Siehe auch**

[ST\\_IsValid](#), [ST\\_Summary](#)

**7.6.4 ST\_MakeValid**

ST\_MakeValid — Versucht, eine ungültige Geometrie gültig zu machen, ohne dass Scheitelpunkte verloren gehen.

**Synopsis**

```

geometry ST_MakeValid(geometry input);
geometry ST_MakeValid(geometry input, text params);

```

**Beschreibung**

Die Funktion versucht, eine gültige Darstellung einer gegebenen ungültigen Geometrie zu erstellen, ohne dass einer der Eingabepunkte verloren geht. Gültige Geometrien werden unverändert zurückgegeben.

Unterstützte Eingaben sind: PUNKTE, MEHRPUNKTE, LINIENSTRÄNGE, MEHRLINIENSTRÄNGE, POLYGONEN, MEHRPOLYGONEN und GEOMETRIESAMMLUNGEN, die eine beliebige Mischung dieser Elemente enthalten.

Bei vollständiger oder teilweiser Dimensionsreduzierung kann die Ausgangsgeometrie eine Sammlung von Geometrien mit geringerer bis gleicher Dimension oder eine Geometrie mit geringerer Dimension sein.

Einzelne Polygone können im Falle von Selbstüberschneidungen zu Multi-Geometrien werden.

Mit dem Argument `params` kann eine Optionszeichenfolge angegeben werden, mit der die Methode für die Erstellung gültiger Geometrien ausgewählt wird. Die Optionszeichenfolge hat das Format "method=lineworklstructure keepcollapsed=true|false". Wird kein "params"-Argument angegeben, wird standardmäßig der "linework"-Algorithmus verwendet.

Der Schlüssel "Methode" hat zwei Werte.

- "linework" ist der ursprüngliche Algorithmus, der gültige Geometrien erstellt, indem er zunächst alle Linien extrahiert, diese Linien zusammen kodiert und dann einen Wert aus dem Linienwerk ausgibt.



- "Struktur" ist ein Algorithmus, der zwischen inneren und äußeren Ringen unterscheidet und eine neue Geometrie erstellt, indem er äußere Ringe zusammenfasst und dann alle inneren Ringe differenziert.

Der Schlüssel "keepcollapsed" ist nur für den Algorithmus "structure" gültig und nimmt den Wert "true" oder "false" an. Wenn er auf "false" gesetzt ist, werden Geometriekomponenten, die auf eine niedrigere Dimensionalität kollabieren, z. B. ein Ein-Punkt-Linienzug, fallengelassen.

Wird vom GEOS Modul ausgeführt

Verfügbarkeit: 2.0.0

Verbessert: 2.0.1, Geschwindigkeitsverbesserungen

Verbessert: 2.1.0, Unterstützung für GEOMETRYCOLLECTION und MULTIPOINT hinzugefügt.

Verbessert: 3.1.0, Entfernen von Koordinaten mit NaN-Werten hinzugefügt.

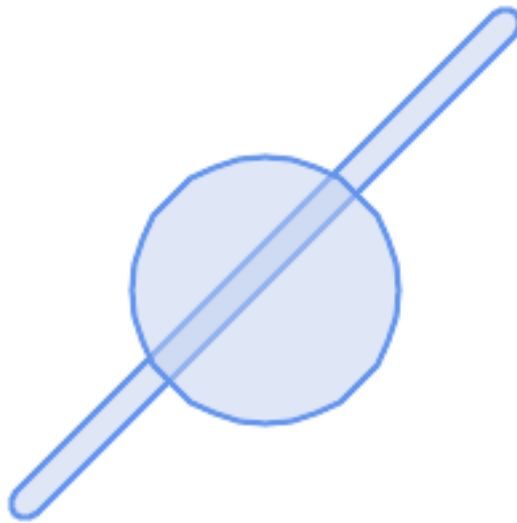
Verbessert: 3.2.0, zusätzliche Algorithmus-Optionen, 'Linienwerk' und 'Struktur', die GEOS  $\geq$  3.10.0 erfordern.



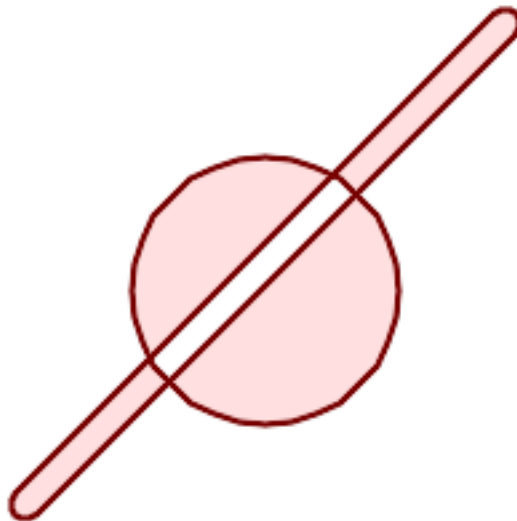
Diese Funktion unterstützt 3d und lässt den Z-Index nicht fallen.

## Beispiele

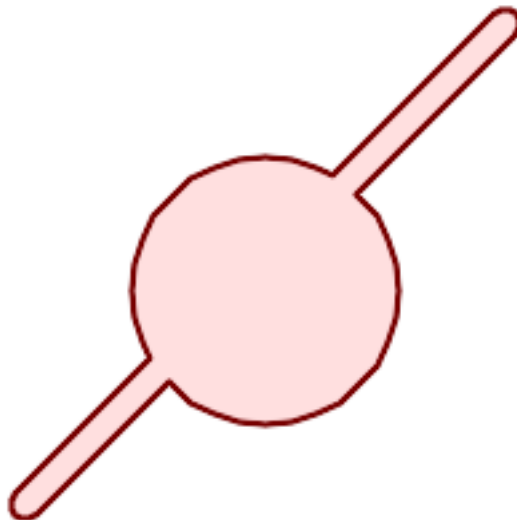
---



*vor\_geom: MULTIPOLYGON aus 2 sich überlappenden Polygonen*



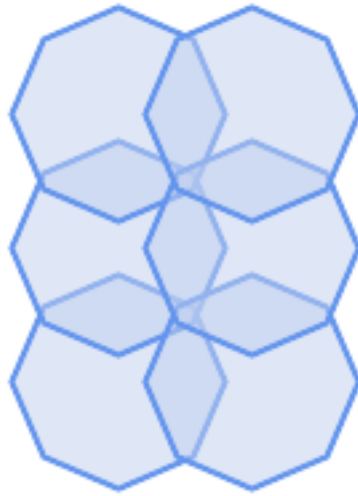
*nach\_geom: MULTIPOLYGON aus 4 sich nicht überlappenden Polygonen*



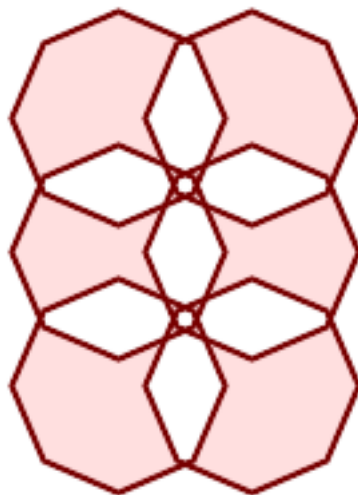
*after\_geom\_structure: MULTIPOLYGON aus 1 nicht überlappenden Polygon*

```
SELECT f.geom AS before_geom, ST_MakeValid(f.geom) AS after_geom, ST_MakeValid(f.geom, ←
    'method=structure') AS after_geom_structure
FROM (SELECT 'MULTIPOLYGON(((186 194,187 194,188 195,189 195,190 195,
191 195 192 195 193 194 194 194 194 194 193 195 192 195 191
```

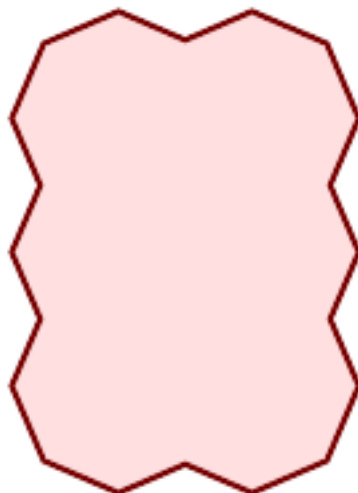




*vor\_geom: MULTIPOLYGON aus 6 sich überschneidenden Polygonen*



*nach\_geom: MULTIPOLYGON aus 14 nicht überlappenden Polygonen*



*after\_geom\_structure: MULTIPOLYGON aus 1 nicht überlappendem Polygon*

```
SELECT c.geom AS before_geom,
       ST_MakeValid(c.geom) AS after_geom,
       ST_MakeValid(c.geom, 'method=structure') AS after_geom_structure
FROM (SELECT 'MULTIPOLYGON(((91 50,79 22,51 10,23 22,11 50,23 78,51 90,79 78,91 ↵
```

## Beispiele

```
SELECT ST_AsText(ST_MakeValid(
  'LINESTRING(0 0, 0 0)',
  'method=structure keepcollapsed=true'
));

st_astext
-----
POINT(0 0)

SELECT ST_AsText(ST_MakeValid(
  'LINESTRING(0 0, 0 0)',
  'method=structure keepcollapsed=false'
));

st_astext
-----
LINESTRING EMPTY
```

## Siehe auch

[ST\\_IsValid](#), [ST\\_Collect](#), [ST\\_CollectionExtract](#)

## 7.7 Funktionen des räumlichen Bezugssystems

### 7.7.1 ST\_InverseTransformPipeline

`ST_InverseTransformPipeline` — Rückgabe einer neuen Geometrie mit in ein anderes räumliches Bezugssystem transformierten Koordinaten unter Verwendung der Umkehrung einer definierten Koordinatentransformationspipeline.

#### Synopsis

geometry `ST_InverseTransformPipeline`(geometry geom, text pipeline, integer to\_srid);

#### Beschreibung

Rückgabe einer neuen Geometrie mit in ein anderes räumliches Bezugssystem transformierten Koordinaten unter Verwendung einer definierten Koordinatentransformationspipeline, die in umgekehrter Richtung verläuft.

Einzelheiten zum Schreiben einer Transformationspipeline finden Sie unter [ST\\_TransformPipeline](#).

Verfügbarkeit: 3.4.0

Die SRID der Eingabegeometrie wird ignoriert, und die SRID der Ausgabegeometrie wird auf Null gesetzt, sofern nicht über den optionalen Parameter `to_srid` ein Wert angegeben wird. Bei Verwendung von [ST\\_TransformPipeline](#) wird die Pipeline in Vorwärtsrichtung ausgeführt. Bei Verwendung von `ST_InverseTransformPipeline()` wird die Pipeline in umgekehrter Richtung ausgeführt.

Transformationen unter Verwendung von Pipelines sind eine spezialisierte Version von [ST\\_Transform](#). In den meisten Fällen wählt "ST\_Transform" die richtigen Operationen für die Konvertierung zwischen Koordinatensystemen und sollte daher bevorzugt werden.

## Beispiele

Ändern Sie WGS 84 long lat in UTM 31N unter Verwendung der Konvertierung EPSG:16031

```
-- Inverse direction
SELECT ST_AsText(ST_InverseTransformPipeline('POINT(426857.9877165967 5427937.523342293)'::←
  geometry,
  'urn:ogc:def:coordinateOperation:EPSG::16031')) AS wgs_geom;

          wgs_geom
-----
POINT(2 48.99999999999999)
(1 row)
```

Beispiel GDA2020.

```
-- using ST_Transform with automatic selection of a conversion pipeline.
SELECT ST_AsText(ST_Transform('SRID=4939;POINT(143.0 -37.0)'::geometry, 7844)) AS ←
  gda2020_auto;

          gda2020_auto
-----
POINT(143.00000635638918 -36.999986706128176)
(1 row)
```

## Siehe auch

[ST\\_Transform](#), [ST\\_TransformPipeline](#)

### 7.7.2 ST\_SetSRID

ST\_SetSRID — Legen Sie den SRID für eine Geometrie fest.

#### Synopsis

geometry **ST\_SetSRID**(geometry geom, integer srid);

#### Beschreibung

Setzt den SRID einer Geometrie auf einen bestimmten Integer-Wert. Nützlich bei der Erstellung von Begrenzungsrahmen für Abfragen.



#### Note

Diese Funktion transformiert die Geometriekoordinaten in keiner Weise - sie setzt lediglich die Metadaten, die das räumliche Bezugssystem definieren, in dem die Geometrie angenommen wird. Verwenden Sie [ST\\_Transform](#), wenn Sie die Geometrie in eine neue Projektion transformieren möchten.



Diese Methode implementiert die [OGC Simple Features Implementation Specification for SQL 1.1](#).



Diese Methode unterstützt kreisförmige Strings und Kurven.

## Beispiele

-- Markieren Sie einen Punkt als WGS 84 long lat --

```
SELECT ST_SetSRID(ST_Point(-123.365556, 48.428611),4326) As wgs84long_lat;
-- the ewkt representation (wrap with ST_AsEWKT) -
SRID=4326;POINT(-123.365556 48.428611)
```

-- Markieren Sie einen Punkt als WGS 84 long lat und transformieren Sie ihn dann in den Web-Mercator (Spherical Mercator).

```
SELECT ST_Transform(ST_SetSRID(ST_Point(-123.365556, 48.428611),4326),3785) As spere_merc;
-- the ewkt representation (wrap with ST_AsEWKT) -
SRID=3785;POINT(-13732990.8753491 6178458.96425423)
```

## Siehe auch

Section [4.5](#), [ST\\_SRID](#), [ST\\_Transform](#), [UpdateGeometrySRID](#)

## 7.7.3 ST\_SRID

ST\_SRID — Gibt die Raumbezugskennung für eine Geometrie zurück.

### Synopsis

integer **ST\_SRID**(geometry g1);

### Beschreibung

Liefert die Kennung des Raumbezugs für die ST\_Geometrie, wie in der Tabelle spatial\_ref\_sys definiert. Section [4.5](#)



#### Note

Die Tabelle spatial\_ref\_sys ist eine Tabelle, die alle PostGIS bekannten räumlichen Bezugssysteme katalogisiert und für Transformationen von einem räumlichen Bezugssystem in ein anderes verwendet wird. Daher ist es wichtig, dass Sie die richtige Kennung für das räumliche Bezugssystem haben, wenn Sie Ihre Geometrien transformieren möchten.



Diese Methode implementiert die [OGC Simple Features Implementation Specification for SQL 1.1](#). s2.1.1.1



Diese Methode setzt die SQL/MM-Spezifikation um. SQL-MM 3: 5.1.5



Diese Methode unterstützt kreisförmige Strings und Kurven.

## Beispiele

```
SELECT ST_SRID(ST_GeomFromText('POINT(-71.1043 42.315)',4326));
--result
4326
```

## Siehe auch

Section [4.5](#), [ST\\_SetSRID](#), [ST\\_Transform](#), [ST\\_SRID](#), [ST\\_SRID](#)

## 7.7.4 ST\_Transform

ST\_Transform — Rückgabe einer neuen Geometrie mit in ein anderes räumliches Bezugssystem transformierten Koordinaten.

### Synopsis

```
geometry ST_Transform(geometry g1, integer srid);
geometry ST_Transform(geometry geom, text to_proj);
geometry ST_Transform(geometry geom, text from_proj, text to_proj);
geometry ST_Transform(geometry geom, text from_proj, integer to_srid);
```

### Beschreibung

Gibt eine neue Geometrie zurück, deren Koordinaten in ein anderes räumliches Bezugssystem transformiert wurden. Der Ziel-Raumbezug `to_srid` kann durch einen gültigen SRID-Integer-Parameter identifiziert werden (d. h. er muss in der Tabelle `spatial_ref_sys` vorhanden sein). Alternativ kann ein als PROJ.4-String definierter Raumbezug für `to_proj` und/oder `from_proj` verwendet werden, allerdings sind diese Methoden nicht optimiert. Wenn das Ziel-Raumbezugssystem durch einen PROJ.4-String anstelle eines SRID ausgedrückt wird, wird der SRID der Ausgangsgeometrie auf Null gesetzt. Mit Ausnahme der Funktionen mit `from_proj` müssen Eingabegeometrien einen definierten SRID haben.

ST\_Transform wird oft mit [ST\\_SetSRID](#) verwechselt. ST\_Transform ändert tatsächlich die Koordinaten einer Geometrie von einem räumlichen Bezugssystem in ein anderes, während ST\_SetSRID() lediglich den SRID-Bezeichner der Geometrie ändert.

ST\_Transform wählt automatisch eine geeignete Konvertierungspipeline für das Quell- und Ziel-Raumbezugssystem aus. Um eine bestimmte Konvertierungsmethode zu verwenden, verwenden Sie [ST\\_TransformPipeline](#).



#### Note

Erfordert, dass PostGIS mit PROJ-Unterstützung kompiliert ist. Verwenden Sie [PostGIS\\_Full\\_Version](#), um zu bestätigen, dass Sie PROJ-Unterstützung einkompiliert haben.



#### Note

Wenn mehr als eine Transformation verwendet wird, ist es sinnvoll, einen funktionalen Index für die am häufigsten verwendeten Transformationen zu haben, um die Vorteile der Indexnutzung zu nutzen.



#### 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.

Verbessert: In Version 2.3.0 wurde die Unterstützung für direkten PROJ.4 Text eingeführt.



Diese Methode setzt die SQL/MM-Spezifikation um. SQL-MM 3: 5.1.6



Diese Methode unterstützt kreisförmige Strings und Kurven.



Diese Funktion unterstützt polyedrische Flächen.



## Beispiele

### Änderung der Geometrie der Ebene des Bundesstaates Massachusetts von US-Fuß auf WGS 84 long lat

```
SELECT ST_AsText(ST_Transform(ST_GeomFromText('POLYGON((743238 2967416,743238 2967450,
743265 2967450,743265.625 2967416,743238 2967416)'),2249),4326)) As wgs_geom;

wgs_geom
-----
POLYGON((-71.1776848522251 42.3902896512902,-71.1776843766326 42.3903829478009,
-71.1775844305465 42.3903826677917,-71.1775825927231 42.3902893647987,-71.177684
8522251 42.3902896512902));
(1 row)

--3D Circular String example
SELECT ST_AsEWKT(ST_Transform(ST_GeomFromEWKT('SRID=2249;CIRCULARSTRING(743238 2967416 ↵
1,743238 2967450 2,743265 2967450 3,743265.625 2967416 3,743238 2967416 4)'),4326));

st_asewkt
-----
SRID=4326;CIRCULARSTRING(-71.1776848522251 42.3902896512902 1,-71.1776843766326 ↵
42.3903829478009 2,
-71.1775844305465 42.3903826677917 3,
-71.1775825927231 42.3902893647987 3,-71.1776848522251 42.3902896512902 4)
```

Beispiel für die Erstellung eines partiellen funktionalen Indexes. Für Tabellen, bei denen Sie nicht sicher sind, dass alle Geometrien ausgefüllt werden, ist es am besten, einen partiellen Index zu verwenden, der Null-Geometrien auslässt, was sowohl Platz spart als auch Ihren Index kleiner und effizienter macht.

```
CREATE INDEX idx_geom_26986_parcel
ON parcels
USING gist
(ST_Transform(geom, 26986))
WHERE geom IS NOT NULL;
```

### Beispiele für die Verwendung von PROJ.4 Text zur Transformation mit benutzerdefinierten räumlichen Bezügen.

```
-- Find intersection of two polygons near the North pole, using a custom Gnomonic projection
-- See http://boundlessgeo.com/2012/02/flattening-the-peel/
WITH data AS (
SELECT
ST_GeomFromText('POLYGON((170 50,170 72,-130 72,-130 50,170 50)'), 4326) AS p1,
ST_GeomFromText('POLYGON((-170 68,-170 90,-141 90,-141 68,-170 68)'), 4326) AS p2,
'+proj=gnom +ellps=WGS84 +lat_0=70 +lon_0=-160 +no_defs'::text AS gnom
)
SELECT ST_AsText(
ST_Transform(
ST_Intersection(ST_Transform(p1, gnom), ST_Transform(p2, gnom)),
gnom, 4326))
FROM data;

st_astext
-----
POLYGON((-170 74.053793645338,-141 73.4268621378904,-141 68,-170 68,-170 74.053793645338) ↵
)
```

## Konfigurieren des Transformationsverhaltens

Manchmal kann eine Koordinatentransformation, die eine Rasterverschiebung beinhaltet, fehlschlagen, z.B. wenn PROJ.4 nicht mit Rasterverschiebungsdateien erstellt wurde oder die Koordinate nicht innerhalb des Bereichs liegt, für den die Rasterverschiebung definiert ist. Standardmäßig gibt PostGIS einen Fehler aus, wenn keine Rasterverschiebungsdatei vorhanden ist,

aber dieses Verhalten kann auf einer pro-SRID-Basis konfiguriert werden, indem entweder verschiedene `to_proj` Werte des PROJ.4-Textes getestet oder der `proj4text` Wert in der `spatial_ref_sys` Tabelle geändert wird.

Der `proj4text`-Parameter `+datum=NAD87` ist zum Beispiel eine Kurzform für den folgenden `+nadgrids`-Parameter:

```
+nadgrids=@conus,@alaska,@ntv2_0.gsb,@ntv1_can.dat
```

Das Präfix `@` bedeutet, dass kein Fehler gemeldet wird, wenn die Dateien nicht vorhanden sind, aber wenn das Ende der Liste erreicht wird, ohne dass eine Datei gefunden wurde (d. h. gefunden und überschneidend), wird ein Fehler ausgegeben.

Wenn Sie dagegen sicherstellen wollten, dass zumindest die Standarddateien vorhanden sind, aber wenn alle Dateien ohne Treffer durchsucht wurden, eine Nulltransformation angewendet wird, könnten Sie dies verwenden:

```
+nadgrids=@conus,@alaska,@ntv2_0.gsb,@ntv1_can.dat,null
```

Die Null-Rasterverschiebungsdatei ist eine gültige Rasterverschiebungsdatei, die die ganze Welt abdeckt und keine Verschiebung anwendet. Wenn Sie also PostGIS so ändern wollten, dass bei Transformationen zu SRID 4267, die nicht im korrekten Bereich liegen, kein ERROR ausgelöst wird, würden Sie das folgende Beispiel verwenden:

```
UPDATE spatial_ref_sys SET proj4text = '+proj=longlat +ellps=clrk66 +nadgrids=@conus, ↵
    @alaska,@ntv2_0.gsb,@ntv1_can.dat,null +no_defs' WHERE srid = 4267;
```

## Siehe auch

Section [4.5](#), [ST\\_SetSRID](#), [ST\\_SRID](#), [UpdateGeometrySRID](#), [ST\\_TransformPipeline](#)

## 7.7.5 ST\_TransformPipeline

`ST_TransformPipeline` — Rückgabe einer neuen Geometrie mit in ein anderes räumliches Bezugssystem transformierten Koordinaten unter Verwendung einer definierten Koordinatentransformationspipeline.

### Synopsis

```
geometry ST_TransformPipeline(geometry g1, text pipeline, integer to_srid);
```

### Beschreibung

Rückgabe einer neuen Geometrie mit in ein anderes räumliches Bezugssystem transformierten Koordinaten unter Verwendung einer definierten Koordinatentransformationspipeline.

Transformations-Pipelines werden mit einem der folgenden String-Formate definiert:

- `urn:ogc:def:coordinateOperation:AUTHORITY::CODE`. Beachten Sie, dass ein einfacher `EPSG:CODE` String eine Koordinatenoperation nicht eindeutig identifiziert: Der gleiche EPSG-Code kann für eine CRS-Definition verwendet werden.
- Eine PROJ-Pipeline-Zeichenkette in der Form: `+proj=pipeline ...`. Die automatische Achsennormalisierung wird nicht angewendet, und der Aufrufer muss gegebenenfalls einen zusätzlichen Pipelineschritt hinzufügen oder `axiswap` Schritte entfernen.
- Verkettete Operationen der Form: `urn:ogc:def:coordinateOperation,coordinateOperation:EPSG::3895,coo`

Verfügbarkeit: 3.4.0

Die SRID der Eingangsgeometrie wird ignoriert, und die SRID der Ausgangsgeometrie wird auf Null gesetzt, sofern nicht über den optionalen Parameter `to_srid` ein Wert angegeben wird. Bei Verwendung von `ST_TransformPipeline()` wird die Pipeline in Vorwärtsrichtung ausgeführt. Bei Verwendung von [ST\\_InverseTransformPipeline](#) wird die Pipeline in der umgekehrten Richtung ausgeführt.

Transformationen unter Verwendung von Pipelines sind eine spezialisierte Version von [ST\\_Transform](#). In den meisten Fällen wählt "ST\_Transform" die richtigen Operationen für die Konvertierung zwischen Koordinatensystemen und sollte daher bevorzugt werden.

## Beispiele

Ändern Sie WGS 84 long lat in UTM 31N unter Verwendung der Konvertierung EPSG:16031

```
-- Forward direction
SELECT ST_AsText(ST_TransformPipeline('SRID=4326;POINT(2 49)::geometry,
  'urn:ogc:def:coordinateOperation:EPSG::16031')) AS utm_geom;

          utm_geom
-----
POINT(426857.9877165967 5427937.523342293)
(1 row)

-- Inverse direction
SELECT ST_AsText(ST_InverseTransformPipeline('POINT(426857.9877165967 5427937.523342293):: ←
  geometry,
  'urn:ogc:def:coordinateOperation:EPSG::16031')) AS wgs_geom;

          wgs_geom
-----
POINT(2 48.999999999999999)
(1 row)
```

## Beispiel GDA2020.

```
-- using ST_Transform with automatic selection of a conversion pipeline.
SELECT ST_AsText(ST_Transform('SRID=4939;POINT(143.0 -37.0)::geometry, 7844)) AS ←
  gda2020_auto;

          gda2020_auto
-----
POINT(143.00000635638918 -36.999986706128176)
(1 row)

-- using a defined conversion (EPSG:8447)
SELECT ST_AsText(ST_TransformPipeline('SRID=4939;POINT(143.0 -37.0)::geometry,
  'urn:ogc:def:coordinateOperation:EPSG::8447')) AS gda2020_code;

          gda2020_code
-----
POINT(143.0000063280214 -36.999986718287545)
(1 row)

-- using a PROJ pipeline definition matching EPSG:8447, as returned from
-- 'projinfo -s EPSG:4939 -t EPSG:7844'.
-- NOTE: any 'axisswap' steps must be removed.
SELECT ST_AsText(ST_TransformPipeline('SRID=4939;POINT(143.0 -37.0)::geometry,
  '+proj=pipeline
  +step +proj=unitconvert +xy_in=deg +xy_out=rad
  +step +proj=hgridshift +grids=au_icsm_GDA94_GDA2020_conformal_and_distortion.tif
  +step +proj=unitconvert +xy_in=rad +xy_out=deg')) AS gda2020_pipeline;

          gda2020_pipeline
-----
POINT(143.0000063280214 -36.999986718287545)
(1 row)
```

## Siehe auch

[ST\\_Transform](#), [ST\\_InverseTransformPipeline](#)

### 7.7.6 `postgis_srs_codes`

`postgis_srs_codes` — Gibt die Liste der SRS-Codes zurück, die mit der angegebenen Behörde verbunden sind.

#### Synopsis

```
setof text postgis_srs_codes(text auth_name);
```

#### Beschreibung

Gibt eine Menge aller `auth_srid` für den angegebenen `auth_name` zurück.

Verfügbarkeit: 3.4.0

Proj Version 6+

#### Beispiele

Geben Sie die ersten zehn Codes für die EPSG-Behörde an.

```
SELECT * FROM postgis_srs_codes('EPSG') LIMIT 10;
```

```
postgis_srs_codes
-----
2000
20004
20005
20006
20007
20008
20009
2001
20010
20011
```

#### Siehe auch

[postgis\\_srs](#), [postgis\\_srs\\_all](#), [postgis\\_srs\\_search](#)

### 7.7.7 `postgis_srs`

`postgis_srs` — Rückgabe eines Metadatensatzes für die angefragte Behörde und srid.

#### Synopsis

```
setof record postgis_srs(text auth_name, text auth_srid);
```

#### Beschreibung

Gibt einen Metadatensatz für den angeforderten `auth_srid` für den angegebenen `auth_name` zurück. Der Datensatz enthält den `auth_name`, `auth_srid`, `sname`, `srtext`, `proj4text` und die Ecken des Nutzungsbereichs, `point_sw` und `point_ne`.

Verfügbarkeit: 3.4.0

Proj Version 6+

## Beispiele

Abrufen der Metadaten für EPSG:3005.

```
SELECT * FROM postgis_srs('EPSG', '3005');

auth_name | EPSG
auth_srid | 3005
srsname   | NAD83 / BC Albers
srtext    | PROJCS["NAD83 / BC Albers", ... ]
proj4text | +proj=aea +lat_0=45 +lon_0=-126 +lat_1=50 +lat_2=58.5 +x_0=1000000 +y_0=0 +
        datum=NAD83 +units=m +no_defs +type=crs
point_sw  | 0101000020E6100000E17A14AE476161C00000000000204840
point_ne  | 0101000020E610000085EB51B81E855CC0E17A14AE47014E40
```

## Siehe auch

[postgis\\_srs\\_codes](#), [postgis\\_srs\\_all](#), [postgis\\_srs\\_search](#)

### 7.7.8 postgis\_srs\_all

`postgis_srs_all` — Gibt Metadatensätze für jedes räumliche Bezugssystem in der zugrunde liegenden Proj-Datenbank zurück.

## Synopsis

setof Datensatz `postgis_srs_all`(void);

## Beschreibung

Gibt eine Menge aller Metadatensätze in der zugrunde liegenden Proj-Datenbank zurück. Die Datensätze haben die Bezeichnungen `auth_name`, `auth_srid`, `srsname`, `srtext`, `proj4text` und die Ecken des Verwendungsbereichs, `point_sw` und `point_ne`.

Verfügbarkeit: 3.4.0

Proj Version 6+

## Beispiele

Abruf der ersten 10 Metadatensätze aus der Proj-Datenbank.

```
SELECT auth_name, auth_srid, srsname FROM postgis_srs_all() LIMIT 10;
```

auth_name	auth_srid	srsname
EPSG	2000	Anguilla 1957 / British West Indies Grid
EPSG	20004	Pulkovo 1995 / Gauss-Kruger zone 4
EPSG	20005	Pulkovo 1995 / Gauss-Kruger zone 5
EPSG	20006	Pulkovo 1995 / Gauss-Kruger zone 6
EPSG	20007	Pulkovo 1995 / Gauss-Kruger zone 7
EPSG	20008	Pulkovo 1995 / Gauss-Kruger zone 8
EPSG	20009	Pulkovo 1995 / Gauss-Kruger zone 9
EPSG	2001	Antigua 1943 / British West Indies Grid
EPSG	20010	Pulkovo 1995 / Gauss-Kruger zone 10
EPSG	20011	Pulkovo 1995 / Gauss-Kruger zone 11

**Siehe auch**

[postgis\\_srs\\_codes](#), [postgis\\_srs](#), [postgis\\_srs\\_search](#)

**7.7.9 postgis\_srs\_search**

`postgis_srs_search` — Gibt Metadatenätze für projizierte Koordinatensysteme zurück, die Nutzungsbereiche haben, die den Parameter `bounds` vollständig enthalten.

**Synopsis**

```
setof record postgis_srs_search(geometry bounds, text auth_name=EPSG);
```

**Beschreibung**

Gibt eine Reihe von Metadatenätzen für projizierte Koordinatensysteme zurück, die Nutzungsbereiche haben, die den Parameter `bounds` vollständig enthalten. Jeder Datensatz enthält `auth_name`, `auth_srid`, `sname`, `srttext`, `proj4text`, und die Ecken des Nutzungsbereichs, `point_sw` und `point_ne`.

Die Suche sucht nur nach projizierten Koordinatensystemen und ist dafür gedacht, dass die Nutzer die möglichen Systeme erkunden, die für den Umfang ihrer Daten geeignet sind.

Verfügbarkeit: 3.4.0

Proj Version 6+

**Beispiele**

Suche nach projizierten Koordinatensystemen in Louisiana.

```
SELECT auth_name, auth_srid, sname,
       ST_AsText(point_sw) AS point_sw,
       ST_AsText(point_ne) AS point_ne
FROM postgis_srs_search('SRID=4326;LINESTRING(-90 30, -91 31)')
LIMIT 3;
```

auth_name	auth_srid	sname	point_sw	point_ne
EPSG	2801	NAD83(HARN) / Louisiana South	POINT(-93.94 28.85)	POINT(-88.75 31.07)
EPSG	3452	NAD83 / Louisiana South (ftUS)	POINT(-93.94 28.85)	POINT(-88.75 31.07)
EPSG	3457	NAD83(HARN) / Louisiana South (ftUS)	POINT(-93.94 28.85)	POINT(-88.75 31.07)

Durchsuchen Sie eine Tabelle nach maximaler Ausdehnung und finden Sie passende projizierte Koordinatensysteme.

```
WITH ext AS (
  SELECT ST_Extent(geom) AS geom, Max(ST_SRID(geom)) AS srid
  FROM foo
)
SELECT auth_name, auth_srid, sname,
       ST_AsText(point_sw) AS point_sw,
       ST_AsText(point_ne) AS point_ne
FROM ext
CROSS JOIN postgis_srs_search(ST_SetSRID(ext.geom, ext.srid))
LIMIT 3;
```

**Siehe auch**

[postgis\\_srs\\_codes](#), [postgis\\_srs\\_all](#), [postgis\\_srs](#)

## 7.8 Geometrische Konstruktoren

### 7.8.1 Well-known-Text (WKT) Repräsentation

#### 7.8.1.1 ST\_BdPolyFromText

`ST_BdPolyFromText` — Konstruiert ein Polygon aus einer beliebigen Ansammlung von geschlossenen Linienzügen, welche als `MultiLineString` in der Well-Known Text Darstellung vorliegen müssen.

**Synopsis**

```
geometry ST_BdPolyFromText(text WKT, integer srid);
```

**Beschreibung**

Konstruiert ein Polygon aus einer beliebigen Ansammlung von geschlossenen Linienzügen, welche als `MultiLineString` in der Well-Known Text Darstellung vorliegen müssen.

**Note**

Meldet einen Fehler, wenn es sich bei dem WKT nicht um einen `MULTILINESTRING` handelt. Meldet einen Fehler, wenn die Ausgabe ein `MULTIPOLYGON` ist; in diesem Fall verwenden Sie bitte `ST_BdMPolyFromText`, oder vergleichen `ST_BuildArea()` für einen PostGIS orientierten Ansatz.



Diese Methode implementiert die [OGC Simple Features Implementation Specification for SQL 1.1](#). s3.2.6.2

Wird vom GEOS Modul ausgeführt

Verfügbarkeit: 1.1.0

**Siehe auch**

[ST\\_BuildArea](#), [ST\\_BdMPolyFromText](#)

#### 7.8.1.2 ST\_BdMPolyFromText

`ST_BdMPolyFromText` — Konstruiert ein `MultiPolygon` aus einer beliebigen Ansammlung von geschlossenen Linienzügen, welche als `MultiLineString` in der Well-Known Text Darstellung vorliegen müssen.

**Synopsis**

```
geometry ST_BdMPolyFromText(text WKT, integer srid);
```

## Beschreibung

Konstruiert ein Polygon aus einer beliebigen Ansammlung von geschlossenen Linienzügen, Polygonen und MultiLineStrings, welche in der Well-Known Text Darstellung vorliegen müssen.



### 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.



Diese Methode implementiert die [OGC Simple Features Implementation Specification for SQL 1.1](#). s3.2.6.2

Wird vom GEOS Modul ausgeführt

Verfügbarkeit: 1.1.0

## Siehe auch

[ST\\_BuildArea](#), [ST\\_BdPolyFromText](#)

### 7.8.1.3 ST\_GeogFromText

`ST_GeogFromText` — Gibt einen geographischen Datentyp aus einer Well-known-Text (WKT), oder einer erweiterten WKT (EWKT), Darstellung zurück.

## Synopsis

```
geography ST_GeogFromText(text EWKT);
```

## Beschreibung

Gibt ein geographisches Objekt in der Well-known-Text oder in der erweiterten Well-known-Text Darstellung zurück. Falls nicht angegeben, wird die SRID 4326 angenommen. Dies ist ein Alias für `ST_GeographyFromText`. Punkte werden immer in Form von Länge und Breite ausgedrückt.

## Beispiele

```
--- converting lon lat coords to geography
ALTER TABLE sometable ADD COLUMN geog geography(POINT,4326);
UPDATE sometable SET geog = ST_GeogFromText('SRID=4326;POINT(' || lon || ' ' || lat || ')') ←
;

--- specify a geography point using EPSG:4267, NAD27
SELECT ST_AsEWKT(ST_GeogFromText('SRID=4267;POINT(-77.0092 38.889588)'));
```

## Siehe auch

[ST\\_AsText](#), [ST\\_GeographyFromText](#)

### 7.8.1.4 ST\_GeographyFromText

`ST_GeographyFromText` — Gibt einen geographischen Datentyp aus einer Well-known-Text (WKT), oder einer erweiterten WKT (EWKT), Darstellung zurück.



## Synopsis

geography **ST\_GeographyFromText**(text EWKT);

## Beschreibung

Gibt ein geographisches Objekt in der Well-known-Text Darstellung zurück. Falls nicht angegeben, wird die SRID 4326 angenommen.

## Siehe auch

[ST\\_GeogFromText](#), [ST\\_AsText](#)

### 7.8.1.5 ST\_GeomCollFromText

**ST\_GeomCollFromText** — Erzeugt eine Sammelgeometrie mit der gegebenen SRID aus einer WKT-Kollektion. Wenn keine SRID angegeben ist, wird diese standardmäßig auf 0 gesetzt.

## Synopsis

geometry **ST\_GeomCollFromText**(text WKT, integer srid);  
geometry **ST\_GeomCollFromText**(text WKT);

## Beschreibung

Erzeugt eine Sammelgeometrie mit der gegebenen SRID aus einer Well-known-Text (WKT) Darstellung. Wenn keine SRID angegeben ist, wird diese standardmäßig auf 0 gesetzt.

OGC SPEC 3.2.6.2 - Die Option SRID stammt aus dem Konformitätstest

Gibt NULL zurück, wenn der WKT keine GEOMETRYCOLLECTION ist



### Note

Verwenden Sie diese Funktion nicht, wenn Sie sich vollkommen sicher sind, dass ihre WKT Geometrie eine Sammelgeometrie ist. Sie ist langsamer als **ST\_GeomFromText**, da sie einen zusätzlichen Validierungsschritt ausführt.



Diese Methode implementiert die [OGC Simple Features Implementation Specification for SQL 1.1](#). s3.2.6.2



Diese Methode setzt die SQL/MM-Spezifikation um.

## Beispiele

```
SELECT ST_GeomCollFromText('GEOMETRYCOLLECTION(POINT(1 2),LINESTRING(1 2, 3 4))');
```

## Siehe auch

[ST\\_GeomFromText](#), [ST\\_SRID](#)

### 7.8.1.6 ST\_GeomFromEWKT

**ST\_GeomFromEWKT** — Gibt einen spezifizierten ST\_Geometry-Wert von einer erweiterten Well-known-Text Darstellung (EWKT) zurück.

## Synopsis

geometry **ST\_GeomFromEWKT**(text EWKT);

## Beschreibung

Erzeugt ein PostGIS ST\_Geometry Objekt aus der erweiterten OGC Well-known-Text (EWKT) Darstellung.



### Note

EWKT ist kein Format des OGC Standards, sondern ein PostGIS eigenes Format, welches den Identifikator (SRID) des räumlichen Koordinatenreferenzsystem mit einbindet

Erweiterung: Mit 2.0.0 wurde die Unterstützung für polyedrische Oberflächen und TIN eingeführt.



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



Diese Methode unterstützt kreisförmige Strings und Kurven.



Diese Funktion unterstützt polyedrische Flächen.



Diese Funktion unterstützt Dreiecke und dreieckige unregelmäßige Netzoberflächen (TIN).

## Beispiele

```
SELECT ST_GeomFromEWKT('SRID=4269;LINESTRING(-71.160281 42.258729,-71.160837 ↵
  42.259113,-71.161144 42.25932)');
SELECT ST_GeomFromEWKT('SRID=4269;MULTILINESTRING((-71.160281 42.258729,-71.160837 ↵
  42.259113,-71.161144 42.25932)');

SELECT ST_GeomFromEWKT('SRID=4269;POINT(-71.064544 42.28787)');

SELECT ST_GeomFromEWKT('SRID=4269;POLYGON((-71.1776585052917 ↵
  42.3902909739571,-71.1776820268866 42.3903701743239,
-71.1776063012595 42.3903825660754,-71.1775826583081 42.3903033653531,-71.1776585052917 ↵
  42.3902909739571)');

SELECT ST_GeomFromEWKT('SRID=4269;MULTIPOLYGON((( -71.1031880899493 42.3152774590236,
-71.1031627617667 42.3152960829043,-71.102923838298 42.3149156848307,
-71.1023097974109 42.3151969047397,-71.1019285062273 42.3147384934248,
-71.102505233663 42.3144722937587,-71.10277487471 42.3141658254797,
-71.103113945163 42.3142739188902,-71.10324876416 42.31402489987,
-71.1033002961013 42.3140393340215,-71.1033488797549 42.3139495090772,
-71.103396240451 42.3138632439557,-71.1041521907712 42.3141153348029,
-71.1041411411543 42.3141545014533,-71.1041287795912 42.3142114839058,
-71.1041188134329 42.3142693656241,-71.1041112482575 42.3143272556118,
-71.1041072845732 42.3143851580048,-71.1041057218871 42.3144430686681,
-71.1041065602059 42.3145009876017,-71.1041097995362 42.3145589148055,
-71.1041166403905 42.3146168544148,-71.1041258822717 42.3146748022936,
-71.1041375307579 42.3147318674446,-71.1041492906949 42.3147711126569,
-71.1041598612795 42.314808571739,-71.1042515013869 42.3151287620809,
-71.1041173835118 42.3150739481917,-71.1040809891419 42.3151344119048,
-71.1040438678912 42.3151191367447,-71.1040194562988 42.3151832057859,
-71.1038734225584 42.3151140942995,-71.1038446938243 42.3151006300338,
-71.1038315271889 42.315094347535,-71.1037393329282 42.315054824985,
-71.1035447555574 42.3152608696313,-71.1033436658644 42.3151648370544,
-71.1032580383161 42.3152269126061,-71.103223066939 42.3152517403219,
```

```
-71.1031880899493 42.3152774590236)),
((-71.1043632495873 42.315113108546,-71.1043583974082 42.3151211109857,
-71.1043443253471 42.3150676015829,-71.1043850704575 42.3150793250568,-71.1043632495873 ←
 42.315113108546)))');
```

```
--3d circular string
SELECT ST_GeomFromEWKT('CIRCULARSTRING(220268 150415 1,220227 150505 2,220227 150406 3)');
```

```
--Polyhedral Surface example
SELECT ST_GeomFromEWKT('POLYHEDRALSURFACE (
  ((0 0 0, 0 0 1, 0 1 1, 0 1 0, 0 0 0)),
  ((0 0 0, 0 1 0, 1 1 0, 1 0 0, 0 0 0)),
  ((0 0 0, 1 0 0, 1 0 1, 0 0 1, 0 0 0)),
  ((1 1 0, 1 1 1, 1 0 1, 1 0 0, 1 1 0)),
  ((0 1 0, 0 1 1, 1 1 1, 1 1 0, 0 1 0)),
  ((0 0 1, 1 0 1, 1 1 1, 0 1 1, 0 0 1))
)');
```

## Siehe auch

[ST\\_AsEWKT](#), [ST\\_GeomFromText](#)

### 7.8.1.7 ST\_GeomFromMARC21

`ST_GeomFromMARC21` — Nimmt MARC21/XML-Geodaten als Eingabe und gibt ein PostGIS-Geometrieobjekt zurück.

## Synopsis

geometry `ST_GeomFromMARC21` ( text marxml );

## Beschreibung

Diese Funktion erstellt eine PostGIS-Geometrie aus einem MARC21/XML-Datensatz, der einen POINT oder ein POLYGON enthalten kann. Bei mehreren geografischen Dateneinträgen im selben MARC21/XML-Datensatz wird ein MULTIPOINT oder MULTIPOLYGON zurückgegeben. Wenn der Datensatz gemischte Geometrietypen enthält, wird GEOMETRYCOLLECTION zurückgegeben. Es wird NULL zurückgegeben, wenn der MARC21/XML-Datensatz keine geografischen Daten enthält (Datenfeld:034).

LOC MARC21/XML-Versionen werden unterstützt:

- [MARC21/XML 1.1](#)

Verfügbarkeit: 3.3.0, erfordert libxml2 2.6+



#### Note

MARC21/XML Coded Cartographic Mathematical Data bietet derzeit keine Möglichkeit, das räumliche Bezugssystem der kodierten Koordinaten zu beschreiben, so dass diese Funktion immer eine Geometrie mit SRID 0 zurückgibt.



#### Note

Die zurückgegebenen POLYGON Geometrien sind immer im Uhrzeigersinn ausgerichtet.

**Beispiele**

Konvertierung von MARC21/XML-Geodaten, die einen einzelnen POINT enthalten, kodiert als hddd . dddddd

```

SELECT
  ST_AsText (
    ST_GeomFromMARC21 ('
      <record xmlns="http://www.loc.gov/MARC21/slim">
        <leader
>00000nz a2200000nc 4500</leader>
        <controlfield tag="001"
>040277569</controlfield>
        <datafield tag="034" ind1=" " ind2=" ">
          <subfield code="d"
>W004.50000</subfield>
          <subfield code="e"
>W004.50000</subfield>
          <subfield code="f"
>N054.25000</subfield>
          <subfield code="g"
>N054.25000</subfield>
        </datafield>
      </record
>'));

      st_astext
      -----
      POINT(-4.5 54.25)
      (1 row)

```

Konvertierung von MARC21/XML-Geodaten, die eine einzelne POLYGON kodiert als hdddmms

```

SELECT
  ST_AsText (
    ST_GeomFromMARC21 ('
      <record xmlns="http://www.loc.gov/MARC21/slim">
        <leader
>01062cem a2200241 a 4500</leader>
        <controlfield tag="001"
> 84696781 </controlfield>
        <datafield tag="034" ind1="1" ind2=" ">
          <subfield code="a"
>a</subfield>
          <subfield code="b"
>50000</subfield>
          <subfield code="d"
>E0130600</subfield>
          <subfield code="e"
>E0133100</subfield>
          <subfield code="f"
>N0523900</subfield>
          <subfield code="g"
>N0522300</subfield>
        </datafield>
      </record
>'));

      st_astext

```

```

-----
POLYGON((13.1 52.65,13.516666666666667 52.65,13.516666666666667 ←
        52.38333333333333,13.1 52.38333333333333,13.1 52.65))
(1 row)

```

### Konvertierung von MARC21/XML-Geodaten, die ein POLYGON und ein POINT enthalten:

```

SELECT
ST_AsText (
  ST_GeomFromMARC21 ('
<record xmlns="http://www.loc.gov/MARC21/slim">
  <datafield tag="034" ind1="1" ind2=" ">
    <subfield code="a"
>a</subfield>
    <subfield code="b"
>50000</subfield>
    <subfield code="d"
>E0130600</subfield>
    <subfield code="e"
>E0133100</subfield>
    <subfield code="f"
>N0523900</subfield>
    <subfield code="g"
>N0522300</subfield>
  </datafield>
  <datafield tag="034" ind1=" " ind2=" ">
    <subfield code="d"
>W004.500000</subfield>
    <subfield code="e"
>W004.500000</subfield>
    <subfield code="f"
>N054.250000</subfield>
    <subfield code="g"
>N054.250000</subfield>
  </datafield>
</record
>'));

```

st\_astext ↔

```

-----
GEOMETRYCOLLECTION(POLYGON((13.1 52.65,13.516666666666667 ←
        52.65,13.516666666666667 52.38333333333333,13.1 52.38333333333333,13.1 ←
        52.65)),POINT(-4.5 54.25))
(1 row)

```

#### Siehe auch

[ST\\_AsMARC21](#)

#### 7.8.1.8 ST\_GeometryFromText

**ST\_GeometryFromText** — Gibt einen spezifizierten ST\_Geometry-Wert von einer Well-known-Text Darstellung (WKT) zurück. Die Bezeichnung ist ein Alias für ST\_GeomFromText

## Synopsis

```
geometry ST_GeometryFromText(text WKT);
geometry ST_GeometryFromText(text WKT, integer srid);
```

## Beschreibung



Diese Methode implementiert die [OGC Simple Features Implementation Specification for SQL 1.1](#).



Diese Methode setzt die SQL/MM-Spezifikation um. SQL-MM 3: 5.1.40

## Siehe auch

[ST\\_GeomFromText](#)

### 7.8.1.9 ST\_GeomFromText

ST\_GeomFromText — Gibt einen spezifizierten ST\_Geometry Wert aus einer Well-known-Text Darstellung (WKT) zurück.

## Synopsis

```
geometry ST_GeomFromText(text WKT);
geometry ST_GeomFromText(text WKT, integer srid);
```

## Beschreibung

Erzeugt ein PostGIS ST\_Geometry Objekt aus der OGC Well-known-Text Darstellung.



### 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.



Diese Methode implementiert die [OGC Simple Features Implementation Specification for SQL 1.1](#). s3.2.6.2 - die Option SRID ist vom Konformitätstest.



Diese Methode setzt die SQL/MM-Spezifikation um. SQL-MM 3: 5.1.40



Diese Methode unterstützt kreisförmige Strings und Kurven.



### Note

Obwohl nicht OGC-konform, ist [ST\\_MakePoint](#) schneller als ST\_GeomFromText und ST\_PointFromText. Es ist auch einfacher für numerische Koordinatenwerte zu verwenden. [ST\\_Point](#) ist eine weitere Option, die ähnlich schnell wie [ST\\_MakePoint](#) ist und OGC-konform ist, aber nur 2D-Punkte unterstützt.



### Warning

Änderung: 2.0.0 - In Vorgängerversionen von PostGIS war ST\_GeomFromText('GEOMETRYCOLLECTION(EMPTY)') erlaubt. Um eine bessere Übereinstimmung mit der SQL/MM Norm zu erreichen, ist dies in PostGIS 2.0.0 nun nicht mehr gestattet. Hier sollte nun ST\_GeomFromText('GEOMETRYCOLLECTION EMPTY') geschrieben werden.

## Beispiele

```

SELECT ST_GeomFromText('LINESTRING(-71.160281 42.258729,-71.160837 42.259113,-71.161144 42.25932)');
SELECT ST_GeomFromText('LINESTRING(-71.160281 42.258729,-71.160837 42.259113,-71.161144 42.25932)',4269);

SELECT ST_GeomFromText('MULTILINESTRING((-71.160281 42.258729,-71.160837 42.259113,-71.161144 42.25932))');

SELECT ST_GeomFromText('POINT(-71.064544 42.28787)');

SELECT ST_GeomFromText('POLYGON((-71.1776585052917 42.3902909739571,-71.1776820268866 42.3903701743239,-71.1776063012595 42.3903825660754,-71.1775826583081 42.3903033653531,-71.1776585052917 42.3902909739571))');

SELECT ST_GeomFromText('MULTIPOLYGON((( -71.1031880899493 42.3152774590236,-71.1031627617667 42.3152960829043,-71.102923838298 42.3149156848307,-71.1023097974109 42.3151969047397,-71.1019285062273 42.3147384934248,-71.102505233663 42.3144722937587,-71.10277487471 42.3141658254797,-71.103113945163 42.3142739188902,-71.10324876416 42.31402489987,-71.1033002961013 42.3140393340215,-71.1033488797549 42.3139495090772,-71.103396240451 42.3138632439557,-71.1041521907712 42.3141153348029,-71.1041411411543 42.3141545014533,-71.1041287795912 42.3142114839058,-71.1041188134329 42.3142693656241,-71.1041112482575 42.3143272556118,-71.1041072845732 42.3143851580048,-71.1041057218871 42.3144430686681,-71.1041065602059 42.3145009876017,-71.1041097995362 42.3145589148055,-71.1041166403905 42.3146168544148,-71.1041258822717 42.3146748022936,-71.1041375307579 42.3147318674446,-71.1041492906949 42.3147711126569,-71.1041598612795 42.314808571739,-71.1042515013869 42.3151287620809,-71.1041173835118 42.3150739481917,-71.1040809891419 42.3151344119048,-71.1040438678912 42.3151191367447,-71.1040194562988 42.3151832057859,-71.1038734225584 42.3151140942995,-71.1038446938243 42.3151006300338,-71.1038315271889 42.315094347535,-71.1037393329282 42.315054824985,-71.1035447555574 42.3152608696313,-71.1033436658644 42.3151648370544,-71.1032580383161 42.3152269126061,-71.103223066939 42.3152517403219,-71.1031880899493 42.3152774590236)),((-71.1043632495873 42.315113108546,-71.1043583974082 42.3151211109857,-71.1043443253471 42.3150676015829,-71.1043850704575 42.3150793250568,-71.1043632495873 42.315113108546))',4326);

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

```

## Siehe auch

[ST\\_GeomFromEWKT](#), [ST\\_GeomFromWKB](#), [ST\\_SRID](#)

### 7.8.1.10 ST\_LineFromText

**ST\_LineFromText** — Erzeugt eine Geometrie aus einer WKT Darstellung mit der angegebenen SRID. Wenn keine SRID angegeben wird, wird diese standardmäßig auf 0 gesetzt.

## Synopsis

```

geometry ST_LineFromText(text WKT);
geometry ST_LineFromText(text WKT, integer srid);

```

## Beschreibung

Erzeugt eine Geometrie aus einer WKT Darstellung mit der angegebenen SRID. Wenn keine SRID angegeben wird, wird diese standardmäßig auf 0 gesetzt. Wenn das übergebene WKT kein LineString ist, wird NULL zurückgegeben.



### Note

OGC SPEC 3.2.6.2 - die Option SRID ist vom Konformitätstest.



### Note

Wenn Sie wissen, dass die Geometrie nur aus LINESTRINGs besteht, ist es effizienter einfach ST\_GeomFromText zu verwenden. Diese Funktion ruft auch nur ST\_GeomFromText auf und fügt die Information hinzu, dass es sich um einen Linienzug handelt.



Diese Methode implementiert die [OGC Simple Features Implementation Specification for SQL 1.1](#). s3.2.6.2



Diese Methode setzt die SQL/MM-Spezifikation um. SQL-MM 3: 7.2.8

## Beispiele

```
SELECT ST_LineFromText('LINESTRING(1 2, 3 4)') AS aline, ST_LineFromText('POINT(1 2)') AS ↵
      null_return;
aline          | null_return
-----
01020000000200000000000000000000F ... | t
```

## Siehe auch

[ST\\_GeomFromText](#)

### 7.8.1.11 ST\_MLineFromText

ST\_MLineFromText — Liest einen festgelegten ST\_MultiLineString Wert von einer WKT-Darstellung aus.

## Synopsis

```
geometry ST_MLineFromText(text WKT, integer srid);
geometry ST_MLineFromText(text WKT);
```

## Beschreibung

Erzeugt eine Geometrie aus einer Well-known-Text (WKT) Darstellung mit der angegebenen SRID. Wenn keine SRID angegeben wird, wird diese standardmäßig auf 0 gesetzt.

OGC SPEC 3.2.6.2 - Die Option SRID stammt aus dem Konformitätstest

Gibt NULL zurück wenn der WKT kein MULTILINESTRING ist.



### Note

Verwenden Sie diese Funktion nicht, wenn Sie sich vollkommen sicher sind, dass ihre WKT Geometrie nur aus Punkten besteht. Sie ist langsamer als ST\_GeomFromText, da sie einen zusätzlichen Validierungsschritt hinzufügt.



- ✔ Diese Methode implementiert die [OGC Simple Features Implementation Specification for SQL 1.1](#). s3.2.6.2
- ✔ Diese Methode setzt die SQL/MM-Spezifikation um. SQL-MM 3: 9.4.4

### Beispiele

```
SELECT ST_MLineFromText('MULTILINESTRING((1 2, 3 4), (4 5, 6 7))');
```

### Siehe auch

[ST\\_GeomFromText](#)

#### 7.8.1.12 ST\_MPointFromText

`ST_MPointFromText` — Erzeugt eine Geometrie aus WKT mit der angegebenen SRID. Wenn keine SRID angegeben wird, wird diese standardmäßig auf 0 gesetzt.

### Synopsis

```
geometry ST_MPointFromText(text WKT, integer srid);  
geometry ST_MPointFromText(text WKT);
```

### Beschreibung

Erzeugt eine Geometrie aus WKT mit der angegebenen SRID. Wenn SRID nicht angegeben ist, wird sie standardmäßig auf 0 gesetzt.

OGC SPEC 3.2.6.2 - Die Option SRID stammt aus dem Konformitätstest

Gibt NULL zurück, wenn der WKT kein MULTIPOINT ist.



#### Note

Verwenden Sie diese Funktion nicht, wenn Sie sich vollkommen sicher sind, dass ihre WKT Geometrie nur aus Punkten besteht. Sie ist langsamer als `ST_GeomFromText`, da sie einen zusätzlichen Validierungsschritt hinzufügt.

- ✔ Diese Methode implementiert die [OGC Simple Features Implementation Specification for SQL 1.1](#). 3.2.6.2
- ✔ Diese Methode setzt die SQL/MM-Spezifikation um. SQL-MM 3: 9.2.4

### Beispiele

```
SELECT ST_MPointFromText('MULTIPOINT((1 2), (3 4))');  
SELECT ST_MPointFromText('MULTIPOINT((-70.9590 42.1180), (-70.9611 42.1223))', 4326);
```

### Siehe auch

[ST\\_GeomFromText](#)

### 7.8.1.13 ST\_MPolyFromText

ST\_MPolyFromText — Erzeugt eine MultiPolygon Geometrie aus WKT mit der angegebenen SRID. Wenn SRID nicht angegeben ist, wird sie standardmäßig auf 0 gesetzt.

#### Synopsis

```
geometry ST_MPolyFromText(text WKT, integer srid);
geometry ST_MPolyFromText(text WKT);
```

#### Beschreibung

Erzeugt ein MultiPolygon von WKT mit der gegebenen SRID. Wenn SRID nicht angegeben ist, wird sie standardmäßig auf 0 gesetzt.

OGC SPEC 3.2.6.2 - Die Option SRID stammt aus dem Konformitätstest

Meldet einen Fehler, wenn der WKT kein MULTIPOLYGON ist.



#### Note

Verwenden Sie diese Funktion nicht, wenn Sie sich vollkommen sicher sind, dass ihre WKT Geometrie nur aus Multi-Polygonen besteht. Sie ist langsamer als ST\_GeomFromText, da sie einen zusätzlichen Validierungsschritt hinzufügt.



Diese Methode implementiert die [OGC Simple Features Implementation Specification for SQL 1.1](#). s3.2.6.2



Diese Methode setzt die SQL/MM-Spezifikation um. SQL-MM 3: 9.6.4

#### Beispiele

```
SELECT ST_MPolyFromText('MULTIPOLYGON(((0 0 1,20 0 1,20 20 1,0 20 1,0 0 1),(5 5 3,5 7 3,7 7 ←
3,7 5 3,5 5 3)))');
SELECT ST_MPolyFromText('MULTIPOLYGON(((−70.916 42.1002,−70.9468 42.0946,−70.9765 ←
42.0872,−70.9754 42.0875,−70.9749 42.0879,−70.9752 42.0881,−70.9754 42.0891,−70.9758 ←
42.0894,−70.9759 42.0897,−70.9759 42.0899,−70.9754 42.0902,−70.9756 42.0906,−70.9753 ←
42.0907,−70.9753 42.0917,−70.9757 42.0924,−70.9755 42.0928,−70.9755 42.0942,−70.9751 ←
42.0948,−70.9755 42.0953,−70.9751 42.0958,−70.9751 42.0962,−70.9759 42.0983,−70.9767 ←
42.0987,−70.9768 42.0991,−70.9771 42.0997,−70.9771 42.1003,−70.9768 42.1005,−70.977 ←
42.1011,−70.9766 42.1019,−70.9768 42.1026,−70.9769 42.1033,−70.9775 42.1042,−70.9773 ←
42.1043,−70.9776 42.1043,−70.9778 42.1048,−70.9773 42.1058,−70.9774 42.1061,−70.9779 ←
42.1065,−70.9782 42.1078,−70.9788 42.1085,−70.9798 42.1087,−70.9806 42.109,−70.9807 ←
42.1093,−70.9806 42.1099,−70.9809 42.1109,−70.9808 42.1112,−70.9798 42.1116,−70.9792 ←
42.1127,−70.979 42.1129,−70.9787 42.1134,−70.979 42.1139,−70.9791 42.1141,−70.9987 ←
42.1116,−71.0022 42.1273,
−70.9408 42.1513,−70.9315 42.1165,−70.916 42.1002)))',4326);
```

#### Siehe auch

[ST\\_GeomFromText](#), [ST\\_SRID](#)

### 7.8.1.14 ST\_PointFromText

ST\_PointFromText — Erzeugt eine Punktgeometrie mit gegebener SRID von WKT. Wenn SRID nicht angegeben ist, wird sie standardmäßig auf 0 gesetzt.

## Synopsis

```
geometry ST_PointFromText(text WKT);  
geometry ST_PointFromText(text WKT, integer srid);
```

## Beschreibung

Erzeugt ein PostGIS ST\_Geometrie Punktobjekt von der OGC Well-known-Text Darstellung. Wenn die SRID nicht angegeben ist, wird sie standardmäßig auf "unknown" (zurzeit 0) gesetzt. Falls die Geometrie nicht in der WKT Punktdarstellung vorliegt, wird NULL zurückgegeben. Bei einer invaliden WKT Darstellung wird eine Fehlermeldung angezeigt.



### Note

Die Funktion `ST_PointFromText` hat zwei Varianten. Die erste Variante nimmt keine SRID entgegen und gibt eine Geometrie ohne ein bestimmtes Koordinatenreferenzsystem aus. Die zweite Variante nimmt eine SRID als zweiten Übergabewert entgegen und gibt eine Geometrie zurück, die diese SRID als Teil ihrer Metadaten beinhaltet. Die SRID muss in der Tabelle "spatial\_ref\_sys" definiert sein.



### Note

Verwenden Sie diese Funktion nicht, wenn Sie sich vollkommen sicher sind, dass ihre WKT Geometrie nur aus Punkten besteht. Sie ist langsamer als `ST_GeomFromText`, da sie einen zusätzlichen Validierungsschritt hinzufügt. Wenn Sie Punkte aus Koordinaten in Länge und Breite erstellen und mehr auf Rechenleistung und Genauigkeit wertlegen als auf OGC-Konformität, so verwenden Sie bitte `ST_MakePoint` oder den OGC-konformen Alias `ST_Point`.



Diese Methode implementiert die [OGC Simple Features Implementation Specification for SQL 1.1](#). s3.2.6.2 - die Option SRID ist vom Konformitätstest.



Diese Methode setzt die SQL/MM-Spezifikation um. SQL-MM 3: 6.1.8

## Beispiele

```
SELECT ST_PointFromText ('POINT(-71.064544 42.28787) ');  
SELECT ST_PointFromText ('POINT(-71.064544 42.28787)', 4326);
```

## Siehe auch

[ST\\_GeomFromText](#), [ST\\_MakePoint](#), [ST\\_Point](#), [ST\\_SRID](#)

### 7.8.1.15 ST\_PolygonFromText

`ST_PolygonFromText` — Erzeugt eine Geometrie aus WKT mit der angegebenen SRID. Wenn keine SRID angegeben wird, wird diese standardmäßig auf 0 gesetzt.

## Synopsis

```
geometry ST_PolygonFromText(text WKT);  
geometry ST_PolygonFromText(text WKT, integer srid);
```

## Beschreibung

Erzeugt eine Geometrie mit gegebener SRID von WKT. Wenn SRID nicht angegeben ist, wird sie standardmäßig auf 0 gesetzt. Gibt NULL zurück, wenn WKT kein Polygon ist.

OGC SPEC 3.2.6.2 - Die Option SRID stammt aus dem Konformitätstest



### Note

Verwenden Sie diese Funktion nicht, wenn Sie sich vollkommen sicher sind, dass ihre WKT Geometrie nur aus Polygonen besteht. Sie ist langsamer als ST\_GeomFromText, da sie einen zusätzlichen Validierungsschritt hinzufügt.



Diese Methode implementiert die [OGC Simple Features Implementation Specification for SQL 1.1](#). s3.2.6.2



Diese Methode setzt die SQL/MM-Spezifikation um. SQL-MM 3: 8.3.6

## Beispiele

```
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
-----
010300000001000000050000006...
```

```
SELECT ST_PolygonFromText('POINT(1 2)') IS NULL as point_is_notpoly;
point_is_not_poly
-----
t
```

## Siehe auch

[ST\\_GeomFromText](#)

### 7.8.1.16 ST\_WKTTToSQL

ST\_WKTTToSQL — 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\_WKTTToSQL**(text WKT);

## Beschreibung



Diese Methode setzt die SQL/MM-Spezifikation um. SQL-MM 3: 5.1.34

## Siehe auch

[ST\\_GeomFromText](#)

## 7.8.2 Well-known-Binary (WKB) Repräsentation

### 7.8.2.1 ST\_GeogFromWKB

`ST_GeogFromWKB` — Erzeugt ein geographisches Objekt aus der Well-known-Binary (WKB) oder der erweiterten Well-known-Binary (EWKB) Darstellung.

#### Synopsis

```
geography ST_GeogFromWKB(bytea wkb);
```

#### Beschreibung

Die Funktion `ST_GeogFromWKB` empfängt eine Well-known-Binary (WKB) oder eine erweiterte PostGIS WKB (EWKB) Darstellung einer Geometrie und erzeugt eine Instanz des entsprechenden geographischen Datentyps. Diese Funktion übernimmt die Rolle der Geometrie-Factory/Fabrik in SQL.

Wenn die SRID nicht festgelegt ist, wird 4326 (WGS 84) angenommen.



Diese Methode unterstützt kreisförmige Strings und Kurven.

#### Beispiele

```
--Although bytea rep contains single \, these need to be escaped when inserting into a ↵
  table
SELECT ST_AsText (
ST_GeogFromWKB(E'\001\002\000\000\000\002\000\000\000\0037\205\353Q ↵
  \270~\300\323Mb\020X\231C@\020X9\264\310~\300)\217\302\365\230 ↵
  C@')
);
                                st_astext
-----
LINESTRING(-113.98 39.198,-113.981 39.195)
(1 row)
```

#### Siehe auch

[ST\\_GeogFromText](#), [ST\\_AsBinary](#)

### 7.8.2.2 ST\_GeomFromEWKB

`ST_GeomFromEWKB` — Gibt einen geometrischen Datentyp (`ST_Geometry`) aus einer Well-known-Binary (WKB) Darstellung zurück.

#### Synopsis

```
geometry ST_GeomFromEWKB(bytea EWKB);
```

## Beschreibung

Erzeugt ein PostGIS ST\_Geometry Objekt aus der erweiterten OGC Well-known-Text (EWKT) Darstellung.



### Note

EWKB ist kein Format des OGC Standards, sondern ein PostGIS eigenes Format, welches den Identifikator (SRID) des räumlichen Koordinatenreferenzsystem mit einbindet

Erweiterung: Mit 2.0.0 wurde die Unterstützung für polyedrische Oberflächen und TIN eingeführt.



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



Diese Methode unterstützt kreisförmige Strings und Kurven.



Diese Funktion unterstützt polyedrische Flächen.



Diese Funktion unterstützt Dreiecke und dreieckige unregelmäßige Netzoberflächen (TIN).

## Beispiele

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](#)

### 7.8.2.3 ST\_GeomFromWKB

**ST\_GeomFromWKB** — Erzeugt ein geometrisches Objekt aus der Well-known-Binary (WKB) Darstellung und einer optionalen SRID.

#### Synopsis

```
geometry ST_GeomFromWKB(bytea geom);
geometry ST_GeomFromWKB(bytea geom, integer srid);
```

#### Beschreibung

Die Funktion **ST\_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.



Diese Methode implementiert die **OGC Simple Features Implementation Specification for SQL 1.1**. s3.2.7.2 - die optionale SRID kommt vom Konformitätstest.



Diese Methode setzt die SQL/MM-Spezifikation um. SQL-MM 3: 5.1.41



Diese Methode unterstützt kreisförmige Strings und Kurven.

#### Beispiele

```
--Although bytea rep contains single \, these need to be escaped when inserting into a table
-- unless standard_conforming_strings is set to on.
SELECT ST_AsEWKT(
ST_GeomFromWKB(E'\001\002\000\000\000\000\002\000\000\000\0037\205\353Q
\270~\300\323Mb\020X\231C@020X9\264\310~\300)\217\302\365\230
C@',4326)
);
          st_asewkt
-----
SRID=4326;LINESTRING(-113.98 39.198,-113.981 39.195)
(1 row)

SELECT
  ST_AsText(
    ST_GeomFromWKB(
      ST_AsEWKB('POINT(2 5)::geometry')
    )
  );
  st_astext
-----
POINT(2 5)
(1 row)
```

#### Siehe auch

**ST\_WKBToSQL**, **ST\_AsBinary**, **ST\_GeomFromEWKB**

### 7.8.2.4 ST\_LineFromWKB

ST\_LineFromWKB — Erzeugt einen LINESTRING mit gegebener SRID aus einer WKB-Darstellung

#### Synopsis

```
geometry ST_LineFromWKB(bytea WKB);
geometry ST_LineFromWKB(bytea WKB, integer srid);
```

#### Beschreibung

Die Funktion ST\_GeogFromWKB nimmt eine Well-known-Binary Darstellung der Geometrie und eine Id für das Koordinatenreferenzsystem (SRID) entgegen und erzeugt eine Instanz des entsprechenden geometrischen Datentyps - in diesem Fall eine Geometrie vom Typ LineString. Diese Funktion übernimmt die Rolle der Geometrie-Factory in SQL.

Wenn keine SRID angegeben ist, wird diese auf 0 gesetzt. NULL wird zurückgegeben, wenn die Eingabe bytea keinen LINESTRING darstellt.



#### Note

OGC SPEC 3.2.6.2 - die Option SRID ist vom Konformitätstest.



#### Note

Wenn Sie wissen, dass Ihre Geometrie nur aus LINESTRINGs besteht, ist es effizienter einfach ST\_GeomFromWKB zu verwenden. Diese Funktion ruft auch nur ST\_GeomFromWKB auf und fügt die Information hinzu, dass es sich um einen Linienzug handelt.



Diese Methode implementiert die [OGC Simple Features Implementation Specification for SQL 1.1](#). s3.2.6.2



Diese Methode setzt die SQL/MM-Spezifikation um. SQL-MM 3: 7.2.9

#### Beispiele

```
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
-----|-----
01020000000200000000000000000000F ... | t
```

#### Siehe auch

[ST\\_GeomFromWKB](#), [ST\\_LinestringFromWKB](#)

### 7.8.2.5 ST\_LinestringFromWKB

ST\_LinestringFromWKB — Erzeugt eine Geometrie mit gegebener SRID aus einer WKB-Darstellung.

#### Synopsis

```
geometry ST_LinestringFromWKB(bytea WKB);
geometry ST_LinestringFromWKB(bytea WKB, integer srid);
```



## Beschreibung

Die Funktion `ST_LineStringFromWKB` nimmt eine Well-known-Binary Darstellung der Geometrie und eine Id für das Koordinatenreferenzsystem (SRID) entgegen und erzeugt eine Instanz des entsprechenden geometrischen Datentyps - in diesem Fall eine Geometrie vom Typ `LineString`. Diese Funktion übernimmt die Rolle der Geometrie-Factory in SQL.

Wenn keine SRID angegeben ist, wird diese standardmäßig auf 0 gesetzt. `NULL` wird zurückgegeben, wenn die Eingabe `bytea` keinen `LINestring` darstellt. Ist ein Alias für `ST_LineFromWKB`.



### Note

OGC SPEC 3.2.6.2 - optionale SRID ist vom Konformitätstest.



### Note

Wenn Sie wissen, dass Ihre Geometrie nur aus `LINestring`s besteht, ist es effizienter einfach `ST_GeomFromWKB` zu verwenden. Diese Funktion ruft auch nur `ST_GeomFromWKB` auf und fügt die Information hinzu, dass es sich um einen `LINestring` handelt.



Diese Methode implementiert die [OGC Simple Features Implementation Specification for SQL 1.1](#). s3.2.6.2



Diese Methode setzt die SQL/MM-Spezifikation um. SQL-MM 3: 7.2.9

## Beispiele

```
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;
-----
01020000000200000000000000000000F ... | t
```

## Siehe auch

[ST\\_GeomFromWKB](#), [ST\\_LineFromWKB](#)

### 7.8.2.6 ST\_PointFromWKB

`ST_PointFromWKB` — Erzeugt eine Geometrie mit gegebener SRID von WKB.

## Synopsis

```
geometry ST_GeomFromWKB(bytea geom);
geometry ST_GeomFromWKB(bytea geom, integer srid);
```

## Beschreibung

Die Funktion `ST_PointFromWKB` nimmt eine Well-known-Binary Darstellung der Geometrie und eine Id für das Koordinatenreferenzsystem (SRID) entgegen und erzeugt eine Instanz des entsprechenden geometrischen Datentyps - in diesem Fall eine Geometrie vom Typ `POINT`. Diese Funktion übernimmt die Rolle der Geometrie-Factory in SQL.

Wenn keine SRID angegeben ist, wird diese standardmäßig auf 0 gesetzt. `NULL` wird zurückgegeben, wenn die Eingabe `bytea` keinen `POINT` darstellt.



Diese Methode implementiert die [OGC Simple Features Implementation Specification for SQL 1.1](#). s3.2.7.2



Diese Methode setzt die SQL/MM-Spezifikation um. SQL-MM 3: 6.1.9



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



Diese Methode unterstützt kreisförmige Strings und Kurven.

## Beispiele

```
SELECT
  ST_AsText (
    ST_PointFromWKB (
      ST_AsEWKB ('POINT(2 5)::geometry)
    )
  );
st_astext
-----
POINT(2 5)
(1 row)

SELECT
  ST_AsText (
    ST_PointFromWKB (
      ST_AsEWKB ('LINESTRING(2 5, 2 6)::geometry)
    )
  );
st_astext
-----
(1 row)
```

## Siehe auch

[ST\\_GeomFromWKB](#), [ST\\_LineFromWKB](#)

### 7.8.2.7 ST\_WKBToSQL

`ST_WKBToSQL` — Gibt einen geometrischen Datentyp (`ST_Geometry`) aus einer Well-known-Binary (WKB) Darstellung zurück. Ein Synonym für `ST_GeomFromWKB`, welches jedoch keine SRID annimmt

## Synopsis

geometry `ST_WKBToSQL`(bytea WKB);

## Beschreibung



Diese Methode setzt die SQL/MM-Spezifikation um. SQL-MM 3: 5.1.36

**Siehe auch**[ST\\_GeomFromWKB](#)**7.8.3 Weitere Formate****7.8.3.1 ST\_Box2dFromGeoHash**

ST\_Box2dFromGeoHash — Gibt die BOX2D einer GeoHash Zeichenkette zurück.

**Synopsis**

```
box2d ST_Box2dFromGeoHash(text geohash, integer precision=full_precision_of_geohash);
```

**Beschreibung**

Gibt die BOX2D einer GeoHash Zeichenkette zurück.

Wenn keine Genauigkeit angegeben ist, gibt ST\_Box2dFromGeoHash ein BOX2D zurück, das auf der vollen Genauigkeit des eingegebenen GeoHash-Strings basiert.

Wenn `precision` angegeben wird, verwendet ST\_Box2dFromGeoHash entsprechend viele Zeichen des GeoHash um die BOX2D zu erzeugen. Niedrigere Werte erzeugen eine größere BOX2D und höhere Werte erhöhen die Genauigkeit.

Verfügbarkeit: 2.1.0

**Beispiele**

```
SELECT ST_Box2dFromGeoHash('9qqj7nmxcgyy4d0dbxqz0');

          st_geomfromgeohash
-----
BOX(-115.172816 36.114646,-115.172816 36.114646)

SELECT ST_Box2dFromGeoHash('9qqj7nmxcgyy4d0dbxqz0', 0);

          st_box2dfromgeohash
-----
BOX(-180 -90,180 90)

SELECT ST_Box2dFromGeoHash('9qqj7nmxcgyy4d0dbxqz0', 10);
          st_box2dfromgeohash
-----
BOX(-115.17282128334 36.1146408319473,-115.172810554504 36.1146461963654)
```

**Siehe auch**[ST\\_GeoHash](#), [ST\\_GeomFromGeoHash](#), [ST\\_PointFromGeoHash](#)**7.8.3.2 ST\_GeomFromGeoHash**

ST\_GeomFromGeoHash — Gibt die Geometrie einer GeoHash Zeichenfolge zurück.

**Synopsis**

geometry **ST\_GeomFromGeoHash**(text geohash, integer precision=full\_precision\_of\_geohash);

**Beschreibung**

Gibt die Geometrie einer GeoHash Zeichenfolge zurück. Der geometrische Datentyp ist ein Polygon, das den GeoHash begrenzt. Wenn keine `precision` angegeben wird, dann gibt `ST_GeomFromGeoHash` ein Polygon zurück, das auf der vollständigen Genauigkeit der GeoHash Zeichenfolge beruht.

Wenn `precision` angegeben wird, verwendet `ST_GeomFromGeoHash` entsprechend viele Zeichen des GeoHash, um das Polygon zu erzeugen.

Verfügbarkeit: 2.1.0

**Beispiele**

```
SELECT ST_AsText(ST_GeomFromGeoHash('9qqj7nmxcggy4d0dbxqz0'));
           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('9qqj7nmxcggy4d0dbxqz0', 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('9qqj7nmxcggy4d0dbxqz0', 10));
           st_astext
-----
POLYGON((-115.17282128334 36.1146408319473,-115.17282128334  ←
          36.1146461963654,-115.172810554504 36.1146461963654,-115.172810554504  ←
          36.1146408319473,-115.17282128334 36.1146408319473))
```

**Siehe auch**

[ST\\_GeoHash](#), [ST\\_Box2dFromGeoHash](#), [ST\\_PointFromGeoHash](#)

**7.8.3.3 ST\_GeomFromGML**

`ST_GeomFromGML` — Nimmt als Eingabe eine GML-Darstellung der Geometrie und gibt ein geometrisches PostGIS-Objekt aus.

**Synopsis**

geometry **ST\_GeomFromGML**(text geomgml);  
 geometry **ST\_GeomFromGML**(text geomgml, integer srid);

## Beschreibung

Erzeugt ein PostGIS ST\_Geometry Objekt aus der OGC GML Darstellung.

ST\_GeomFromGML funktioniert nur bei Fragmenten von GML-Geometrien. Auf das ganze GML-Dokument angewendet führt zu einer Fehlermeldung.

Unterstützte OGC GML Versionen:

- GML 3.2.1 Namespace
- GML 3.1.1 Simple Features profile SF-2 (inkl. GML 3.1.0 und 3.0.0 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.



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



Diese Funktion unterstützt polyedrische Flächen.



Diese Funktion unterstützt Dreiecke und dreieckige unregelmäßige Netzoberflächen (TIN).

GML erlaubt das Mischen von Dimensionen (z.B. 2D und 3D innerhalb der selben MultiGeometry). Da PostGIS Geometrien dies nicht zulassen, wandelt ST\_GeomFromGML die gesamte Geometrie in 2D um, sobald eine fehlende Z-Dimension existiert.

GML unterstützt uneinheitliche Koordinatenreferenzsysteme innerhalb derselben Mehrfachgeometrie. Da dies der geometrische Datentyp von PostGIS nicht unterstützt, wird in diesem Fall die Subgeometrie in das Referenzsystem des Knotens, der die Wurzel darstellt, umprojiziert. Wenn kein Attribut "srsName" für den Knoten der GML-Wurzel vorhanden ist, gibt die Funktion eine Fehlermeldung aus.

Die Funktion ST\_GeomFromGML ist nicht kleinlich, was die explizite Vergabe eines GML-Namensraums betrifft. Bei üblichen Anwendungen können Sie die explizite Vergabe weglassen. Wenn Sie aber das XLink Feature von GML verwenden wollen, müssen Sie den Namensraum explizit angeben.



### Note

SQL/MM Kurvengeometrien werden von der Funktion ST\_GeomFromGML nicht unterstützt

## Beispiele - Eine Einzelgeometrie mit einem srsName

```
SELECT ST_GeomFromGML($$
  <gml:LineString xmlns:gml="http://www.opengis.net/gml"
    srsName="EPSG:4269">
    <gml:coordinates>
      -71.16028,42.258729 -71.160837,42.259112 -71.161143,42.25932
    </gml:coordinates>
  </gml:LineString>
$$);
```

### 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 xmlns:gml="http://www.opengis.net/gml">
<gml:polygonPatches>
  <gml:PolygonPatch>
    <gml:exterior>
      <gml:LinearRing
        ><gml:posList srsDimension="3"
        >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 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 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 [2.2.3](#), [ST\\_AsGML](#), [ST\\_GMLToSQL](#)

**7.8.3.4 ST\_GeomFromGeoJSON**

**ST\_GeomFromGeoJSON** — Nimmt als Eingabe eine GeoJSON-Darstellung der Geometrie und gibt ein geometrisches PostGIS-Objekt aus.

**Synopsis**

```

geometry ST_GeomFromGeoJSON(text geomjson);
geometry ST_GeomFromGeoJSON(json geomjson);
geometry ST_GeomFromGeoJSON(jsonb geomjson);

```

**Beschreibung**

Erzeugt ein geometrisches PostGIS Objekt aus der GeoJSON Darstellung.

**ST\_GeomFromGeoJSON** funktioniert nur bei Fragmenten von JSON-Geometrien. Auf das ganze JSON-Dokument angewendet führt zu einer Fehlermeldung.

Verbessert: 3.0.0 Geometrie wird standardmäßig auf SRID=4326 gesetzt, wenn nicht anders angegeben.

Erweiterung: 2.5.0 unterstützt nun auch die Eingabe von json und jsonb.

Verfügbarkeit: 2.0.0 benötigt - JSON-C >= 0.9

**Note**

Wenn Sie die JSON-C Unterstützung nicht aktiviert haben, sehen Sie eine Fehlermeldung anstatt einer Ausgabe. Um JSON-C zu aktivieren, führen Sie bitte `configure --with-jsondir=/path/to/json-c` aus. Für Einzelheiten siehe Section [2.2.3](#).



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

### Beispiele

```
SELECT ST_AsText(ST_GeomFromGeoJSON('{"type":"Point","coordinates":[-48.23456,20.12345]}')) ←  
As wkt;  
wkt  
-----  
POINT(-48.23456 20.12345)
```

```
-- a 3D linestring  
SELECT ST_AsText(ST_GeomFromGeoJSON('{"type":"LineString","coordinates ←  
":[[1,2,3],[4,5,6],[7,8,9]]}')) As wkt;  
wkt  
-----  
LINESTRING(1 2,4 5,7 8)
```

### Siehe auch

[ST\\_AsText](#), [ST\\_AsGeoJSON](#), [Section 2.2.3](#)

#### 7.8.3.5 ST\_GeomFromKML

`ST_GeomFromKML` — Nimmt als Eingabe eine KML-Darstellung der Geometrie und gibt ein geometrisches PostGIS-Objekt aus.

### Synopsis

geometry `ST_GeomFromKML`(text geomkml);

### Beschreibung

Erzeugt ein PostGIS `ST_Geometry` Objekt aus der OGC KML Darstellung.

`T_GeomFromKML` funktioniert nur bei Fragmenten von KML-Geometrien. Auf das ganze KML-Dokument angewendet führt zu einer Fehlermeldung.

Unterstützte OGC KML Versionen:

- KML 2.2.0 Namespace

OGC KML Standards, vgl.: <http://www.opengeospatial.org/standards/kml>:

Verfügbarkeit: 1.5, benötigt libxml2 2.6+



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



### Note

SQL/MM Kurvengeometrien werden von der Funktion `ST_GeomFromKML` nicht unterstützt



### Beispiele - Eine Einzelgeometrie mit einem srsName

```
SELECT ST_GeomFromKML($$
  <LineString>
    <coordinates>
>-71.1663,42.2614
    -71.1667,42.2616</coordinates>
  </LineString>
$$);
```

### Siehe auch

Section [2.2.3](#), [ST\\_AsKML](#)

### 7.8.3.6 ST\_GeomFromTWKB

ST\_GeomFromTWKB — Erzeugt eine Geometrie aus einer TWKB ("[Tiny Well-Known Binary](#)") Darstellung.

### Synopsis

geometry **ST\_GeomFromTWKB**(bytea twkb);

### Beschreibung

Die Funktion ST\_GeomFromTWKB nimmt eine TWKB ("[Tiny Well-Known Binary](#)") Darstellung und erzeugt ein Objekt mit dem entsprechenden geometrischen Datentyp.

### Beispiele

```
SELECT ST_AsText(ST_GeomFromTWKB(ST_AsTWKB('LINESTRING(126 34, 127 35)::geometry')));

      st_astext
-----
LINESTRING(126 34, 127 35)
(1 row)

SELECT ST_AsEWKT(
  ST_GeomFromTWKB(E'\\x620002f7f40dbce4040105')
);

                               st_asewkt
-----
LINESTRING(-113.98 39.198,-113.981 39.195)
(1 row)
```

### Siehe auch

[ST\\_AsTWKB](#)

### 7.8.3.7 ST\_GMLToSQL

ST\_GMLToSQL — Gibt einen spezifizierten ST\_Geometry Wert aus einer GML-Darstellung zurück. Dies ist ein Aliasname für ST\_GeomFromGML

## Synopsis

```
geometry ST_GMLToSQL(text geomgml);  
geometry ST_GMLToSQL(text geomgml, integer srid);
```

## Beschreibung



Diese Methode setzt die SQL/MM-Spezifikation um. SQL-MM 3: 5.1.50 (ausgenommen Unterstützung von Kurven).

Verfügbarkeit: 1.5, benötigt libxml2 1.6+

Erweiterung: Mit 2.0.0 wurde die Unterstützung für polyedrische Oberflächen und TIN eingeführt.

Erweiterung: 2.0.0 Standardwert für den optionalen Parameter SRID eingefügt.

## Siehe auch

Section [2.2.3](#), [ST\\_GeomFromGML](#), [ST\\_AsGML](#)

### 7.8.3.8 ST\_LineFromEncodedPolyline

`ST_LineFromEncodedPolyline` — Erzeugt einen LineString aus einem codierten Linienzug.

## Synopsis

```
geometry ST_LineFromEncodedPolyline(text polyline, integer precision=5);
```

## Beschreibung

Erzeugt einen LineString aus einem codierten Linienzug.

Der optionale Parameter `precision` gibt an wieviele Dezimalstellen der kodierten Polylinie erhalten bleiben. Dieser Wert sollte beim Dekodieren und beim Kodieren ident sein, sonst entstehen inkorrekte Koordinaten.

Siehe <http://developers.google.com/maps/documentation/utilities/polylinealgorithm>

Verfügbarkeit: 2.2.0

## Beispiele

```
-- Create a line string from a polyline  
SELECT ST_AsEWKT(ST_LineFromEncodedPolyline('_p~iF~ps|U_ulLnnqC_mqNvxq`@'));  
-- result --  
SRID=4326;LINESTRING(-120.2 38.5,-120.95 40.7,-126.453 43.252)  
  
-- Select different precision that was used for polyline encoding  
SELECT ST_AsEWKT(ST_LineFromEncodedPolyline('_p~iF~ps|U_ulLnnqC_mqNvxq`@',6));  
-- result --  
SRID=4326;LINESTRING(-12.02 3.85,-12.095 4.07,-12.6453 4.3252)
```

## Siehe auch

[ST\\_AsEncodedPolyline](#)

### 7.8.3.9 ST\_PointFromGeoHash

ST\_PointFromGeoHash — Gibt einen Punkt von einer GeoHash Zeichenfolge zurück.

#### Synopsis

```
point ST_PointFromGeoHash(text geohash, integer precision=full_precision_of_geohash);
```

#### Beschreibung

Gibt die Geometrie einer GeoHash Zeichenfolge zurück. Der Punkt entspricht dem Mittelpunkt des GeoHas.

Wenn keine `precision` angegeben wird, dann gibt ST\_PointFromGeoHash einen Punkt zurück, der auf der vollständigen Genauigkeit der gegebenen GeoHash Zeichenfolge beruht.

Wenn `precision` angegeben wird, verwendet ST\_PointFromGeoHash entsprechend viele Zeichen des GeoHash, um den Punkt zu erzeugen.

Verfügbarkeit: 2.1.0

#### Beispiele

```
SELECT ST_AsText(ST_PointFromGeoHash('9qqj7nmxcggy4d0dbxqz0'));
      st_astext
-----
POINT(-115.172816 36.114646)

SELECT ST_AsText(ST_PointFromGeoHash('9qqj7nmxcggy4d0dbxqz0', 4));
      st_astext
-----
POINT(-115.13671875 36.123046875)

SELECT ST_AsText(ST_PointFromGeoHash('9qqj7nmxcggy4d0dbxqz0', 10));
      st_astext
-----
POINT(-115.172815918922 36.1146435141563)
```

#### Siehe auch

[ST\\_GeoHash](#), [ST\\_Box2dFromGeoHash](#), [ST\\_GeomFromGeoHash](#)

### 7.8.3.10 ST\_FromFlatGeobufToTable

ST\_FromFlatGeobufToTable — Erstellt eine Tabelle auf der Grundlage der Struktur der FlatGeobuf-Daten.

#### Synopsis

```
void ST_FromFlatGeobufToTable(text schemaname, text tablename, bytea FlatGeobuf input data);
```

#### Beschreibung

Erzeugt eine Tabelle auf der Grundlage der Struktur der FlatGeobuf-Daten. (<http://flatgeobuf.org>).

`schema` Name des Schemas.

`Tabelle` Tabellenname.

`data` Eingabe von FlatGeobuf-Daten.

Verfügbarkeit: 3.2.0

### 7.8.3.11 ST\_FromFlatGeobuf

ST\_FromFlatGeobuf — Liest FlatGeobuf-Daten.

#### Synopsis

setof anyelement **ST\_FromFlatGeobuf**(anyelement Table reference, bytea FlatGeobuf input data);

#### Beschreibung

Liest FlatGeobuf-Daten (<http://flatgeobuf.org>). HINWEIS: PostgreSQL bytea kann 1GB nicht überschreiten.

`tabletype` Verweis auf einen Tabellentyp.

`data` Eingabe FlatGeobuf-Daten.

Verfügbarkeit: 3.2.0

## 7.9 Geometrieausgabe

### 7.9.1 Well-known-Text (WKT) Repräsentation

#### 7.9.1.1 ST\_AsEWKT

ST\_AsEWKT — Gibt die Well-known-Text(WKT) Darstellung der Geometrie mit den SRID-Metadaten zurück.

#### Synopsis

```
text ST_AsEWKT(geometry g1);
text ST_AsEWKT(geometry g1, integer maxdecimaldigits=15);
text ST_AsEWKT(geography g1);
text ST_AsEWKT(geography g1, integer maxdecimaldigits=15);
```

#### Beschreibung

Gibt die Well-Known-Text-Darstellung der Geometrie mit dem Präfix SRID zurück. Das optionale Argument *maxdecimaldigits* kann verwendet werden, um die maximale Anzahl der in der Ausgabe verwendeten Dezimalstellen nach der Fließkommazahl zu verringern (Standardwert: 15).

Um die inverse Konvertierung der EWKT-Darstellung in eine PostGIS-Geometrie durchzuführen, verwenden Sie [ST\\_GeomFromEWKT](#).



#### Warning

Die Verwendung des Parameters *maxdecimaldigits* kann dazu führen, dass die Ausgabegeometrie ungültig wird. Um dies zu vermeiden, verwenden Sie zuerst [ST\\_ReducePrecision](#) mit einer geeigneten Rastergröße.



#### Note

Die WKT-Spezifikation enthält keine SRID. Um das OGC-WKT-Format zu erhalten, verwenden Sie [ST\\_AsText](#).

---



**Warning**

Im WKT-Format wird die Genauigkeit nicht beibehalten. Verwenden Sie daher für den Transport das Format **ST\_AsBinary** oder **ST\_AsEWKB**, um ein fließendes Abschneiden zu verhindern.

Verbessert: 3.1.0 Unterstützung für optionale Präzisionsparameter.

Erweiterung: Mit 2.0.0 wurde die Unterstützung für den geographischen Datentyp, polyedrische Oberflächen, Dreiecke und TIN eingeführt.

- ✔ Diese Funktion unterstützt 3d und lässt den Z-Index nicht fallen.
- ✔ Diese Methode unterstützt kreisförmige Strings und Kurven.
- ✔ Diese Funktion unterstützt polyedrische Flächen.
- ✔ Diese Funktion unterstützt Dreiecke und dreieckige unregelmäßige Netzoberflächen (TIN).

**Beispiele**

```
SELECT ST_AsEWKT('0103000020E61000000100000005000000000000
                 000000000000000000000000000000000000000000000000000000000000000000
                 F03F000000000000F03F000000000000F03F000000000000F03
                 F00000000000000000000000000000000000000000000000000000000000000000::geometry);

            st_asewkt
-----
SRID=4326;POLYGON((0 0,0 1,1 1,1 0,0 0))
(1 row)

SELECT ST_AsEWKT('01080000800300000000000000000060 ↔
                 E30A4100000000785C0241000000000000F03F0000000018
E20A4100000000485F02410000000000000400000000018
E20A4100000000305C02410000000000000840')

--st_asewkt---
CIRCULARSTRING(220268 150415 1,220227 150505 2,220227 150406 3)
```

**Siehe auch**

[ST\\_AsBinary](#), [ST\\_AsEWKB](#), [ST\\_AsText](#), [ST\\_GeomFromEWKT](#)

**7.9.1.2 ST\_AsText**

ST\_AsText — Gibt die Well-known-Text(WKT) Darstellung der Geometrie/Geographie ohne die SRID Metadaten zurück.

**Synopsis**

```
text ST_AsText(geometry g1);
text ST_AsText(geometry g1, integer maxdecimaldigits = 15);
text ST_AsText(geography g1);
text ST_AsText(geography g1, integer maxdecimaldigits = 15);
```

**Beschreibung**

Liefert die OGC **Well-Known Text** (WKT) Darstellung der Geometrie/Geographie. Das optionale Argument *maxdecimaldigits* kann verwendet werden, um die Anzahl der Nachkommastellen in der Ausgabe von Ordinaten zu begrenzen (Standardwert: 15).

Um die umgekehrte Konvertierung der WKT-Darstellung in PostGIS-Geometrie durchzuführen, verwenden Sie **ST\_GeomFromText**.



**Note**

Die standardmäßige OGC-WKT-Darstellung enthält den SRID nicht. Um den SRID als Teil der Ausgabedarstellung aufzunehmen, verwenden Sie die nicht standardisierte PostGIS-Funktion **ST\_AsEWKT**



**Warning**

Bei der textuellen Darstellung von Zahlen in WKT wird möglicherweise nicht die volle Gleitkommagenauigkeit eingehalten. Um die volle Genauigkeit bei der Datenspeicherung oder beim Datentransport zu gewährleisten, ist es am besten, das Format **Well-Known Binary** (WKB) zu verwenden (siehe **ST\_AsBinary** und *maxdecimaldigits*).



**Warning**

Die Verwendung des Parameters *maxdecimaldigits* kann dazu führen, dass die Ausgabegeometrie ungültig wird. Um dies zu vermeiden, verwenden Sie zuerst **ST\_ReducePrecision** mit einer geeigneten Rastergröße.

Verfügbarkeit: 1.5 - Unterstützung von geographischen Koordinaten.

Erweiterung: 2.5 - der optionale Parameter "precision" wurde eingeführt.

- Diese Methode implementiert die **OGC Simple Features Implementation Specification for SQL 1.1**. s2.1.1.1
- Diese Methode setzt die SQL/MM-Spezifikation um. SQL-MM 3: 5.1.25
- Diese Methode unterstützt kreisförmige Strings und Kurven.

**Beispiele**

```
SELECT ST_AsText('010300000001000000050000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000
F03F000000000000F03F000000000000F03F000000000000F03
F000000000000000000000000000000000000000000000000000000');

  st_astext
-----
POLYGON((0 0,0 1,1 1,1 0,0 0))
```

Die Standardeinstellung ist die Ausgabe mit voller Präzision.

```
SELECT ST_AsText('POINT(111.1111111 1.1111111)');
  st_astext
-----
POINT(111.1111111 1.1111111)
```

Das Argument *maxdecimaldigits* kann verwendet werden, um die Ausgabegenauigkeit zu begrenzen.

```
SELECT ST_AsText('POINT(111.1111111 1.1111111)', 2);
  st_astext
-----
POINT(111.11 1.11)
```

**Siehe auch**

[ST\\_AsBinary](#), [ST\\_AsEWKB](#), [ST\\_AsEWKT](#), [ST\\_GeomFromText](#)

**7.9.2 Well-known-Binary (WKB) Repräsentation****7.9.2.1 ST\_AsBinary**

`ST_AsBinary` — Rückgabe der OGC/ISO Well-Known Binary (WKB)-Darstellung der Geometrie/Geografie ohne SRID-Metadaten.

**Synopsis**

```
bytea ST_AsBinary(geometry g1);
bytea ST_AsBinary(geometry g1, text NDR_or_XDR);
bytea ST_AsBinary(geography g1);
bytea ST_AsBinary(geography g1, text NDR_or_XDR);
```

**Beschreibung**

Gibt die OGC/ISO **Well-Known Binary** (WKB) Darstellung der Geometrie zurück. Bei der ersten Funktionsvariante wird standardmäßig die Endian-Kodierung der Servermaschine verwendet. Die zweite Funktionsvariante nimmt ein Textargument entgegen, das die Endian-Kodierung angibt, entweder Little-Endian ('NDR') oder Big-Endian ('XDR').

Das WKB-Format ist nützlich, um Geometriedaten aus der Datenbank zu lesen und dabei die volle numerische Präzision beizubehalten. Dadurch wird die Präzisionsrundung vermieden, die bei Textformaten wie WKT auftreten kann.

Um die inverse Konvertierung von WKB in PostGIS-Geometrie durchzuführen, verwenden Sie [ST\\_GeomFromWKB](#).

**Note**

Das OGC/ISO WKB-Format enthält keinen SRID. Um das EWKB-Format zu erhalten, das den SRID enthält, verwenden Sie [ST\\_AsEWKB](#)

**Note**

Das Standardverhalten in PostgreSQL 9.0 wurde geändert, um bytea in Hex-Kodierung auszugeben. Wenn Ihre GUI-Tools das alte Verhalten erfordern, dann setzen Sie `SET bytea_output='escape'` in Ihrer Datenbank.

Erweiterung: Mit 2.0.0 wurde die Unterstützung für polyedrische Oberflächen, Dreiecke und TIN eingeführt.

Erweiterung: 2.0.0 - Unterstützung für höherdimensionale Koordinatensysteme eingeführt.

Erweiterung: 2.0.0 Unterstützung zum Festlegen des Endian beim geographischen Datentyp eingeführt.

Verfügbarkeit: 1.5.0 Unterstützung von 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: `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`.



Diese Methode implementiert die [OGC Simple Features Implementation Specification for SQL 1.1](#). s2.1.1.1



Diese Methode setzt die SQL/MM-Spezifikation um. SQL-MM 3: 5.1.37



Diese Methode unterstützt kreisförmige Strings und Kurven.

- ✓ Diese Funktion unterstützt polyedrische Flächen.
- ✓ Diese Funktion unterstützt Dreiecke und dreieckige unregelmäßige Netzoberflächen (TIN).
- ✓ Diese Funktion unterstützt 3d und lässt den Z-Index nicht fallen.

### Beispiele

```
SELECT ST_AsBinary(ST_GeomFromText('POLYGON((0 0,0 1,1 1,1 0,0 0))',4326));

           st_asbinary
-----
\x010300000001000000050000000000000000000000000000000000000000000000000000000000000000
000000f03f000000000000f03f000000000000f03f000000000000f03f00000000000000000000000000
0000000000000000000000000000
```

```
SELECT ST_AsBinary(ST_GeomFromText('POLYGON((0 0,0 1,1 1,1 0,0 0))',4326), 'XDR');

           st_asbinary
-----
\x000000000030000000100000005000000000000000000000000000000000000000000000000000003ff000
00000000003ff00000000000003ff0000000000003ff00000000000000000000000000000000000000
0000000000000000000000000000
```

### Siehe auch

[ST\\_GeomFromWKB](#), [ST\\_AsEWKB](#), [ST\\_AsTWKB](#), [ST\\_AsText](#),

#### 7.9.2.2 ST\_AsEWKB

`ST_AsEWKB` — Rückgabe der Extended Well-Known Binary (EWKB) Darstellung der Geometrie mit SRID-Metadaten.

### Synopsis

```
bytea ST_AsEWKB(geometry g1);
bytea ST_AsEWKB(geometry g1, text NDR_or_XDR);
```

### Beschreibung

Gibt die **Extended Well-Known Binary** (EWKB) Darstellung der Geometrie mit SRID-Metadaten zurück. Bei der ersten Funktionsvariante wird standardmäßig die Endian-Kodierung der Servermaschine verwendet. Die zweite Funktionsvariante nimmt ein Textargument entgegen, das die Endian-Kodierung angibt, entweder Little-Endian ('NDR') oder Big-Endian ('XDR').

Das WKB-Format ist nützlich, um Geometriedaten aus der Datenbank zu lesen und dabei die volle numerische Präzision beizubehalten. Dadurch wird die Präzisionsrundung vermieden, die bei Textformaten wie WKT auftreten kann.

Um die inverse Konvertierung von EWKB in PostGIS-Geometrie durchzuführen, verwenden Sie [ST\\_GeomFromEWKB](#).



#### Note

Um das OGC/ISO WKB-Format zu erhalten, verwenden Sie [ST\\_AsBinary](#). Beachten Sie, dass das OGC/ISO WKB-Format nicht den SRID enthält.





## Beispiele

```
SELECT ST_AsHEXEWKB(ST_GeomFromText('POLYGON((0 0,0 1,1 1,1 0,0 0))',4326));
      which gives same answer as

      SELECT ST_GeomFromText('POLYGON((0 0,0 1,1 1,1 0,0 0))',4326)::text;

      st_ashexewkb
      -----
      0103000020E6100000010000000500
      000000000000000000000000000000
      000000000000000000000000000000F03F
      0000000000000000F03F000000000000F03F000000000000F03
      F00000000000000000000000000000000000000000000000000000
```

## 7.9.3 Weitere Formate

### 7.9.3.1 ST\_AsEncodedPolyline

ST\_AsEncodedPolyline — Erzeugt eine codierte Polylinie aus einer LineString Geometrie.

#### Synopsis

```
text ST_AsEncodedPolyline(geometry geom, integer precision=5);
```

#### Beschreibung

Gibt die Geometrie als kodierte Polylinie aus. Dieses Format wird von "Google Maps" mit precision=5 und von "Open Source Routing Machine" mit precision=5 oder 6 verwendet.

Der optionale Parameter `precision` gibt an wieviele Dezimalstellen der kodierte Polylinie erhalten bleiben. Dieser Wert sollte beim Dekodieren und beim Kodieren ident sein, sonst entstehen inkorrekte Koordinaten.

Verfügbarkeit: 2.2.0

#### Beispiele

##### Grundlegendes

```
SELECT ST_AsEncodedPolyline(GeomFromEWKT('SRID=4326;LINESTRING(-120.2 38.5,-120.95 40.7,-126.453 43.252)'));
      --result--
      |_p~iF~ps|U_ulLnnqC_mqNvxq`@
```

Anwendung in Verbindung mit LINESTRING und ST\_Segmentize für den geographischen Datentyp, und auf Google Maps stellen

```
-- the SQL for Boston to San Francisco, segments every 100 KM
SELECT ST_AsEncodedPolyline(
  ST_Segmentize(
    ST_GeogFromText('LINESTRING(-71.0519 42.4935,-122.4483 37.64)'),
    100000)::geometry) As encodedFlightPath;
```

In JavaScript sieht dies ungefähr wie folgt aus, wobei die \$ Variable durch das Abfrageergebnis ersetzt wird

```
<script type="text/javascript" src="http://maps.googleapis.com/maps/api/js?libraries= ↵
  geometry"
></script>
<script type="text/javascript">
  flightPath = new google.maps.Polyline({
    path: google.maps.geometry.encoding.decodePath("$encodedFlightPath ↵
    "),
    map: map,
    strokeColor: '#0000CC',
    strokeOpacity: 1.0,
    strokeWeight: 4
  });
</script>
```

### Siehe auch

[ST\\_LineFromEncodedPolyline](#), [ST\\_Segmentize](#)

#### 7.9.3.2 ST\_AsFlatGeobuf

ST\_AsFlatGeobuf — Rückgabe einer FlatGeobuf-Darstellung einer Reihe von Zeilen.

### Synopsis

```
bytea ST_AsFlatGeobuf(anelement set row);
bytea ST_AsFlatGeobuf(anelement row, bool index);
bytea ST_AsFlatGeobuf(anelement row, bool index, text geom_name);
```

### Beschreibung

Gibt eine FlatGeobuf-Darstellung (<http://flatgeobuf.org>) einer Reihe von Zeilen zurück, die einer FeatureCollection entsprechen. HINWEIS: PostgreSQL bytea kann 1GB nicht überschreiten.

`row` Datenzeilen mit zumindest einer Geometriespalte.

`index` schaltet die Erstellung von räumlichen Indizes ein. Die Voreinstellung ist `false`.

`geom_name` ist die Bezeichnung der Geometriespalte in den Datenzeilen. Wenn NULL, dann wird standardmäßig die erste aufgefundene Geometriespalte verwendet.

Verfügbarkeit: 3.2.0

#### 7.9.3.3 ST\_AsGeobuf

ST\_AsGeobuf — Gibt eine Menge an Zeilen in der Geobuf Darstellung aus.

### Synopsis

```
bytea ST_AsGeobuf(anelement set row);
bytea ST_AsGeobuf(anelement row, text geom_name);
```

## Beschreibung

Gibt Zeilen einer FeatureCollection in der Geobuf Darstellung (<https://github.com/mapbox/geobuf>) aus. Von jeder Eingabegeometrie wird die maximale Genauigkeit analysiert, um eine optimale Speicherung zu erreichen. Anmerkung: In der jetzigen Form kann Geobuf nicht "gestreamt" werden, wodurch die gesamte Ausgabe im Arbeitsspeicher zusammengestellt wird.

row Datenzeilen mit zumindest einer Geometriespalte.

geom\_name ist die Bezeichnung der Geometriespalte in den Datenzeilen. Wenn NULL, dann wird standardmäßig die erste aufgefundene Geometriespalte verwendet.

Verfügbarkeit: 2.4.0

## Beispiele

```
SELECT encode(ST_AsGeobuf(q, 'geom'), 'base64')
  FROM (SELECT ST_GeomFromText('POLYGON((0 0,0 1,1 1,1 0,0 0))') AS geom) AS q;
st_asgeobuf
-----
GAAiEAoOCgwIBBoIAAAAAgIAAAE=
```

### 7.9.3.4 ST\_AsGeoJSON

ST\_AsGeoJSON — Rückgabe einer Geometrie oder eines Merkmals im GeoJSON-Format.

#### Synopsis

```
text ST_AsGeoJSON(record feature, text geom_column="", integer maxdecimaldigits=9, boolean pretty_bool=false, text id_column="")
text ST_AsGeoJSON(geometry geom, integer maxdecimaldigits=9, integer options=8);
text ST_AsGeoJSON(geography geog, integer maxdecimaldigits=9, integer options=0);
```

#### Beschreibung

Gibt eine Geometrie als GeoJSON "geometry"-Objekt oder eine Zeile als GeoJSON "feature"-Objekt zurück.

Die resultierenden GeoJSON-Geometrie- und Merkmalsdarstellungen entsprechen den [GeoJSON-Spezifikationen RFC 7946](#), außer wenn die geparsten Geometrien mit einem anderen CRS als WGS84-Längen- und Breitengrad referenziert sind ([EPSG:4326, urn:ogc:def:crs:OGC::CRS84](#)); dem GeoJSON-Geometrieobjekt wird dann standardmäßig ein kurzer CRS-SRID-Bezeichner beigefügt. Es werden sowohl 2D- als auch 3D-Geometrien unterstützt. GeoJSON unterstützt nur SFS 1.1 Geometrietypen (z.B. keine Unterstützung für Kurven).

Der Parameter `geom_column` wird verwendet, um zwischen mehreren Geometriespalten zu unterscheiden. Wird er weggelassen, wird die erste Geometriespalte im Datensatz ermittelt. Umgekehrt erspart die Übergabe des Parameters die Suche nach dem Spaltentyp.

Der Parameter `maxdecimaldigits` kann zur Reduzierung der Nachkommastellen in der Ausgabe verwendet werden (standardmäßig 9). Wenn EPSG:4326 verwendet wird, kann `maxdecimaldigits=6` eine gute Wahl für viele Karten bei der Bildschirmausgabe sein.



#### Warning

Die Verwendung des Parameters `maxdecimaldigits` kann dazu führen, dass die Ausgabegeometrie ungültig wird. Um dies zu vermeiden, verwenden Sie zuerst [ST\\_ReducePrecision](#) mit einer geeigneten Rastergröße.

Das Argument `options` kann verwendet werden, um BBOX oder CRS in der GeoJSON-Ausgabe hinzuzufügen:

- 0: keine option
- 1: GeoJSON BBOX
- 2: GeoJSON CRS-Kurzform (z.B. EPSG:4326)
- 4: GeoJSON CRS-Langform (z.B. urn:ogc:def:crs:EPSG::4326)
- 8: GeoJSON CRS-Kurzform, außer bei EPSG:4326 (default)

Der Parameter `id_column` wird verwendet, um das Element "id" der zurückgegebenen GeoJSON-Features festzulegen. Gemäß GeoJSON RFC SOLLTE dies immer dann verwendet werden, wenn ein Feature einen häufig verwendeten Bezeichner hat, wie z.B. einen Primärschlüssel. Wenn dies nicht angegeben wird, erhalten die erzeugten Features kein "id"-Mitglied und alle anderen Spalten als die Geometrie, einschließlich potenzieller Schlüssel, landen einfach im "properties"-Mitglied des Features.

In der GeoJSON-Spezifikation ist festgelegt, dass Polygone nach der Rechts-Regel ausgerichtet werden, und einige Clients verlangen diese Ausrichtung. Dies kann durch die Verwendung von `ST_ForcePolygonCCW` sichergestellt werden. Die Spezifikation verlangt auch, dass die Geometrie im WGS84-Koordinatensystem (SRID = 4326) vorliegt. Bei Bedarf kann die Geometrie mit `ST_Transform` in WGS84 projiziert werden: `ST_Transform( geom, 4326 )`.

GeoJSON kann online getestet und angesehen werden unter [geojson.io](http://geojson.io) und [geojsonlint.com](http://geojsonlint.com). Es wird von vielen Web-Mapping-Frameworks unterstützt:

- [Beispiel für ein OpenLayers GeoJSON](#)
- [Beispiel für ein Leaflet GeoJSON](#)
- [Beispiel für ein Mapbox GL GeoJSON](#)

Verfügbarkeit: 1.3.4

Verfügbarkeit: 1.5.0 Unterstützung von geographischen Koordinaten.

Änderung: 2.0.0 Unterstützung für Standardargumente und benannte Argumente.

Änderung: 3.0.0 Unterstützung von Datensätzen bei der Eingabe

Änderung: 3.0.0 Ausgabe der SRID wenn nicht EPSG:4326

Geändert: 3.5.0 erlaubt die Angabe der Spalte, die die Feature-ID enthält



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

## Beispiele

Erzeugen Sie eine FeatureCollection:

```
SELECT json_build_object(
  'type', 'FeatureCollection',
  'features', json_agg(ST_AsGeoJSON(t.*, id_column =
> 'id')::json)
)
FROM ( VALUES (1, 'one', 'POINT(1 1)::geometry),
              (2, 'two', 'POINT(2 2)'),
              (3, 'three', 'POINT(3 3)')
) as t(id, name, geom);
```

```
{"type" : "FeatureCollection", "features" : [{"type": "Feature", "geometry": {"type": "Point", "coordinates": [1,1]}, "id": 1, "properties": {"name": "one"}}, {"type": "Feature", "geometry": {"type": "Point", "coordinates": [2,2]}, "id": 2, "properties": {"name": "two"}}, {"type": "Feature", "geometry": {"type": "Point", "coordinates": [3,3]}, "id": 3, "properties": {"name": "three"}}]}
```

Erzeugen Sie ein Feature:

```
SELECT ST_AsGeoJSON(t.*, id_column =
> 'id')
FROM (VALUES (1, 'one', 'POINT(1 1)::geometry)) AS t(id, name, geom);
```

```
st_asgeojson
```

```
-----
{"type": "Feature", "geometry": {"type": "Point", "coordinates": [1,1]}, "id": 1, "properties": {"name": "one"}}
```

Vergessen Sie nicht, Ihre Daten in WGS84 Längen- und Breitengrade umzuwandeln, um der GeoJSON-Spezifikation zu entsprechen:

```
SELECT ST_AsGeoJSON(ST_Transform(geom,4326)) from fe_edges limit 1;
```

```
st_asgeojson
```

```
-----
{"type": "MultiLineString", "coordinates": [[[[-89.734634999999997, 31.492072000000000],
[-89.734955999999997, 31.492237999999997]]]]}
```

3D-Geometrien werden unterstützt:

```
SELECT ST_AsGeoJSON('LINESTRING(1 2 3, 4 5 6)');
```

```
-----
{"type": "LineString", "coordinates": [[[1,2,3], [4,5,6]]]}
```

Options argument can be used to add BBOX and CRS in GeoJSON output:

```
SELECT ST_AsGeoJSON(ST_SetSRID('POINT(1 1)::geometry', 4326), 9, 4|1);
```

```
-----
{"type": "Point", "crs": {"type": "name", "properties": {"name": "urn:ogc:def:crs:EPSG::4326"}}, "bbox": [1.000000000, 1.000000000, 1.000000000, 1.000000000], "coordinates": [1,1]}
```

**Siehe auch**

[ST\\_GeomFromGeoJSON](#), [ST\\_ForcePolygonCCW](#), [ST\\_Transform](#)

### 7.9.3.5 ST\_AsGML

ST\_AsGML — Gibt die Geometrie als GML-Element - Version 2 oder 3 - zurück.

**Synopsis**

```
text ST_AsGML(geometry geom, integer maxdecimaldigits=15, integer options=0);
text ST_AsGML(geography geog, integer maxdecimaldigits=15, integer options=0, text nprefix=null, text id=null);
text ST_AsGML(integer version, geometry geom, integer maxdecimaldigits=15, integer options=0, text nprefix=null, text id=null);
text ST_AsGML(integer version, geography geog, integer maxdecimaldigits=15, integer options=0, text nprefix=null, text id=null);
```

## Beschreibung

Gibt die Geometrie als ein Geography Markup Language (GML) Element zurück. Ein Versionsparameter kann mit 2 oder 3 angegeben werden. Wenn kein Versionsparameter angegeben ist, wird dieser standardmäßig Version 2 angenommen. Der Parameter `maxdecimaldigits` kann verwendet werden, um die Anzahl der Nachkommastellen bei der Ausgabe zu reduzieren (standardmäßig 15).



### Warning

Die Verwendung des Parameters `maxdecimaldigits` kann dazu führen, dass die Ausgabegeometrie ungültig wird. Um dies zu vermeiden, verwenden Sie zuerst `ST_ReducePrecision` mit einer geeigneten Rastergröße.

GML 2 verweist auf Version 2.1.2, GML 3 auf Version 3.1.1

Der Übergabewert "options" ist ein Bitfeld. Es kann verwendet werden um das Koordinatenreferenzsystem bei der GML Ausgabe zu bestimmen und um die Daten in Länge/Breite anzugeben.

- 0: GML Kurzform für das CRS (z.B. EPSG:4326), Standardwert
- 1: GML Langform für das CRS (z.B. urn:ogc:def:crs:EPSG::4326)
- 2: Nur für GML 3, entfernt das srsDimension Attribut von der Ausgabe.
- 4: Nur für GML 3, Für Linien verwenden Sie bitte den Tag `<LineString>` anstatt `<Curve>`.
- 16: Deklarieren, dass die Daten in Breite/Länge (z.B. SRID=4326) vorliegen. Standardmäßig wird angenommen, dass die Daten planar sind. Diese Option ist nur bei Ausgabe in GML 3.1.1, in Bezug auf die Anordnung der Achsen sinnvoll. Falls Sie diese setzen, werden die Koordinaten von Länge/Breite auf Breite/Länge vertauscht.
- 32: Ausgabe der BBox der Geometrie (Umhüllende/Envelope).

Der Übergabewert 'namespace prefix' kann verwendet werden, um ein benutzerdefiniertes Präfix für den Namensraum anzugeben, oder kein Präfix (wenn leer). Wenn Null oder weggelassen, so wird das Präfix "gml" verwendet.

Verfügbarkeit: 1.3.2

Verfügbarkeit: 1.5.0 Unterstützung von 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.



Diese Methode setzt die SQL/MM-Spezifikation um. SQL-MM IEC 13249-3: 17.2



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



Diese Funktion unterstützt polyedrische Flächen.



Diese Funktion unterstützt Dreiecke und dreieckige unregelmäßige Netzoberflächen (TIN).

**Beispiele: Version 2**

```
SELECT ST_AsGML(ST_GeomFromText('POLYGON((0 0,0 1,1 1,1 0,0 0))',4326));
      st_asgml
      -----
      <gml:Polygon srsName="EPSG:4326"
><gml:outerBoundaryIs
><gml:LinearRing
><gml:coordinates
>0,0 0,1 1,1 1,0 0,0</gml:coordinates
></gml:LinearRing
></gml:outerBoundaryIs
></gml:Polygon>
```

**Beispiele: Version 3**

```
-- Flip coordinates and output extended EPSG (16 | 1)--
SELECT ST_AsGML(3, ST_GeomFromText('POINT(5.234234233242 6.34534534534)',4326), 5, 17);
      st_asgml
      -----
      <gml:Point srsName="urn:ogc:def:crs:EPSG::4326"
><gml:pos
>6.34535 5.23423</gml:pos
></gml:Point>
```

```
-- Output the envelope (32) --
SELECT ST_AsGML(3, ST_GeomFromText('LINESTRING(1 2, 3 4, 10 20)',4326), 5, 32);
      st_asgml
      -----
      <gml:Envelope srsName="EPSG:4326">
        <gml:lowerCorner
>1 2</gml:lowerCorner>
        <gml:upperCorner
>10 20</gml:upperCorner>
      </gml:Envelope>
```

```
-- Output the envelope (32) , reverse (lat lon instead of lon lat) (16), long srs (1)= 32 | ←
16 | 1 = 49 --
SELECT ST_AsGML(3, ST_GeomFromText('LINESTRING(1 2, 3 4, 10 20)',4326), 5, 49);
      st_asgml
      -----
<gml:Envelope srsName="urn:ogc:def:crs:EPSG::4326">
  <gml:lowerCorner
>2 1</gml:lowerCorner>
  <gml:upperCorner
>20 10</gml:upperCorner>
</gml:Envelope>
```

```
-- Polyhedral Example --
SELECT ST_AsGML(3, ST_GeomFromEWKT('POLYHEDRALSURFACE( ((0 0 0, 0 0 1, 0 1 1, 0 1 0, 0 0 0) ←
),
((0 0 0, 0 1 0, 1 1 0, 1 0 0, 0 0 0)), ((0 0 0, 1 0 0, 1 0 1, 0 0 1, 0 0 0)),
((1 1 0, 1 1 1, 1 0 1, 1 0 0, 1 1 0)),
((0 1 0, 0 1 1, 1 1 1, 1 1 0, 0 1 0)), ((0 0 1, 1 0 1, 1 1 1, 0 1 1, 0 0 1)) )'));
      st_asgml
```



```

-----
<gml:PolyhedralSurface>
<gml:polygonPatches>
  <gml:PolygonPatch>
    <gml:exterior>
      <gml:LinearRing>
        <gml:posList srsDimension="3"
>0 0 0 0 0 1 0 1 1 0 1 0 0 0 0</gml:posList>
        </gml:LinearRing>
      </gml:exterior>
    </gml:PolygonPatch>
    <gml:PolygonPatch>
      <gml:exterior>
        <gml:LinearRing>
          <gml:posList srsDimension="3"
>0 0 0 0 1 0 1 1 0 1 0 0 0 0</gml:posList>
          </gml:LinearRing>
        </gml:exterior>
      </gml:PolygonPatch>
    <gml:PolygonPatch>
      <gml:exterior>
        <gml:LinearRing>
          <gml:posList srsDimension="3"
>0 0 0 1 0 0 1 0 1 0 0 1 0 0 0</gml:posList>
          </gml:LinearRing>
        </gml:exterior>
      </gml:PolygonPatch>
    <gml:PolygonPatch>
      <gml:exterior>
        <gml:LinearRing>
          <gml:posList srsDimension="3"
>1 1 0 1 1 1 1 0 1 1 0 0 1 1 0</gml:posList>
          </gml:LinearRing>
        </gml:exterior>
      </gml:PolygonPatch>
    <gml:PolygonPatch>
      <gml:exterior>
        <gml:LinearRing>
          <gml:posList srsDimension="3"
>0 1 0 0 1 1 1 1 1 1 0 0 1 0</gml:posList>
          </gml:LinearRing>
        </gml:exterior>
      </gml:PolygonPatch>
    <gml:PolygonPatch>
      <gml:exterior>
        <gml:LinearRing>
          <gml:posList srsDimension="3"
>0 0 1 1 0 1 1 1 1 0 1 1 0 0 1</gml:posList>
          </gml:LinearRing>
        </gml:exterior>
      </gml:PolygonPatch>
  </gml:polygonPatches>
</gml:PolyhedralSurface>

```

**Siehe auch**[ST\\_GeomFromGML](#)

### 7.9.3.6 ST\_AsKML

ST\_AsKML — Gibt die Geometrie als GML-Element - Version 2 oder 3 - zurück.

#### Synopsis

text **ST\_AsKML**(geometry geom, integer maxdecimaldigits=15, text nprefix=NULL);

text **ST\_AsKML**(geography geog, integer maxdecimaldigits=15, text nprefix=NULL);

#### Beschreibung

Gibt die Geometrie als ein Keyhole Markup Language (KML) Element zurück. Diese Funktion verfügt über mehrere Varianten. Die maximale Anzahl der Dezimalstellen die bei der Ausgabe verwendet wird (standardmäßig 15), die Version ist standardmäßig 2 und der Standardnamensraum hat kein Präfix.



#### Warning

Die Verwendung des Parameters *maxdecimaldigits* kann dazu führen, dass die Ausgabegeometrie ungültig wird. Um dies zu vermeiden, verwenden Sie zuerst **ST\_ReducePrecision** mit einer geeigneten Rastergröße.



#### 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

Geändert: 3.0.0 - Die Signatur der "versionierten" Variante wurde entfernt.



#### Note

Die Ausgabe AsKML funktioniert nicht bei Geometrien ohne SRID



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

## Beispiele

```
SELECT ST_AsKML(ST_GeomFromText('POLYGON((0 0,0 1,1 1,1 0,0 0))',4326));

      st_askml
      -
      <Polygon
><outerBoundaryIs
><LinearRing
><coordinates
>0,0 0,1 1,1 1,0 0,0</coordinates
></LinearRing
></outerBoundaryIs
></Polygon>

--3d linestring
SELECT ST_AsKML('SRID=4326;LINESTRING(1 2 3, 4 5 6)');
      <LineString
><coordinates
>1,2,3 4,5,6</coordinates
></LineString>
```

## Siehe auch

[ST\\_AsSVG](#), [ST\\_AsGML](#)

### 7.9.3.7 ST\_AsLatLonText

ST\_AsLatLonText — Gibt die "Grad, Minuten, Sekunden"-Darstellung für den angegebenen Punkt aus.

## Synopsis

text **ST\_AsLatLonText**(geometry pt, text format=“);

## Beschreibung

Gibt die "Grad, Minuten, Sekunden"-Darstellung des Punktes aus.



### Note

Es wird angenommen, dass der Punkt in einer Breite/Länge-Projektion vorliegt. Die X (Länge) und Y (Breite) Koordinaten werden bei der Ausgabe in den "üblichen" Bereich (-180 to +180 für die Länge, -90 to +90 für die Breite) normalisiert.

Der Textparameter ist eine Zeichenkette für die Formatierung der Ausgabe, ähnlich wie die Zeichenkette für die Formatierung der Datumsausgabe. Gültige Zeichen sind "D" für Grad/Degrees, "M" für Minuten, "S" für Sekunden, und "C" für die Himmelsrichtung (NSEW). DMS Zeichen können wiederholt werden, um die gewünschte Zeichenbreite und Genauigkeit anzugeben ("SSS.SSSS" bedeutet z.B. " 1.0023").

"M", "S", und "C" sind optional. Wenn "C" weggelassen wird, werden Grad mit einem "-" Zeichen versehen, wenn Süd oder West. Wenn "S" weggelassen wird, werden die Minuten als Dezimalzahl mit der vorgegebenen Anzahl an Kommastellen angezeigt. Wenn "M" 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_aslatlonText
-----
2\textdegree{}19'29.928"S 3\textdegree{}14'3.243"W
```

### Ein Format angeben (identisch mit Standardformat).

```
SELECT (ST_AsLatLonText('POINT (-3.2342342 -2.32498)', 'D\textdegree{}M'S.SSS"C'));
      st_aslatlonText
-----
2\textdegree{}19'29.928"S 3\textdegree{}14'3.243"W
```

### Andere Zeichen als D, M, S, C und "." werden lediglich durchgereicht.

```
SELECT (ST_AsLatLonText('POINT (-3.2342342 -2.32498)', 'D degrees, M minutes, S seconds to the C'));
      st_aslatlonText
-----
2 degrees, 19 minutes, 30 seconds to the S 3 degrees, 14 minutes, 3 seconds to the W
```

### Grad mit einem Vorzeichen versehen - anstatt der Himmelsrichtung.

```
SELECT (ST_AsLatLonText('POINT (-3.2342342 -2.32498)', 'D\textdegree{}M'S.SSS"));
      st_aslatlonText
-----
-2\textdegree{}19'29.928" -3\textdegree{}14'3.243"
```

### Dezimalgrad.

```
SELECT (ST_AsLatLonText('POINT (-3.2342342 -2.32498)', 'D.DDDD degrees C'));
      st_aslatlonText
-----
2.3250 degrees S 3.2342 degrees W
```

### Überhöhte Werte werden normalisiert.

```
SELECT (ST_AsLatLonText('POINT (-302.2342342 -792.32498)'));
      st_aslatlonText
-----
72\textdegree{}19'29.928"S 57\textdegree{}45'56.757"E
```

## 7.9.3.8 ST\_AsMARC21

ST\_AsMARC21 — Gibt die Geometrie als MARC21/XML-Datensatz mit einem geografischen Datenfeld (034) zurück.

### Synopsis

```
text ST_AsMARC21 ( geometry geom , text format='hddmmss' );
```

### Beschreibung

Diese Funktion gibt einen MARC21/XML-Datensatz mit **Codierten kartographisch-mathematischen Daten** zurück, der die Bounding Box einer bestimmten Geometrie darstellt. Der Parameter `Format` erlaubt die Codierung der Koordinaten in den Unterfeldern `$d`, `$e`, `$f` und `$g` in allen vom MARC21/XML-Standard unterstützten Formaten zu codieren. Gültige Formate sind:

- Himmelsrichtung, Grad, Minuten und Sekunden (Standard): hdddmms
- Dezimalgrad mit Himmelsrichtung: hddd.dddddd
- Dezimalgrad ohne Himmelsrichtung: ddd.dddddd
- Dezimalminuten mit Himmelsrichtung: hdddm.mmm
- Dezimalminuten ohne Himmelsrichtung: dddm.mmm
- Dezimalsekunden mit Himmelsrichtung: hdddmss.sss

Das Dezimalzeichen kann auch ein Komma sein, z.B. hdddm,mmm.

Die Genauigkeit von Dezimalformaten kann durch die Anzahl der Zeichen nach dem Dezimalzeichen begrenzt werden, z. B. hddm.mm für Dezimalminuten mit einer Genauigkeit von zwei Dezimalstellen.

Diese Funktion ignoriert die Dimensionen Z und M.

LOC MARC21/XML-Versionen werden unterstützt:

- [MARC21/XML 1.1](#)

Verfügbarkeit: 3.3.0



#### Note

Diese Funktion unterstützt keine Geometrien, die nicht lon/lat sind, da diese nicht vom MARC21/XML-Standard (Coded Cartographic Mathematical Data) unterstützt werden.



#### Note

Der MARC21/XML-Standard bietet keine Möglichkeit, das räumliche Referenzsystem für kartografisch-mathematische Daten zu kommentieren, was bedeutet, dass diese Information nach der Konvertierung in MARC21/XML verloren geht.

## Beispiele

Konvertierung eines POINT in MARC21/XML im Format hdddmss (Standard)

```
SELECT ST_AsMARC21 ('SRID=4326;POINT(-4.504289 54.253312) '::geometry);

          st_asmarc21
-----
<record xmlns="http://www.loc.gov/MARC21/slim">
  <datafield tag="034" ind1="1" ind2=" ">
    <subfield code="a"
>a</subfield>
    <subfield code="d"
>W0043015</subfield>
    <subfield code="e"
>W0043015</subfield>
    <subfield code="f"
>N0541512</subfield>
    <subfield code="g"
>N0541512</subfield>
  </datafield>
</record>
```

## Konvertierung eines POLYGON in MARC21/XML in dezimaler Form

```

SELECT ST_AsMARC21 ('SRID=4326;POLYGON((-4.5792388916015625 ↔
54.18172660239091,-4.56756591796875 ↔
54.196993557130355,-4.546623229980469 ↔
54.18313300502024,-4.5792388916015625 54.18172660239091))'::geometry, ' ↔
hddd.dddd');

<record xmlns="http://www.loc.gov/MARC21/slim">
  <datafield tag="034" ind1="1" ind2=" ">
    <subfield code="a"
>a</subfield>
    <subfield code="d"
>W004.5792</subfield>
    <subfield code="e"
>W004.5466</subfield>
    <subfield code="f"
>N054.1970</subfield>
    <subfield code="g"
>N054.1817</subfield>
  </datafield>
</record>

```

Konvertierung einer GEOMETRYCOLLECTION in MARC21/XML im Dezimalminutenformat. Die Reihenfolge der Geometrien in der MARC21/XML-Ausgabe entspricht der Reihenfolge in der Sammlung.

```

SELECT ST_AsMARC21 ('SRID=4326;GEOMETRYCOLLECTION(POLYGON((13.1 ↔
52.65,13.516666666666667 52.65,13.516666666666667 52.38333333333333,13.1 ↔
52.38333333333333,13.1 52.65)),POINT(-4.5 54.25))'::geometry, 'hdddmm. ↔
mmmm');

          st_asmarc21
-----
<record xmlns="http://www.loc.gov/MARC21/slim">
  <datafield tag="034" ind1="1" ind2=" ">
    <subfield code="a"
>a</subfield>
    <subfield code="d"
>E01307.0000</subfield>
    <subfield code="e"
>E01331.0000</subfield>
    <subfield code="f"
>N05240.0000</subfield>
    <subfield code="g"
>N05224.0000</subfield>
  </datafield>
  <datafield tag="034" ind1="1" ind2=" ">
    <subfield code="a"
>a</subfield>
    <subfield code="d"
>W00430.0000</subfield>
    <subfield code="e"
>W00430.0000</subfield>
    <subfield code="f"
>N05415.0000</subfield>

```

```
        <subfield code="g"
>N05415.0000</subfield>
        </datafield>
</record>
```

## Siehe auch

[ST\\_GeomFromMARC21](#)

### 7.9.3.9 ST\_AsMVTGeom

`ST_AsMVTGeom` — Transformiert eine Geometrie in den Koordinatenraum einer MVT-Kachel.

## Synopsis

geometry `ST_AsMVTGeom`(geometry geom, box2d bounds, integer extent=4096, integer buffer=256, boolean clip\_geom=true);

## Beschreibung

Transformiert eine Geometrie in den Koordinatenraum einer MVT-Kachel ([Mapbox Vector Tile](#)) und beschneidet sie gegebenenfalls auf die Kachelgrenzen. Die Geometrie muss im Koordinatensystem der Zielkarte liegen (ggf. mit [ST\\_Transform](#)). Üblicherweise ist dies [Web Mercator](#) (SRID:3857).

Die Funktion versucht, die Gültigkeit der Geometrie zu erhalten, und korrigiert sie gegebenenfalls. Dies kann dazu führen, dass die Ergebnisgeometrie auf eine niedrigere Dimension kollabiert.

Die rechteckigen Grenzen der Kachel im Koordinatenraum der Zielkarte müssen angegeben werden, damit die Geometrie transformiert und bei Bedarf beschnitten werden kann. Die Begrenzungen können mit [ST\\_TileEnvelope](#) erzeugt werden.

Diese Funktion wird verwendet, um die Geometrie in den von [ST\\_AsMVT](#) benötigten Kachelkoordinatenraum zu konvertieren.

`geom` ist die zu transformierende Geometrie, im Koordinatensystem der Zielkarte.

`bounds` ist die rechteckige Begrenzung der Kachel im Kartenkoordinatenraum, ohne Puffer.

`extent` ist die Größe der Kachelausdehnung im Kachelkoordinatenraum, wie in der [MVT-Spezifikation](#) definiert. Der Standardwert ist 4096.

`buffer` ist die Puffergröße im Kachelkoordinatenraum für das Clipping der Geometrie. Der Standardwert ist 256.

`clip_geom` ist ein boolescher Wert, der bestimmt, ob Geometrien abgeschnitten oder unverändert kodiert werden. Der Standardwert ist `true`.

Verfügbarkeit: 2.4.0



## Note

Ab 3.0 kann zum Ausschneiden und Validieren von MVT-Polygonen bei der Konfiguration `Wagyu` gewählt werden. Diese Bibliothek ist schneller und liefert genauere Ergebnisse als die standardmäßige GEOS-Bibliothek, sie kann aber kleine Polygone verwerfen.

## Beispiele

```
SELECT ST_AsText(ST_AsMVTGeom(
    ST_GeomFromText('POLYGON ((0 0, 10 0, 10 5, 0 -5, 0 0))'),
    ST_MakeBox2D(ST_Point(0, 0), ST_Point(4096, 4096)),
    4096, 0, false));
           st_astext
-----
MULTIPOLYGON(((5 4096,10 4091,10 4096,5 4096)),((5 4096,0 4101,0 4096,5 4096)))
```

Kanonisches Beispiel für eine Web-Mercator-Kachel unter Verwendung einer berechneten Kachelgrenze zur Abfrage und zum Ausschneiden von Geometrie.

```
SELECT ST_AsMVTGeom(
    ST_Transform( geom, 3857 ),
    ST_TileEnvelope(12, 513, 412), extent =
> 4096, buffer =
> 64) AS geom
FROM data
WHERE geom && ST_TileEnvelope(12, 513, 412, margin =
> (64.0 / 4096))
```

## Siehe auch

[ST\\_AsMVT](#), [ST\\_TileEnvelope](#), [PostGIS\\_Wagyu\\_Version](#)

### 7.9.3.10 ST\_AsMVT

`ST_AsMVT` — Aggregatfunktion, die eine MVT-Darstellung einer Reihe von Zeilen zurückgibt.

## Synopsis

```
bytea ST_AsMVT(anyelement set row);
bytea ST_AsMVT(anyelement row, text name);
bytea ST_AsMVT(anyelement row, text name, integer extent);
bytea ST_AsMVT(anyelement row, text name, integer extent, text geom_name);
bytea ST_AsMVT(anyelement row, text name, integer extent, text geom_name, text feature_id_name);
```

## Beschreibung

Eine Aggregatfunktion, die eine binäre **Mapbox Vector Tile**-Darstellung eines Satzes von Zeilen zurückgibt, die einer Kachelebene entsprechen. Die Zeilen müssen eine Geometriespalte enthalten, die als Feature-Geometrie kodiert wird. Die Geometrie muss im Kachelkoordinatenraum vorliegen und gemäß der **MVT-Spezifikation** gültig sein. `ST_AsMVTGeom` kann zur Transformation der Geometrie in den Kachelkoordinatenraum verwendet werden. Andere Zeilenspalten werden als Feature-Attribute kodiert.

Das **Mapbox Vector Tile** Format kann Geoobjekte mit unterschiedlichen Attributen speichern. Um diese Möglichkeit zu nutzen, muss eine JSONB-Spalte in den Datensätzen mitgeliefert werden, welche in einer tieferen Ebene die JSON-Objekte enthält. Die Schlüssel und Werte im JSONB werden als Featureattribute kodiert.

Durch das Aneinanderhängen mehrerer Funktionsaufrufe mittels `||`, können Tiles mit mehreren Ebenen erstellt werden.



### Important

Darf nicht mit einer `GEOMETRYCOLLECTION` im Datensatz aufgerufen werden. Es kann aber `ST_AsMVTGeom` verwendet werden, um eine Sammelgeometrie einzubinden.



`row` Datenzeilen mit zumindest einer Geometriespalte.

`name` ist die Bezeichnung der Ebene. Standardmäßig die Zeichenkette "default".

`extent` ist die Ausdehnung der Tiles in Bildschirmseinheiten. Standardmäßig 4096.

`geom_name` ist der Name der Geometriespalte in den Zeilendaten. Standard ist die erste Geometriespalte. Beachten Sie, dass PostgreSQL standardmäßig automatisch **unquotierte Bezeichner in Kleinbuchstaben umwandelt**, was bedeutet, dass dieser Parameter in Kleinbuchstaben angegeben werden muss, es sei denn, die Geometriespalte ist in Anführungszeichen gesetzt, z. B. "MyMVTGeom".

`feature_id_name` ist die Bezeichnung der Feature-ID Spalte im Datensatz. Ist der Übergabewert NULL oder negativ, dann wird die Feature-ID nicht gesetzt. Die erste Spalte mit einem passenden Namen und einem gültigen Datentyp (Smallint, Integer, Bigint) wird als Feature-ID verwendet, alle nachfolgenden Spalten werden als Eigenschaften hinzugefügt. JSON-Properties werden nicht unterstützt.

Erweiterung: 3.0 - Unterstützung für eine Feature-ID.

Erweiterung: 2.5.0 - Unterstützung von nebenläufigen Abfragen.

Verfügbarkeit: 2.4.0

## Beispiele

```
WITH mvtgeom AS
(
  SELECT ST_AsMVTGeom(geom, ST_TileEnvelope(12, 513, 412), extent =
> 4096, buffer =
> 64) AS geom, name, description
  FROM points_of_interest
  WHERE geom && ST_TileEnvelope(12, 513, 412, margin =
> (64.0 / 4096))
)
SELECT ST_AsMVT(mvtgeom.*)
FROM mvtgeom;
```

## Siehe auch

[ST\\_AsMVTGeom](#), [ST\\_TileEnvelope](#)

### 7.9.3.11 ST\_AsSVG

`ST_AsSVG` — Gibt eine Geometrie als SVG-Pfad aus.

## Synopsis

```
text ST_AsSVG(geometry geom, integer rel=0, integer maxdecimaldigits=15);
text ST_AsSVG(geography geog, integer rel=0, integer maxdecimaldigits=15);
```

## Beschreibung

Gibt die Geometrie als Skalare Vektor Graphik (SVG-Pfadgeometrie) aus. Verwenden Sie 1 als zweiten Übergabewert um die Pfadgeometrie in relativen Schritten zu implementieren; Standardmäßig (oder 0) verwendet absolute Schritte. Der dritte Übergabewert kann verwendet werden, um die maximale Anzahl der Dezimalstellen bei der Ausgabe einzuschränken (standardmäßig 15). Punktgeometrie wird als `cx/cy` übersetzt wenn der Übergabewert 'rel' gleich 0 ist, `x/y` wenn 'rel' 1 ist. Mehrfachgeometrie wird durch Beistriche (",") getrennt, Sammelgeometrie wird durch Strichpunkt (";") getrennt.

Für die Arbeit mit PostGIS-SVG-Grafiken gibt es die `pg_svg`-Bibliothek, die plpgsql-Funktionen für die Arbeit mit Ausgaben von `ST_AsSVG` bereitstellt.

Verbessert: 3.4.0 zur Unterstützung aller Kurventypen

Änderung: 2.0.0 verwendet Standardargumente und unterstützt benannte Argumente.



#### 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.



Diese Methode unterstützt kreisförmige Strings und Kurven.

### Beispiele

```
SELECT ST_AsSVG('POLYGON((0 0,0 1,1 1,1 0,0 0))'::geometry);
```

```
st_assvg
```

```
-----
```

```
M 0 0 L 0 -1 1 -1 1 0 Z
```

### Kreisförmige Schnur

```
SELECT ST_AsSVG( ST_GeomFromText ('CIRCULARSTRING(-2 0,0 2,2 0,0 2,2 4)') );
```

```
st_assvg
```

```
-----
```

```
M -2 0 A 2 2 0 0 1 2 0 A 2 2 0 0 1 2 -4
```

### Multi-Kurve

```
SELECT ST_AsSVG('MULTICURVE((5 5,3 5,3 3,0 3),  
CIRCULARSTRING(0 0,2 1,2 2))'::geometry, 0, 0);
```

```
st_assvg
```

```
-----
```

```
M 5 -5 L 3 -5 3 -3 0 -3 M 0 0 A 2 2 0 0 0 2 -2
```

### Multi-Surface

```
SELECT ST_AsSVG('MULTISURFACE(  
CURVEPOLYGON(CIRCULARSTRING(-2 0,-1 -1,0 0,1 -1,2 0,0 2,-2 0),  
(-1 0,0 0.5,1 0,0 1,-1 0)),  
((7 8,10 10,6 14,4 11,7 8)))'::geometry, 0, 2);
```

```
st_assvg
```

```
-----
```

```
M -2 0 A 1 1 0 0 0 0 0 A 1 1 0 0 0 2 0 A 2 2 0 0 0 -2 0 Z
```

```
M -1 0 L 0 -0.5 1 0 0 -1 -1 0 Z
```

```
M 7 -8 L 10 -10 6 -14 4 -11 Z
```

### 7.9.3.12 ST\_AsTWKB

`ST_AsTWKB` — Gibt die Geometrie als TWKB, aka "Tiny Well-known Binary" zurück

## Synopsis

bytea **ST\_AsTWKB**(geometry geom, integer prec=0, integer prec\_z=0, integer prec\_m=0, boolean with\_sizes=false, boolean with\_boxes=false);  
 bytea **ST\_AsTWKB**(geometry[] geom, bigint[] ids, integer prec=0, integer prec\_z=0, integer prec\_m=0, boolean with\_sizes=false, boolean with\_boxes=false);

## Beschreibung

Gibt die Geometrie im TWKB ("Tiny Well-Known Binary") Format aus. TWKB ist ein **komprimiertes binäres Format** mit dem Schwerpunkt, die Ausgabegröße zu minimieren.

Der Parameter 'decimaldigits' bestimmt die Anzahl der Dezimalstellen bei der Ausgabe. Standardmäßig werden die Werte vor der Zeichenkodierung auf die 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>.

Erweiterung: 2.4.0 Hauptspeicher- und Geschwindigkeitsverbesserungen.

Verfügbarkeit: 2.2.0

## Beispiele

```
SELECT ST_AsTWKB('LINESTRING(1 1,5 5)')::geometry);
          st_astwkb
-----
\x020000202020808
```

Um ein aggregiertes TWKB-Objekt inklusive Identifikatoren zu erzeugen, fassen Sie bitte die gewünschte Geometrie und Objekte zuerst mittels "array\_agg()" zusammen und rufen anschließend die passende TWKB Funktion auf.

```
SELECT ST_AsTWKB(array_agg(geom), array_agg(gid)) FROM mytable;
          st_astwkb
-----
\x040402020400000202
```

## Siehe auch

[ST\\_GeomFromTWKB](#), [ST\\_AsBinary](#), [ST\\_AsEWKB](#), [ST\\_AsEWKT](#), [ST\\_GeomFromText](#)

### 7.9.3.13 ST\_AsX3D

**ST\_AsX3D** — Gibt eine Geometrie im X3D XML Knotenelement-Format zurück: ISO-IEC-19776-1.2-X3DEncodings-XML

**Synopsis**


text `ST_AsX3D(geometry g1, integer maxdecimaldigits=15, integer options=0);`

**Beschreibung**

Gibt eine Geometrie als X3D knotenformatiertes XML Element zurück <http://www.web3d.org/standards/number/19776-1>. Falls `maxdecimaldigits` (Genauigkeit) nicht angegeben ist, wird sie standardmäßig 15.

---

**Note**



Es gibt verschiedene Möglichkeiten eine PostGIS Geometrie in X3D zu übersetzen, da sich der X3D Geometriotyp 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="WE" "latitude_first"` verwenden Sie bitte  $(2+1) = 3$

PostGIS Datentyp	2D X3D Datentyp	3D X3D Datentyp
LINestring	zurzeit nicht implementiert - wird PolyLine2D	LineSet
MULTILINestring	zurzeit nicht implementiert - wird PolyLine2D	IndexedLineSet
MULTIPOINT	Polypoint2D	PointSet
POINT	gibt leerzeichengetrennte Koordinaten aus	gibt leerzeichengetrennte Koordinaten aus
(MULTI) POLYGON, POLYHEDRALSURFACE	Ungültiges X3D Markup	IndexedFaceSet (die inneren Ringe werden zurzeit als ein weiteres FaceSet abgebildet)
TIN	TriangleSet2D (zurzeit nicht implementiert)	IndexedTriangleSet

---

**Note**



Die Unterstützung von 2D-Geometrie ist noch nicht vollständig. Die inneren Ringe werden zur Zeit lediglich als gesonderte Polygone abgebildet. Wir arbeiten daran.

---

Viele Fortschritte im 3D-Bereich, insbesondere mit [X3D-Integration mit HTML5](#)

---

Es gibt auch einen feinen OpenSource X3D Viewer, den Sie benützen können, um Geometrien darzustellen. Free Wrl <http://freewrl.sourceforge.net> Binärdateien sind für Mac, Linux und Windows verfügbar. Sie können den mitgelieferten FreeWRL\_Launcher verwenden, um Geometrien darzustellen.

Schauen Sie sich auch [PostGIS minimalistischer X3D-Viewer](#) an, der diese Funktion nutzt, und [x3dDom html/js open source toolkit](#).

Verfügbarkeit: 2.0.0: ISO-IEC-19776-1.2-X3DEncodings-XML

Erweiterung: 2.2.0: Unterstützung für geographische Koordinaten und Vertauschen der Achsen (x/y, Länge/Breite). Für nähere Details siehe Optionen.



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



Diese Funktion unterstützt polyedrische Flächen.



Diese Funktion unterstützt Dreiecke und dreieckige unregelmäßige Netzoberflächen (TIN).

**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>
>' 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>

```

## PostGIS-Gebäude

Kopieren Sie die Ausgabe dieser Abfrage und fügen Sie sie in **x3d scene viewer** ein und klicken Sie auf Anzeigen

```

SELECT string_agg('<Shape
>' || ST_AsX3D(ST_Extrude(geom, 0,0, i*0.5)) ||
  '<Appearance>
    <Material diffuseColor="' || (0.01*i)::text || ' 0.8 0.2" specularColor="' || ↔
    (0.05*i)::text || ' 0 0.5"/>
  </Appearance>
</Shape
>', '')
FROM ST_Subdivide(ST_Letters('PostGIS'),20) WITH ORDINALITY AS f(geom,i);

```



*Gebäude, die durch Unterteilung von PostGIS und Extrusion gebildet werden*

### Beispiel: Ein Achteck, um 3 Einheiten gehoben und mit einer dezimalen Genauigkeit von 6

```

SELECT ST_AsX3D(
ST_Translate(
  ST_Force_3d(
    ST_Buffer(ST_Point(10,10),5, 'quad_segs=2')), 0,0,
    3)
,6) As x3dfrag;

x3dfrag
-----
<IndexedFaceSet coordIndex="0 1 2 3 4 5 6 7">
  <Coordinate point="15 10 3 13.535534 6.464466 3 10 5 3 6.464466 6.464466 3 5 10 3 ↔
    6.464466 13.535534 3 10 15 3 13.535534 13.535534 3 " />
</IndexedFaceSet>

```

### Beispiel: TIN

```

SELECT ST_AsX3D(ST_GeomFromEWKT('TIN (((
    0 0 0,
    0 0 1,
    0 1 0,
    0 0 0
  )), ((
    0 0 0,

```

```

        0 1 0,
        1 1 0,
        0 0 0
    ))
    )')) As x3dfrag;

    x3dfrag
    -----
<IndexedTriangleSet index='0 1 2 3 4 5'
><Coordinate point='0 0 0 0 0 1 0 1 0 0 0 0 0 1 0 1 1 0' /></IndexedTriangleSet>

```

### Beispiel: Geschlossener MultiLinestring (die Begrenzung eines Polygons mit Lücken)

```

SELECT ST_AsX3D(
    ST_GeomFromEWKT('MULTILINESTRING((20 0 10,16 -12 10,0 -16 10,-12 -12 ↔
        10,-20 0 10,-12 16 10,0 24 10,16 16 10,20 0 10),
    (12 0 10,8 8 10,0 12 10,-8 8 10,-8 0 10,-8 -4 10,0 -8 10,8 -4 10,12 0 10))')
) As x3dfrag;

    x3dfrag
    -----
<IndexedLineSet coordIndex='0 1 2 3 4 5 6 7 0 -1 8 9 10 11 12 13 14 15 8'>
  <Coordinate point='20 0 10 16 -12 10 0 -16 10 -12 -12 10 -20 0 10 -12 16 10 0 24 10 16 ↔
    16 10 12 0 10 8 8 10 0 12 10 -8 8 10 -8 0 10 -8 -4 10 0 -8 10 8 -4 10 ' />
</IndexedLineSet>

```

#### 7.9.3.14 ST\_GeoHash

ST\_GeoHash — Gibt die Geometrie in der GeoHash Darstellung aus.

##### Synopsis

text **ST\_GeoHash**(geometry geom, integer maxchars=full\_precision\_of\_point);

##### Beschreibung

Berechnet eine **GeoHash**-Darstellung einer Geometrie. Ein GeoHash kodiert einen geografischen Punkt in eine Textform, die auf der Grundlage von Präfixen sortierbar und durchsuchbar ist. Ein kürzerer GeoHash ist eine weniger präzise Darstellung eines Punktes. Man kann ihn sich als eine Box vorstellen, die den Punkt enthält.

Auch Nicht-Punkt-Geometriewerte mit einer Ausdehnung ungleich Null können auf GeoHash-Codes abgebildet werden. Die Genauigkeit des Codes hängt von der geografischen Ausdehnung der Geometrie ab.

Wenn `maxchars` nicht angegeben wird, ist der zurückgegebene GeoHash-Code für die kleinste Zelle, die die Eingabegeometrie enthält. Punkte geben einen GeoHash mit einer Genauigkeit von 20 Zeichen zurück (ungefähr genug, um die volle doppelte Genauigkeit der Eingabe zu speichern). Andere Geometrietyper können je nach Umfang der Geometrie einen GeoHash mit geringerer Genauigkeit zurückgeben. Größere Geometrien werden mit geringerer, kleinere mit höherer Genauigkeit dargestellt. Der durch den GeoHash-Code ermittelte Kasten enthält immer das eingegebene Merkmal.

Wenn `maxchars` angegeben ist, hat der zurückgegebene GeoHash-Code höchstens so viele Zeichen. Er entspricht einer (möglicherweise) weniger genauen Darstellung der Eingabegeometrie. Bei Nicht-Punkten ist der Ausgangspunkt der Berechnung der Mittelpunkt des Begrenzungsrahmens der Geometrie.

Verfügbarkeit: 1.4.0

**Note**

ST\_GeoHash setzt voraus, dass die Eingabegeometrie in geografischen (Lon/Lat) Koordinaten vorliegt.



Diese Methode unterstützt kreisförmige Strings und Kurven.

**Beispiele**

```
SELECT ST_GeoHash( ST_Point(-126,48) );

      st_geohash
-----
c0w3hf1s70w3hf1s70w3

SELECT ST_GeoHash( ST_Point(-126,48), 5);

      st_geohash
-----
c0w3h

-- This line contains the point, so the GeoHash is a prefix of the point code
SELECT ST_GeoHash('LINESTRING(-126 48, -126.1 48.1)::geometry);

      st_geohash
-----
c0w3
```

**Siehe auch**

[ST\\_GeomFromGeoHash](#), [ST\\_PointFromGeoHash](#), [ST\\_Box2dFromGeoHash](#)

## 7.10 Operatoren

### 7.10.1 Bounding-Box-Operatoren

#### 7.10.1.1 &&

&& — Gibt TRUE zurück, wenn die 2D Bounding Box von A die 2D Bounding Box von B schneidet.

**Synopsis**

```
boolean &&( geometry A , geometry B );
boolean &&( geography A , geography B );
```

**Beschreibung**

Der && Operator gibt TRUE zurück, wenn die 2D Bounding Box von Geometrie A die 2D Bounding Box der Geometrie von B schneidet.



**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



Diese Methode unterstützt kreisförmige Strings und Kurven.



Diese Funktion unterstützt polyedrische Flächen.

**Beispiele**

```
SELECT tbl1.column1, tbl2.column1, tbl1.column2 && tbl2.column2 AS overlaps
FROM ( VALUES
      (1, 'LINESTRING(0 0, 3 3)::geometry),
      (2, 'LINESTRING(0 1, 0 5)::geometry)) AS tbl1,
 ( VALUES
      (3, 'LINESTRING(1 2, 4 6)::geometry)) AS tbl2;
```

column1	column1	overlaps
1	3	t
2	3	f

(2 rows)

**Siehe auch**

[ST\\_Intersects](#), [&>](#), [&<|](#), [&<](#), [~](#), [@](#)

**7.10.1.2 &&(geometry,box2df)**

`&&(geometry,box2df)` — Gibt TRUE zurück, wenn sich die 2D Bounding Box (cached) einer Geometrie mit einer 2D Bounding Box mit Gleitpunktgenauigkeit (BOX2DF) überschneidet.

**Synopsis**

boolean `&&( geometry A , box2df B );`

**Beschreibung**

Der `&&` Operator gibt TRUE zurück, wenn die im Cache befindliche 2D Bounding Box der Geometrie A sich mit der 2D Bounding Box von B, unter Verwendung von Gleitpunktgenauigkeit überschneidet. D.h.: falls B eine (double precision) box2d ist, wird diese intern in eine auf Gleitpunkt genaue 2D Bounding Box (BOX2DF) umgewandelt.

**Note**

Dieser Operand ist eher für die interne Nutzung durch BRIN Indizes, als durch die Anwender, gedacht.

Verfügbarkeit: Mit 2.3.0 wurde die Unterstützung von Block Range INdices (BRIN) eingeführt. Erfordert PostgreSQL 9.5+.



Diese Methode unterstützt kreisförmige Strings und Kurven.



Diese Funktion unterstützt polyedrische Flächen.

## Beispiele

```
SELECT ST_Point(1,1) && ST_MakeBox2D(ST_Point(0,0), ST_Point(2,2)) AS overlaps;
```

```
overlaps
-----
t
(1 row)
```

## Siehe auch

[&&\(box2df,geometry\)](#), [&&\(box2df,box2df\)](#), [~\(geometry,box2df\)](#), [~\(box2df,geometry\)](#), [~\(box2df,box2df\)](#), [@\(geometry,box2df\)](#), [@\(box2df,geometry\)](#), [@\(box2df,box2df\)](#)

### 7.10.1.3 &&(box2df,geometry)

[&&\(box2df,geometry\)](#) — Gibt TRUE zurück, wenn eine 2D float precision bounding box (BOX2DF) eine Geometrie (cached) 2D bounding box schneidet.

## Synopsis

boolean [&&](#)( box2df A , geometry B );

## Beschreibung

Der [&&](#) Operator gibt TRUE zurück, wenn die 2D Bounding Box A die zwischengespeicherte 2D Bounding Box der Geometrie B, unter Benutzung von Fließpunktgenauigkeit, schneidet. D.h.: wenn A eine (double precision) box2d ist, wird diese intern in eine float precision 2D bounding box (BOX2DF) umgewandelt.



### Note

Dieser Operand ist eher für die interne Nutzung durch BRIN Indizes, als durch die Anwender, gedacht.

Verfügbarkeit: Mit 2.3.0 wurde die Unterstützung von Block Range INdices (BRIN) eingeführt. Erfordert PostgreSQL 9.5+.



Diese Methode unterstützt kreisförmige Strings und Kurven.



Diese Funktion unterstützt polyedrische Flächen.

## Beispiele

```
SELECT ST_MakeBox2D(ST_Point(0,0), ST_Point(2,2)) && ST_Point(1,1) AS overlaps;
```

```
overlaps
-----
t
(1 row)
```

## Siehe auch

[&&\(geometry,box2df\)](#), [&&\(box2df,box2df\)](#), [~\(geometry,box2df\)](#), [~\(box2df,geometry\)](#), [~\(box2df,box2df\)](#), [@\(geometry,box2df\)](#), [@\(box2df,geometry\)](#), [@\(box2df,box2df\)](#)

### 7.10.1.4 &&(box2df,box2df)

`&&(box2df,box2df)` — Gibt `TRUE` zurück, wenn sich zwei 2D float precision Bounding Boxes (`BOX2DF`) überschneiden.

#### Synopsis

boolean `&&( box2df A , box2df B )`;

#### Beschreibung

Der `&&` Operator gibt `TRUE` zurück, wenn sich zwei 2D Bounding Boxes A und B, unter Benutzung von float precision, gegenseitig überschneiden. D.h.: Wenn A (oder B) eine (double precision) `box2d` ist, wird diese intern in eine float precision 2D bounding box (`BOX2DF`) umgewandelt



#### Note

Dieser Operator ist für die interne Nutzung durch BRIN Indizes, und nicht so sehr durch Anwender, vorgesehen.

Verfügbarkeit: Mit 2.3.0 wurde die Unterstützung von Block Range INdices (BRIN) eingeführt. Erfordert PostgreSQL 9.5+.



Diese Methode unterstützt kreisförmige Strings und Kurven.



Diese Funktion unterstützt polyedrische Flächen.

#### Beispiele

```
SELECT ST_MakeBox2D(ST_Point(0,0), ST_Point(2,2)) && ST_MakeBox2D(ST_Point(1,1), ST_Point(3,3)) AS overlaps;
```

```
overlaps
-----
t
(1 row)
```

#### Siehe auch

[&&\(geometry,box2df\)](#), [&&\(box2df,geometry\)](#), [~\(geometry,box2df\)](#), [~\(box2df,geometry\)](#), [~\(box2df,box2df\)](#), [@\(geometry,box2df\)](#), [@\(box2df,geometry\)](#), [@\(box2df,box2df\)](#)

### 7.10.1.5 &&&

`&&&` — Gibt `TRUE` zurück, wenn A's n-D bounding box B's n-D bounding box schneidet.

#### Synopsis

boolean `&&&( geometry A , geometry B )`;

## Beschreibung

Der `&&&` Operator gibt TRUE zurück, wenn die n-D bounding box der Geometrie A die n-D bounding box der Geometrie B schneidet.



### Note

Dieser Operand benötigt jeden Index, der für die Geometrie zur Verfügung stellt.

Verfügbarkeit: 2.0.0



Diese Methode unterstützt kreisförmige Strings und Kurven.



Diese Funktion unterstützt polyedrische Flächen.



Diese Funktion unterstützt Dreiecke und dreieckige unregelmäßige Netzoberflächen (TIN).



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

## Beispiele: 3D LineStrings

```
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

[&&](#)

### 7.10.1.6 &&&(geometry,gidx)

`&&&(geometry,gidx)` — Gibt `TRUE` zurück, wenn die (cached) n-D bounding box einer Geometrie eine n-D float precision bounding box (GIDX) schneidet.

#### Synopsis

```
boolean &&&( geometry A , gidx B );
```

#### Beschreibung

Der `&&&` Operator gibt `TRUE` zurück, wenn die zwischengespeicherte n-D bounding box der Geometrie A die n-D bounding box B, unter Benutzung von float precision, schneidet. D.h.: Wenn B eine (double precision) box3d ist, wird diese intern in eine float precision 3D bounding box (GIDX) umgewandelt



#### Note

Dieser Operator ist für die interne Nutzung durch BRIN Indizes, und nicht so sehr durch Anwender, vorgesehen.

Verfügbarkeit: Mit 2.3.0 wurde die Unterstützung von Block Range INdices (BRIN) eingeführt. Erfordert PostgreSQL 9.5+.



Diese Methode unterstützt kreisförmige Strings und Kurven.



Diese Funktion unterstützt polyedrische Flächen.



Diese Funktion unterstützt Dreiecke und dreieckige unregelmäßige Netzoberflächen (TIN).



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

#### Beispiele

```
SELECT ST_MakePoint(1,1,1) &&& ST_3DMakeBox(ST_MakePoint(0,0,0), ST_MakePoint(2,2,2)) AS overlaps;

overlaps
-----
t
(1 row)
```

#### Siehe auch

[&&&\(gidx,geometry\)](#), [&&&\(gidx,gidx\)](#)

### 7.10.1.7 &&&(gidx,geometry)

`&&&(gidx,geometry)` — Gibt `TRUE` zurück, wenn eine n-D float precision bounding box (GIDX) eine (cached) n-D bounding box einer Geometrie schneidet.

#### Synopsis

```
boolean &&&( gidx A , geometry B );
```

## Beschreibung

Der `&&&` Operator gibt `TRUE` zurück, wenn die n-D bounding box A die cached n-D bounding box der Geometrie B, unter Benutzung von float precision, schneidet. D.h.: wenn A eine (double precision) box3d ist, 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+.



Diese Methode unterstützt kreisförmige Strings und Kurven.



Diese Funktion unterstützt polyedrische Flächen.



Diese Funktion unterstützt Dreiecke und dreieckige unregelmäßige Netzoberflächen (TIN).



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

## Beispiele

```
SELECT ST_3DMakeBox(ST_MakePoint(0,0,0), ST_MakePoint(2,2,2)) &&& ST_MakePoint(1,1,1) AS overlaps;
```

```
overlaps
-----
t
(1 row)
```

## Siehe auch

[&&&\(geometry,gidx\), &&&\(gidx,gidx\)](#)

### 7.10.1.8 &&&(gidx,gidx)

`&&&(gidx,gidx)` — Gibt `TRUE` zurück, wenn sich zwei n-D float precision bounding boxes (GIDX) gegenseitig überschneiden.

## Synopsis

boolean `&&&( gidx A , gidx B );`

## Beschreibung

Der `&&&` Operator gibt `TRUE` zurück, wenn sich zwei n-D bounding boxes A und B, unter Benutzung von float precision, gegenseitig überschneiden. D.h.: wenn A (oder B) eine (double precision) box3d ist, wird diese intern in eine float precision 3D bounding box (GIDX) umgewandelt



### Note

Dieser Operator ist für die interne Nutzung durch BRIN Indizes, und nicht so sehr durch Anwender, vorgesehen.

Verfügbarkeit: Mit 2.3.0 wurde die Unterstützung von Block Range INdices (BRIN) eingeführt. Erfordert PostgreSQL 9.5+.



Diese Methode unterstützt kreisförmige Strings und Kurven.



Diese Funktion unterstützt polyedrische Flächen.



Diese Funktion unterstützt Dreiecke und dreieckige unregelmäßige Netzoberflächen (TIN).



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

### Beispiele

```
SELECT ST_3DMakeBox(ST_MakePoint(0,0,0), ST_MakePoint(2,2,2)) &&& ST_3DMakeBox(ST_MakePoint(1,1,1), ST_MakePoint(3,3,3)) AS overlaps;
```

```
overlaps
-----
t
(1 row)
```

### Siehe auch

[&&&\(geometry,gidx\), &&&\(gidx,geometry\)](#)

### 7.10.1.9 &<

**&<** — Gibt TRUE zurück, wenn die bounding box der Geometrie A, die bounding box der Geometrie B überlagert oder links davon liegt.

### Synopsis

boolean **&<**( geometry A , geometry B );

### Beschreibung

Der **&<** Operator gibt TRUE zurück, wenn die bounding box der Geometrie A die bounding box der Geometrie B überlagert oder links davon liegt, oder präziser, überlagert und NICHT rechts von der bounding box der Geometrie B liegt.



#### Note

Dieser Operand benützt jeden Index, der für die Geometrie zur Verfügung stellt.

### Beispiele

```
SELECT tbl1.column1, tbl2.column1, tbl1.column2 &< tbl2.column2 AS overleft
FROM
  ( VALUES
    (1, 'LINESTRING(1 2, 4 6)::geometry) AS tbl1,
  ( VALUES
    (2, 'LINESTRING(0 0, 3 3)::geometry),
    (3, 'LINESTRING(0 1, 0 5)::geometry),
    (4, 'LINESTRING(6 0, 6 1)::geometry) AS tbl2;
```

```

column1 | column1 | overleft
-----+-----+-----
          1 |          2 | f
          1 |          3 | f
          1 |          4 | t
(3 rows)

```

## Siehe auch

[&&](#), [|&>](#), [&>](#), [&<](#)

### 7.10.1.10 &<|

**&<|** — Gibt TRUE zurück, wenn die bounding box von A jene von B überlagert oder unterhalb liegt.

## Synopsis

boolean **&<|**( geometry A , geometry B );

## Beschreibung

Der **&<|** Operator gibt TRUE zurück, wenn die Bounding Box der Geometrie A die Bounding Box der Geometrie B überlagert oder unterhalb liegt, oder präziser, überlagert oder NICHT oberhalb der Bounding der Geometrie B liegt.



Diese Methode unterstützt kreisförmige Strings und Kurven.



Diese Funktion unterstützt polyedrische Flächen.



## Note

Dieser Operand benützt jeden Index, der für die Geometrie zur Verfügung stellt.

## Beispiele

```

SELECT tbl1.column1, tbl2.column1, tbl1.column2 &<| tbl2.column2 AS overbelow
FROM
  ( VALUES
    (1, 'LINESTRING(6 0, 6 4)::geometry) AS tbl1,
  ( VALUES
    (2, 'LINESTRING(0 0, 3 3)::geometry),
    (3, 'LINESTRING(0 1, 0 5)::geometry),
    (4, 'LINESTRING(1 2, 4 6)::geometry) AS tbl2;

```

```

column1 | column1 | overbelow
-----+-----+-----
          1 |          2 | f
          1 |          3 | t
          1 |          4 | t
(3 rows)

```



**Siehe auch**[&&](#), [|&>](#), [&>](#), [&<](#)**7.10.1.11 &>**

`&>` — Gibt TRUE zurück, wenn die Bounding Box von A jene von B überlagert oder rechts davon liegt.

**Synopsis**

```
boolean &>( geometry A , geometry B );
```

**Beschreibung**

Der `&>` Operator gibt TRUE zurück, wenn die Bounding Box der Geometrie A die Bounding Box der Geometrie B überlagert oder rechts von ihr liegt, oder präziser, überlagert und NICHT links von der Bounding Box der Geometrie B liegt.

**Note**

Dieser Operand benützt jeden Index, der für die Geometrie zur Verfügung stellt.

**Beispiele**

```
SELECT tbl1.column1, tbl2.column1, tbl1.column2 &
> tbl2.column2 AS overright
FROM
  ( VALUES
    (1, 'LINESTRING(1 2, 4 6)::geometry) AS tbl1,
  ( VALUES
    (2, 'LINESTRING(0 0, 3 3)::geometry),
    (3, 'LINESTRING(0 1, 0 5)::geometry),
    (4, 'LINESTRING(6 0, 6 1)::geometry) AS tbl2;
```

column1	column1	overright
1	2	t
1	3	t
1	4	f

(3 rows)

**Siehe auch**[&&](#), [|&>](#), [&<](#), [&<](#)**7.10.1.12 <<**

`<<` — Gibt TRUE zurück, wenn die Bounding Box von A zur Gänze links von der von B liegt.

**Synopsis**

```
boolean <<( geometry A , geometry B );
```

## Beschreibung

Der << Operator gibt TRUE zurück, wenn die Bounding Box der Geometrie A zur Gänze links der Bounding Box der Geometrie B liegt.



### Note

Dieser Operand benützt jeden Index, der für die Geometrie zur Verfügung stellt.

## Beispiele

```
SELECT tbl1.column1, tbl2.column1, tbl1.column2 << tbl2.column2 AS left
FROM
  ( VALUES
    (1, 'LINESTRING (1 2, 1 5)::geometry)) AS tbl1,
  ( VALUES
    (2, 'LINESTRING (0 0, 4 3)::geometry),
    (3, 'LINESTRING (6 0, 6 5)::geometry),
    (4, 'LINESTRING (2 2, 5 6)::geometry)) AS tbl2;
```

column1	column1	left
1	2	f
1	3	t
1	4	t

(3 rows)

## Siehe auch

>>, |>>, <<|

### 7.10.1.13 <<|

<<| — Gibt TRUE zurück, wenn A's Bounding Box zur Gänze unterhalb von der von B liegt.

## Synopsis

```
boolean <<|( geometry A , geometry B );
```

## Beschreibung

Der <<| Operator gibt TRUE zurück, wenn die Bounding Box der Geometrie A zur Gänze unterhalb der Bounding Box von Geometrie B liegt.



### Note

Dieser Operand benützt jeden Index, der für die Geometrie zur Verfügung stellt.

## Beispiele

```
SELECT tbl1.column1, tbl2.column1, tbl1.column2 <<| tbl2.column2 AS below
FROM
  ( VALUES
    (1, 'LINESTRING (0 0, 4 3)::geometry) AS tbl1,
  ( VALUES
    (2, 'LINESTRING (1 4, 1 7)::geometry),
    (3, 'LINESTRING (6 1, 6 5)::geometry),
    (4, 'LINESTRING (2 3, 5 6)::geometry) AS tbl2;
```

```
column1 | column1 | below
-----+-----+-----
        1 |         2 | t
        1 |         3 | f
        1 |         4 | f
(3 rows)
```

## Siehe auch

<<, >>, |>>

### 7.10.1.14 =

= — Gibt TRUE zurück, wenn die Koordinaten und die Reihenfolge der Koordinaten der Geometrie/Geographie A und der Geometrie/Geographie B ident sind.

## Synopsis

```
boolean =( geometry A , geometry B );
boolean =( geography A , geography B );
```

## Beschreibung

Der Operator = gibt TRUE zurück, wenn die Koordinaten und die Reihenfolge der Koordinaten der Geometrie/Geographie A und der Geometrie/Geographie B ident sind. PostgreSQL verwendet die =, <, und > Operatoren um die interne Sortierung und den Vergleich von Geometrien durchzuführen (z.B.: in einer GROUP BY oder ORDER BY Klausel).

### Note



Nur die Geometrie/Geographie die in allen Gesichtspunkten übereinstimmt, d.h. mit den selben Koordinaten in der gleichen Reihenfolge, werden von diesem Operator als gleich betrachtet. Für "räumliche Gleichheit", bei der Dinge wie die Reihenfolge der Koordinaten außer Acht gelassen werden, und die es ermöglicht Geobjekte zu erfassen, die denselben räumlichen Bereich mit unterschiedlicher Darstellung abdecken, verwenden Sie bitte [ST\\_OrderingEquals](#) oder [ST\\_Equals](#)



### Caution

Dieser Operator verwendet NICHT die Indizes, welche für die Geometrien vorhanden sind. Um eine Überprüfung auf exakte Gleichheit indexgestützt durchzuführen, kombinieren Sie bitte = mit &&.

Änderung: 2.4.0, in Vorgängerversionen war dies die Gleichheit der umschreibenden Rechtecke, nicht die geometrische Gleichheit. Falls Sie auf Gleichheit der umschreibenden Rechtecke prüfen wollen, verwenden Sie stattdesse bitte `~=`.

✓ Diese Methode unterstützt kreisförmige Strings und Kurven.

✓ Diese Funktion unterstützt polyedrische Flächen.

## Beispiele

```
SELECT 'LINESTRING(0 0, 0 1, 1 0)::geometry = 'LINESTRING(1 1, 0 0)::geometry;
?column?
-----
f
(1 row)

SELECT ST_AsText(column1)
FROM ( VALUES
      ('LINESTRING(0 0, 1 1)::geometry),
      ('LINESTRING(1 1, 0 0)::geometry)) AS foo;
      st_astext
-----
LINESTRING(0 0,1 1)
LINESTRING(1 1,0 0)
(2 rows)

-- Note: the GROUP BY uses the "=" to compare for geometry equivalency.
SELECT ST_AsText(column1)
FROM ( VALUES
      ('LINESTRING(0 0, 1 1)::geometry),
      ('LINESTRING(1 1, 0 0)::geometry)) AS foo
GROUP BY column1;
      st_astext
-----
LINESTRING(0 0,1 1)
LINESTRING(1 1,0 0)
(2 rows)

-- In versions prior to 2.0, this used to return true --
SELECT ST_GeomFromText('POINT(1707296.37 4820536.77)') =
      ST_GeomFromText('POINT(1707296.27 4820536.87)') As pt_intersect;

--pt_intersect --
f
```

## Siehe auch

[ST\\_Equals](#), [ST\\_OrderingEquals](#), [~=](#)

### 7.10.1.15 >>

>> — Gibt TRUE zurück, wenn A's bounding box zur Gänze rechts von der von B liegt.

## Synopsis

boolean >>( geometry A , geometry B );

## Beschreibung

Der >> Operator gibt TRUE zurück, wenn die Bounding Box von Geometrie A zur Gänze rechts der Bounding Box von Geometrie B liegt.



### Note

Dieser Operand benützt jeden Index, der für die Geometrie zur Verfügung stellt.

## Beispiele

```
SELECT tbl1.column1, tbl2.column1, tbl1.column2
>
> tbl2.column2 AS right
FROM
  ( VALUES
    (1, 'LINESTRING (2 3, 5 6)::geometry) AS tbl1,
  ( VALUES
    (2, 'LINESTRING (1 4, 1 7)::geometry),
    (3, 'LINESTRING (6 1, 6 5)::geometry),
    (4, 'LINESTRING (0 0, 4 3)::geometry) AS tbl2;
```

```
column1 | column1 | right
-----+-----+-----
         1 |         2 | t
         1 |         3 | f
         1 |         4 | f
(3 rows)
```

## Siehe auch

<<, >>, <<|

### 7.10.1.16 @

@ — Gibt TRUE zurück, wenn die Bounding Box von A in jener von B enthalten ist.

## Synopsis

```
boolean @( geometry A , geometry B );
```

## Beschreibung

Der @ Operator gibt TRUE zurück, wenn die Bounding Box der Geometrie A 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

[~, &&](#)

### 7.10.1.17 @(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 INdexes (BRIN) eingeführt. Erfordert PostgreSQL 9.5+.



Diese Methode unterstützt kreisförmige Strings und Kurven.



Diese Funktion unterstützt polyedrische Flächen.

## Beispiele

```
SELECT ST_Buffer(ST_GeomFromText('POINT(2 2)'), 1) @ ST_MakeBox2D(ST_Point(0,0), ST_Point(5,5)) AS is_contained;

is_contained
-----
t
(1 row)
```

**Siehe auch**

[&&\(geometry,box2df\)](#), [&&\(box2df,geometry\)](#), [&&\(box2df,box2df\)](#), [~\(geometry,box2df\)](#), [~\(box2df,geometry\)](#), [~\(box2df,box2df\)](#), [@\(box2df,geometry\)](#), [@\(box2df,box2df\)](#)

**7.10.1.18 @(box2df,geometry)**

`@(box2df,geometry)` — Gibt TRUE zurück, wenn eine 2D float precision bounding box (BOX2DF) in der 2D Bounding Box einer Geometrie enthalten ist..

**Synopsis**

```
boolean @( box2df A , geometry B );
```

**Beschreibung**

Der @ Operator gibt TRUE zurück, wenn die 2D bounding box A in der 2D bounding box der Geometrie B, unter Verwendung von float precision, enthalten ist. D.h.: wenn B eine (double precision) box2d ist, wird diese intern in eine float precision 2D bounding box (BOX2DF) umgewandelt

**Note**

Dieser Operand ist eher für die interne Nutzung durch BRIN Indizes, als durch die Anwender, gedacht.

Verfügbarkeit: Mit 2.3.0 wurde die Unterstützung von Block Range INdices (BRIN) eingeführt. Erfordert PostgreSQL 9.5+.



Diese Methode unterstützt kreisförmige Strings und Kurven.



Diese Funktion unterstützt polyedrische Flächen.

**Beispiele**

```
SELECT ST_MakeBox2D(ST_Point(2,2), ST_Point(3,3)) @ ST_Buffer(ST_GeomFromText('POINT(1 1)'), 10) AS is_contained;

is_contained
-----
t
(1 row)
```

**Siehe auch**

[&&\(geometry,box2df\)](#), [&&\(box2df,geometry\)](#), [&&\(box2df,box2df\)](#), [~\(geometry,box2df\)](#), [~\(box2df,geometry\)](#), [~\(box2df,box2df\)](#), [@\(geometry,box2df\)](#), [@\(box2df,box2df\)](#)

### 7.10.1.19 @(box2df,box2df)

@(box2df,box2df) — Gibt TRUE zurück, wenn eine 2D float precision bounding box (BOX2DF) innerhalb einer anderen 2D float precision bounding box enthalten ist.

#### Synopsis

```
boolean @( box2df A , box2df B );
```

#### Beschreibung

Der @ Operator gibt TRUE zurück, wenn die 2D bounding box A innerhalb der 2D bounding box B, unter Verwendung von float precision, enthalten ist. D.h.: wenn A (oder B) eine (double precision) box2d ist, wird diese intern in eine float precision 2D bounding box (BOX2DF) umgewandelt.



#### Note

Dieser Operand ist eher für die interne Nutzung durch BRIN Indizes, als durch die Anwender, gedacht.

Verfügbarkeit: Mit 2.3.0 wurde die Unterstützung von Block Range INdexes (BRIN) eingeführt. Erfordert PostgreSQL 9.5+.



Diese Methode unterstützt kreisförmige Strings und Kurven.



Diese Funktion unterstützt polyedrische Flächen.

#### Beispiele

```
SELECT ST_MakeBox2D(ST_Point(2,2), ST_Point(3,3)) @ ST_MakeBox2D(ST_Point(0,0), ST_Point(5,5)) AS is_contained;
```

```
is_contained
-----
t
(1 row)
```

#### Siehe auch

[&&\(geometry,box2df\)](#), [&&\(box2df,geometry\)](#), [&&\(box2df,box2df\)](#), [~\(geometry,box2df\)](#), [~\(box2df,geometry\)](#), [~\(box2df,box2df\)](#), [@\(geometry,box2df\)](#), [@\(box2df,geometry\)](#)

### 7.10.1.20 |&>

|&> — Gibt TRUE zurück, wenn A's bounding box diejenige von B überlagert oder oberhalb von B liegt.

#### Synopsis

```
boolean |&>( geometry A , geometry B );
```



## Beschreibung

Der `|&>` Operator gibt `TRUE` zurück, wenn die bounding box der Geometrie A die bounding box der Geometrie B überlagert oder oberhalb liegt, oder präziser, überlagert oder NICHT unterhalb der Bounding Box der Geometrie B liegt.



### Note

Dieser Operand benützt jeden Index, der für die Geometrie zur Verfügung stellt.

## Beispiele

```
SELECT tbl1.column1, tbl2.column1, tbl1.column2 |&
> tbl2.column2 AS overabove
FROM
  ( VALUES
    (1, 'LINESTRING(6 0, 6 4)::geometry) AS tbl1,
  ( VALUES
    (2, 'LINESTRING(0 0, 3 3)::geometry),
    (3, 'LINESTRING(0 1, 0 5)::geometry),
    (4, 'LINESTRING(1 2, 4 6)::geometry) AS tbl2;

column1 | column1 | overabove
-----+-----+-----
         1 |         2 | t
         1 |         3 | f
         1 |         4 | f
(3 rows)
```

## Siehe auch

[&&](#), [&>](#), [&<l](#), [&<](#)

### 7.10.1.21 |>>

`|>>` — Gibt `TRUE` zurück, wenn A's bounding box is zur Gänze oberhalb der von B liegt.

## Synopsis

```
boolean |>>( geometry A , geometry B );
```

## Beschreibung

Der Operator `|>>` gibt `TRUE` zurück, wenn die Bounding Box der Geometrie A zur Gänze oberhalb der Bounding Box von Geometrie B liegt.



### Note

Dieser Operand benützt jeden Index, der für die Geometrie zur Verfügung stellt.

## Beispiele

```
SELECT tbl1.column1, tbl2.column1, tbl1.column2 |>> tbl2.column2 AS above
FROM
  ( VALUES
    (1, 'LINESTRING (1 4, 1 7)::geometry) AS tbl1,
  ( VALUES
    (2, 'LINESTRING (0 0, 4 2)::geometry),
    (3, 'LINESTRING (6 1, 6 5)::geometry),
    (4, 'LINESTRING (2 3, 5 6)::geometry) AS tbl2;
```

column1	column1	above
1	2	t
1	3	f
1	4	f

(3 rows)

## Siehe auch

<<, >>, <<|

### 7.10.1.22 ~

~ — Gibt TRUE zurück, wenn A's bounding box die von B enthält.

## Synopsis

boolean ~( geometry A , geometry B );

## Beschreibung

Der ~ Operator gibt TRUE zurück, wenn die bounding box der Geometrie A zur Gänze die bounding box der Geometrie B enthält.



### Note

Dieser Operand benützt jeden Index, der für die Geometrie zur Verfügung stellt.

## Beispiele

```
SELECT tbl1.column1, tbl2.column1, tbl1.column2 ~ tbl2.column2 AS contains
FROM
  ( VALUES
    (1, 'LINESTRING (0 0, 3 3)::geometry) AS tbl1,
  ( VALUES
    (2, 'LINESTRING (0 0, 4 4)::geometry),
    (3, 'LINESTRING (1 1, 2 2)::geometry),
    (4, 'LINESTRING (0 0, 3 3)::geometry) AS tbl2;
```

column1	column1	contains
1	2	f
1	3	t
1	4	t

(3 rows)

**Siehe auch**[@](#), [&&](#)**7.10.1.23 ~([geometry,box2df](#))**

[~\(\[geometry,box2df\]\(#\)\)](#) — Gibt TRUE zurück, wenn die 2D bounding box einer Geometrie eine 2D float precision bounding box (GIDX) enthält.

**Synopsis**

```
boolean ~( geometry A , box2df B );
```

**Beschreibung**

Der ~ Operator gibt TRUE zurück, wenn die 2D bounding box einer Geometrie A die 2D bounding box B, unter Verwendung von float precision, enthält. D.h.: wenn B eine (double precision) [box2d](#) ist, wird diese intern in eine float precision 2D bounding box (BOX2DF) übersetzt

**Note**

Dieser Operand ist eher für die interne Nutzung durch BRIN Indizes, als durch die Anwender, gedacht.

Verfügbarkeit: Mit 2.3.0 wurde die Unterstützung von Block Range INdices (BRIN) eingeführt. Erfordert PostgreSQL 9.5+.



Diese Methode unterstützt kreisförmige Strings und Kurven.



Diese Funktion unterstützt polyedrische Flächen.

**Beispiele**

```
SELECT ST_Buffer(ST_GeomFromText('POINT(1 1)'), 10) ~ ST_MakeBox2D(ST_Point(0,0), ST_Point(↔
(2,2)) AS contains;
```

```
contains
-----
t
(1 row)
```

**Siehe auch**

[&&\(a href="#">geometry,box2df](#)), [&&\(a href="#">box2df,geometry](#)), [&&\(a href="#">box2df,box2df](#)), [~\(a href="#">box2df,geometry](#)), [~\(a href="#">box2df,box2df](#)), [@\(a href="#">geometry,box2df](#)), [@\(a href="#">box2df,geometry](#)), [@\(a href="#">box2df,box2df](#))

**7.10.1.24 ~([box2df,geometry](#))**

[~\(a href="#">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+.



Diese Methode unterstützt kreisförmige Strings und Kurven.



Diese Funktion unterstützt polyedrische Flächen.

## Beispiele

```
SELECT ST_MakeBox2D(ST_Point(0,0), ST_Point(5,5)) ~ ST_Buffer(ST_GeomFromText('POINT(2 2)') ←
, 1) AS contains;
```

```
contains
-----
t
(1 row)
```

## Siehe auch

[&&\(geometry,box2df\)](#), [&&\(box2df,geometry\)](#), [&&\(box2df,box2df\)](#), [~\(geometry,box2df\)](#), [~\(box2df,box2df\)](#), [@\(geometry,box2df\)](#), [@\(box2df,geometry\)](#), [@\(box2df,box2df\)](#)

### 7.10.1.25 ~(box2df,box2df)

`~(box2df,box2df)` — Gibt `TRUE` zurück, wenn eine 2D float precision bounding box (`BOX2DF`) eine andere 2D float precision bounding box (`BOX2DF`) enthält.

## Synopsis

```
boolean ~( box2df A , box2df B );
```

## Beschreibung

Der `~` Operator gibt `TRUE` zurück, wenn die 2D bounding box A die 2D bounding box B, unter Verwendung von float precision, enthält. D.h.: wenn A eine (double precision) `box2d` ist, wird diese intern in eine float precision 2D bounding box (`BOX2DF`) umgewandelt



### Note

Dieser Operand ist eher für die interne Nutzung durch BRIN Indizes, als durch die Anwender, gedacht.

Verfügbarkeit: Mit 2.3.0 wurde die Unterstützung von Block Range INdexes (BRIN) eingeführt. Erfordert PostgreSQL 9.5+.



Diese Methode unterstützt kreisförmige Strings und Kurven.



Diese Funktion unterstützt polyedrische Flächen.

### Beispiele

```
SELECT ST_MakeBox2D(ST_Point(0,0), ST_Point(5,5)) ~ ST_MakeBox2D(ST_Point(2,2), ST_Point(3,3)) AS contains;
```

```
contains
-----
t
(1 row)
```

### Siehe auch

[&&\(geometry,box2df\)](#), [&&\(box2df,geometry\)](#), [&&\(box2df,box2df\)](#), [~\(geometry,box2df\)](#), [~\(box2df,geometry\)](#), [@\(geometry,box2df\)](#), [@\(box2df,geometry\)](#), [@\(box2df,box2df\)](#)

#### 7.10.1.26 ~=

~= — Gibt TRUE zurück, wenn die bounding box von A ident mit jener von B ist.

### Synopsis

boolean ~= ( geometry A , geometry B );

### Beschreibung

Der ~= Operator gibt TRUE zurück, wenn die bounding box der Geometrie/Geographie A ident mit der bounding box der Geometrie/Geographie B ist.



#### Note

Dieser Operand benützt jeden Index, der für die Geometrie zur Verfügung stellt.

Verfügbarkeit: 1.5.0 "Verhaltensänderung"



Diese Funktion unterstützt polyedrische Flächen.



#### Warning

Dieser Operator verhält sich ab PostGIS 1.5 insofern anders, als er vom Prüfen der Übereinstimmung der tatsächlichen Geometrie auf eine ledigliche Überprüfung der Gleichheit der Bounding Boxes abgeändert wurde. Um die Sache noch weiter zu komplizieren, hängt dieses Verhalten der Datenbank davon ab, ob ein hard oder soft upgrade durchgeführt wurde. Um herauszufinden, wie sich die Datenbank in dieser Beziehung verhält, führen Sie bitte die untere Abfrage aus. Um auf exakte Gleichheit zu prüfen benutzen Sie bitte [ST\\_OrderingEquals](#) oder [ST\\_Equals](#).

## Beispiele

```
select 'LINESTRING(0 0, 1 1)::geometry ~=' 'LINESTRING(0 1, 1 0)::geometry as equality;
equality |
-----+
          t |
```

## Siehe auch

[ST\\_Equals](#), [ST\\_OrderingEquals](#), [=](#)

## 7.10.2 Operatoren

### 7.10.2.1 <->

<-> — Gibt die 2D Entfernung zwischen A und B zurück.

## Synopsis

```
double precision <->( geometry A , geometry B );
double precision <->( geography A , geography B );
```

## Beschreibung

Der <-> Operator gibt die 2D Entfernung zwischen zwei Geometrien zurück. Wird er in einer "ORDER BY" Klausel verwendet, so liefert er Index-unterstützte nearest-neighbor Ergebnismengen. PostgreSQL Versionen unter 9.5 geben jedoch lediglich die Entfernung der Centroide der bounding boxes zurück, während PostgreSQL 9.5+ mittels KNN-Methode die tatsächliche Entfernung zwischen den Geometrien, bei geographischen Koordinaten die Entfernung auf der Späre, 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](#), [<#>](#)

### 7.10.2.2 |=|

`|=|` — Gibt die Entfernung zwischen den Trajektorien A und B, am Ort der dichtesten Annäherung, an.

## Synopsis

double precision `|=|( geometry A , geometry B );`

## Beschreibung

Der `|=|` Operator gibt die 3D Entfernung zwischen zwei Trajektorien (Siehe [ST\\_IsValidTrajectory](#)). Dieser entspricht [ST\\_DistanceCPA](#), da es sich jedoch um einen Operator handelt, kann dieser für nearest 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](#)

**7.10.2.3 <#>**

<#> — Gibt die 2D Entfernung zwischen den Bounding Boxes von A und B zurück

**Synopsis**

```
double precision <#>( geometry A , geometry B );
```

**Beschreibung**

Der <#> Operator gibt die Entfernung zwischen zwei floating point bounding boxes zurück, wobei diese eventuell vom räumlichen Index ausgelesen wird (PostgreSQL 9.1+ vorausgesetzt). Praktikabel falls man eine nearest neighbor Abfrage **approximate** nach der Entfernung sortieren will.

**Note**

Dieser Operand verwendet sämtliche Indizes, welche für die Geometrien vorhanden sind. Er unterscheidet sich insofern von anderen Operatoren, welche ebenfalls räumliche Indizes verwenden, als der räumliche Index nur dann verwendet wird, falls sich der Operand in einer ORDER BY Klausel befindet.

**Note**

Der Index kommt nur zum Tragen, wenn eine der Geometrien eine Konstante ist; z.B.: ORDER BY (ST\_GeomFromText('POINT(1 2)') <#> geom) anstatt g1.geom <#>.

Verfügbarkeit: 2.0.0 -- KNN steht erst ab PostgreSQL 9.1+ zur Verfügung

**Beispiele**

```

SELECT *
FROM (
SELECT b.tlrid, b.mtfcc,
       b.geom <#
> ST_GeomFromText ('LINESTRING(746149 2948672,745954 2948576,
                          745787 2948499,745740 2948468,745712 2948438,
                          745690 2948384,745677 2948319)',2249) As b_dist,
       ST_Distance(b.geom, ST_GeomFromText ('LINESTRING(746149 2948672,745954
                          2948576,
                          745787 2948499,745740 2948468,745712 2948438,
                          745690 2948384,745677 2948319)',2249)) As act_dist

```

```
FROM bos_roads As b
ORDER BY b_dist, b.tlid
LIMIT 100) As foo
ORDER BY act_dist, tlid LIMIT 10;
```

tlid	mtfcc	b_dist	act_dist
85732027	S1400	0	0
85732029	S1400	0	0
85732031	S1400	0	0
85734335	S1400	0	0
85736037	S1400	0	0
624683742	S1400	0	128.528874268666
85719343	S1400	260.839270432962	260.839270432962
85741826	S1400	164.759294123275	260.839270432962
85732032	S1400	277.75	311.830282365264
85735592	S1400	222.25	311.830282365264

(10 rows)

## Siehe auch

[ST\\_DWithin](#), [ST\\_Distance](#), [<->](#)

### 7.10.2.4 <<->

<<-> — Gibt den n-D-Abstand zwischen den Geometrien oder Begrenzungsrahmen von A und B zurück

## Synopsis

```
double precision <<->( geometry A , geometry B );
```

## Beschreibung

Der <<-> Operator gibt die n-D (euklidische) Entfernung zwischen den geometrischen Schwerpunkten der Begrenzungsrechtecke zweier Geometrien zurück. Praktikabel für nearest neighbor **approximate** distance ordering.



### Note

Dieser Operator verwendet n-D GiST Indizes, falls diese für die Geometrien vorhanden sind. Er unterscheidet sich insofern von anderen Operatoren, die räumliche Indizes verwenden, indem der räumliche Index nur dann verwendet wird, wenn sich der Operator in einer ORDER BY Klausel befindet.



### Note

Der Index kommt nur zum Tragen, wenn eine der Geometrien eine Konstante ist (sich nicht in einer Subquery/CTE befindet). Z.B. 'SRID=3005;POINT(1011102 450541)::geometry und nicht a.geom

Verfügbarkeit: 2.2.0 -- KNN steht erst ab PostgreSQL 9.1+ zur Verfügung.

## Siehe auch

[<->](#)

## 7.11 Lagevergleiche

### 7.11.1 Topologische Beziehungen

#### 7.11.1.1 ST\_3DIntersects

**ST\_3DIntersects** — Prüft, ob sich zwei Geometrien in 3D räumlich schneiden - nur für Punkte, Linienzüge, Polygone, polyedrische Flächen (Bereich)

#### Synopsis

```
boolean ST_3DIntersects( geometry geomA , geometry geomB );
```

#### Beschreibung

Overlaps, Touches und Within implizieren räumliche Überschneidung. Wenn irgendeine dieser Eigenschaften TRUE zurückgibt, dann überschneiden sich die geometrischen Objekte auch. Disjoint impliziert FALSE für die räumliche Überschneidung.

**Note!**

#### Note

Diese Funktion beinhaltet automatisch einen Bounding-Box-Vergleich, der alle räumlichen Indizes verwendet, die für die Geometrien verfügbar sind.

**Note!**

#### Note

Aufgrund von Floating-Robustness-Fehlern überschneiden sich Geometrien nicht immer so, wie man es nach der geometrischen Bearbeitung erwarten würde. So liegt beispielsweise der Punkt, der einer Geometrie am nächsten liegt, nicht unbedingt auf dem Linienzug. Für diese Art von Problemen, bei denen ein Abstand von einem Zentimeter einfach als Schnittpunkt betrachtet werden soll, verwenden Sie [ST\\_3DDWithin](#).

Änderung: 3.0.0 das SFCGAL Back-end wurde entfernt, das GEOS Back-end unterstützt TIN.

Verfügbarkeit: 2.0.0



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



Diese Funktion unterstützt polyedrische Flächen.



Diese Funktion unterstützt Dreiecke und dreieckige unregelmäßige Netzoberflächen (TIN).



Diese Methode setzt die SQL/MM-Spezifikation um. SQL-MM IEC 13249-3: 5.1

#### Beispiele mit dem geometrischen Datentyp

```
SELECT ST_3DIntersects(pt, line), ST_Intersects(pt, line)
FROM (SELECT 'POINT(0 0 2)::geometry As pt, 'LINESTRING (0 0 1, 0 2 3)::geometry As ←
      line) As foo;
st_3dintersects | st_intersects
-----+-----
f                | t
(1 row)
```

## Beispiele mit dem Datentyp TIN

```
SELECT ST_3DIntersects('TIN(((0 0 0,1 0 0,0 1 0,0 0 0)))'::geometry, 'POINT(.1 .1 0)':: ←
  geometry);
st_3dintersects
-----
t
```

### Siehe auch

[ST\\_3DDWithin](#), [ST\\_Intersects](#)

### 7.11.1.2 ST\_Contains

ST\_Contains — Tests, wenn jeder Punkt von B in A liegt und ihre Innenräume einen gemeinsamen Punkt haben

#### Synopsis

boolean **ST\_Contains**(geometry geomA, geometry geomB);

#### Beschreibung

Gibt TRUE zurück, wenn die Geometrie A die Geometrie B enthält. A enthält B nur dann, wenn alle Punkte von B innerhalb von A liegen (d. h. im Inneren oder am Rand von A) (oder gleichwertig, wenn keine Punkte von B im Äußeren von A liegen) und die Innenräume von A und B mindestens einen Punkt gemeinsam haben.

Mathematisch ausgedrückt:  $ST\_Contains(A, B) \Leftrightarrow (A \cap B = B) \wedge (Int(A) \cap Int(B) \neq \emptyset)$

Die Enthält-Beziehung ist reflexiv: Jede Geometrie enthält sich selbst. (Im Gegensatz dazu enthält im Prädikat **ST\_ContainsProperly** eine Geometrie *nicht* sich selbst.) Die Beziehung ist antisymmetrisch: Wenn  $ST\_Contains(A, B) = true$  und  $ST\_Contains(B, A) = true$ , dann müssen die beiden Geometrien topologisch gleich sein ( $ST\_Equals(A, B) = true$ ).

ST\_Contains ist die Umkehrung von **ST\_Within**.  $ST\_Contains(A, B) = ST\_Within(B, A)$ .



#### Note

Da die Innenräume einen gemeinsamen Punkt haben müssen, besteht eine Feinheit der Definition darin, dass Polygone und Linien *nicht* Linien und Punkte enthalten, die vollständig in ihrer Begrenzung liegen. Für weitere Details siehe [Subtleties of OGC Covers, Contains, Within](#). Das Prädikat **ST\_Covers** bietet eine umfassendere Beziehung.



#### Note

Diese Funktion beinhaltet automatisch einen Bounding-Box-Vergleich, der alle räumlichen Indizes verwendet, die für die Geometrien verfügbar sind. Um die Verwendung eines Indices zu vermeiden, kann die Funktion `_ST_Contains` verwendet werden.

Wird durch das GEOS Modul ausgeführt

Verbessert: 2.3.0 Verbesserung des PIP-Kurzschlusses erweitert um die Unterstützung von MultiPoints mit wenigen Punkten. Frühere Versionen unterstützten nur Punkte in Polygonen.



#### Important

Verbessert: 3.0.0 ermöglicht die Unterstützung von GEOMETRYCOLLECTION

**Important**

Verwenden Sie diese Funktion nicht mit ungültigen Geometrien. Sie werden unerwartete Ergebnisse erhalten.

HINWEIS: Dies ist die "zulässige" Version, die einen booleschen Wert und keine ganze Zahl zurückgibt.




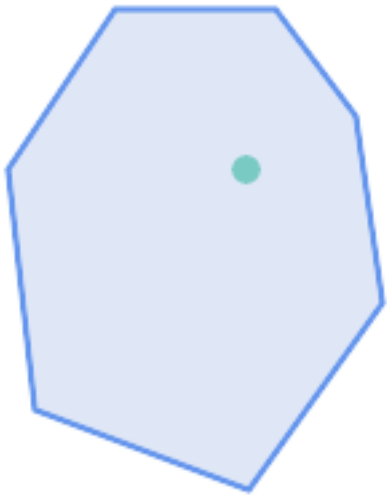
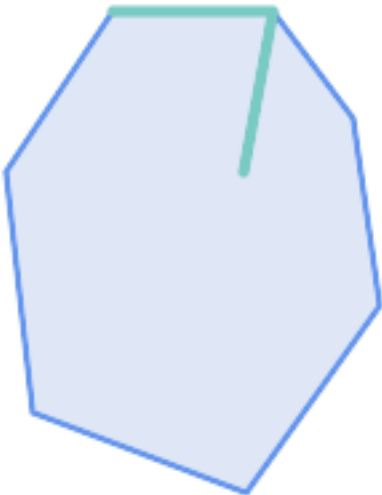
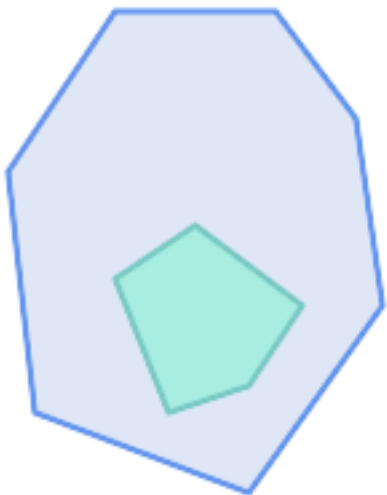
Diese Methode implementiert die **OGC Simple Features Implementation Specification for SQL 1.1**. s2.1.1.2 // s2.1.13.3 - dasselbe wie innerhalb (Geometrie B, Geometrie A)



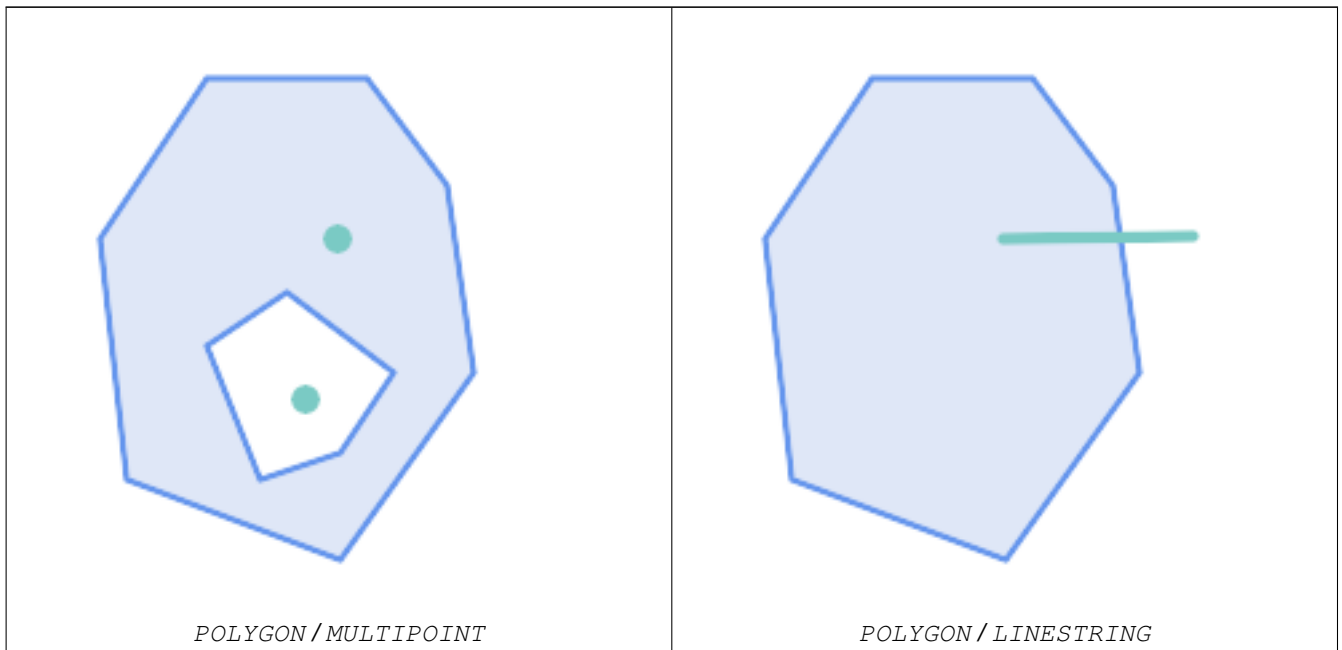
Diese Methode setzt die SQL/MM-Spezifikation um. SQL-MM 3: 5.1.31

**Beispiele**

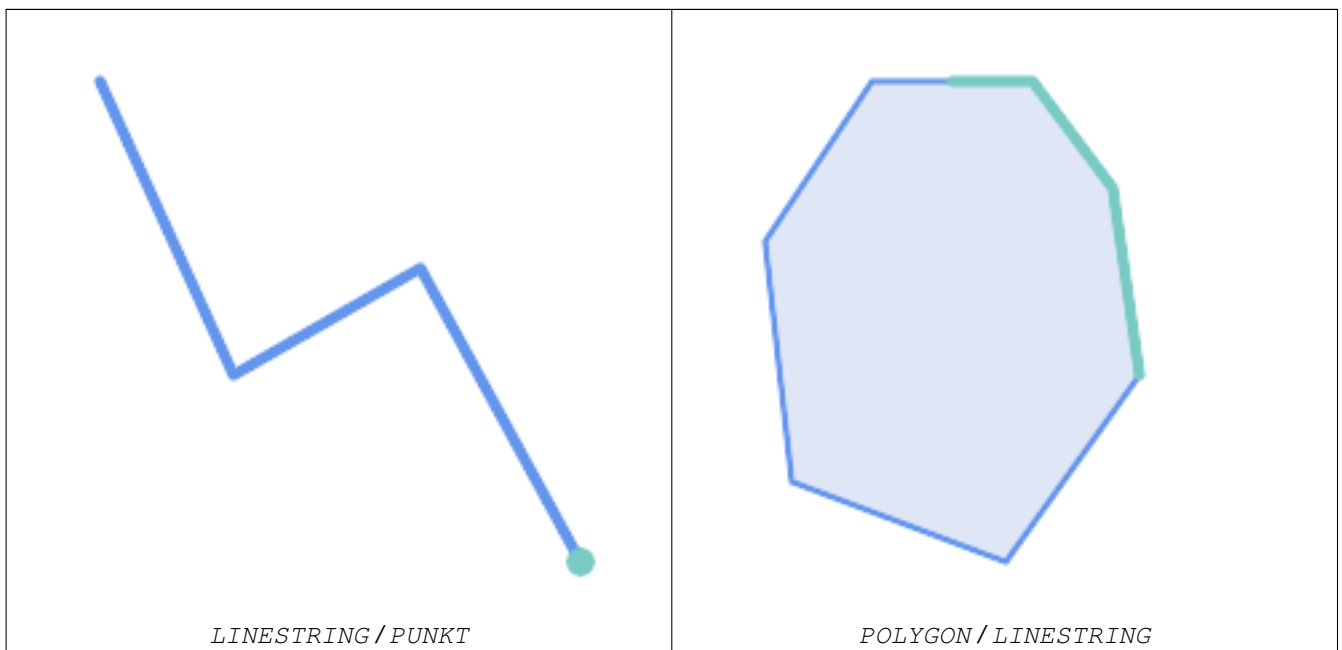
ST\_Contains gibt TRUE in den folgenden Situationen zurück:

 <p style="text-align: center;"><i>LINSTRING / MULTIPOINT</i></p>	 <p style="text-align: center;"><i>POLYGON / PUNKT</i></p>
 <p style="text-align: center;"><i>POLYGON / LINSTRING</i></p>	 <p style="text-align: center;"><i>VIELECK / VIELECK</i></p>

ST\_Contains gibt FALSE in den folgenden Situationen zurück:



Aufgrund der inneren Schnittpunktbedingung ST\_Contains liefert FALSE in den folgenden Situationen (während ST\_Covers TRUE liefert):



```
-- A circle within a circle
SELECT ST_Contains(smallc, bigc) As smallcontainsbig,
       ST_Contains(bigc, smallc) As bigcontainssmall,
       ST_Contains(bigc, ST_Union(smallc, bigc)) as bigcontainsunion,
       ST_Equals(bigc, ST_Union(smallc, bigc)) as bigisunion,
       ST_Covers(bigc, ST_ExteriorRing(bigc)) As bigcoversexterior,
       ST_Contains(bigc, ST_ExteriorRing(bigc)) As bigcontainsexterior
FROM (SELECT ST_Buffer(ST_GeomFromText('POINT(1 2)'), 10) As smallc,
          ST_Buffer(ST_GeomFromText('POINT(1 2)'), 20) As bigc) As foo;
```

```

-- Result
smallcontainsbig | bigcontainssmall | bigcontainsunion | bigisunion | bigcoversexterior | ←
bigcontainsexterior
-----+-----+-----+-----+-----+-----+-----
f              | t              | t              | t              | t              | f

-- Example demonstrating difference between contains and contains properly
SELECT ST_GeometryType(geomA) As geomtype, ST_Contains(geomA,geomA) AS acontainsa, ←
       ST_ContainsProperly(geomA, geomA) AS acontainspropa,
       ST_Contains(geomA, ST_Boundary(geomA)) As acontainsba, ST_ContainsProperly(geomA, ←
       ST_Boundary(geomA)) As acontainspropba
FROM (VALUES ( ST_Buffer(ST_Point(1,1), 5,1) ),
            ( ST_MakeLine(ST_Point(1,1), ST_Point(-1,-1) ) ),
            ( ST_Point(1,1) )
      ) As foo(geomA);

geomtype       | acontainsa | acontainspropa | acontainsba | acontainspropba
-----+-----+-----+-----+-----+-----
ST_Polygon     | t          | f              | f            | f
ST_LineString  | t          | f              | f            | f
ST_Point       | t          | t              | f            | f

```

**Siehe auch**

[ST\\_Boundary](#), [ST\\_ContainsProperly](#), [ST\\_Covers](#), [ST\\_CoveredBy](#), [ST\\_Equals](#), [ST\\_Within](#)

**7.11.1.3 ST\_ContainsProperly**

ST\_ContainsProperly — Prüft, ob jeder Punkt von B im Inneren von A liegt

**Synopsis**

boolean **ST\_ContainsProperly**(geometry geomA, geometry geomB);

**Beschreibung**

Gibt true zurück, wenn jeder Punkt von B im Inneren von A liegt (oder äquivalent dazu, kein Punkt von B liegt im Rand oder außerhalb von A).

Mathematisch ausgedrückt:  $ST\_EnthältEcht(A, B) \Leftrightarrow Int(A) \cap B = B$

A enthält B ordnungsgemäß, wenn die DE-9IM-Schnittpunktmatrix für die beiden Geometrien mit [T\*\*FF\*FF\*]

A enthält sich selbst nicht richtig, aber es enthält sich selbst.

Eine Anwendung für dieses Prädikat ist die Berechnung der Schnittpunkte einer Menge von Geometrien mit einer großen polygonalen Geometrie. Da die Schnittmenge eine recht langsame Operation ist, kann es effizienter sein, containsProperly zu verwenden, um Testgeometrien herauszufiltern, die vollständig innerhalb des Bereichs liegen. In diesen Fällen ist von vornherein bekannt, dass der Schnittpunkt genau die ursprüngliche Testgeometrie ist.



**Note**

Diese Funktion beinhaltet automatisch einen Bounding-Box-Vergleich, der alle räumlichen Indizes verwendet, die für die Geometrien verfügbar sind. Um die Verwendung von Indizes zu vermeiden, verwenden Sie die Funktion `_ST_ContainsProperly`.



**Note**

Der Vorteil dieses Prädikats gegenüber [ST\\_Contains](#) und [ST\\_Intersects](#) ist, dass es effizienter berechnet werden kann, ohne dass die Topologie an einzelnen Punkten berechnet werden muss.

Wird vom GEOS Modul ausgeführt

Verfügbarkeit: 1.4.0



**Important**

Verbessert: 3.0.0 ermöglicht die Unterstützung von GEOMETRYCOLLECTION



**Important**

Verwenden Sie diese Funktion nicht mit ungültigen Geometrien. Sie werden unerwartete Ergebnisse erhalten.

**Beispiele**

```
--a circle within a circle
SELECT ST_ContainsProperly(smallc, bigc) As smallcontainspropbig,
       ST_ContainsProperly(bigc,smallc) As bigcontainspropsmall,
       ST_ContainsProperly(bigc, ST_Union(smallc, bigc)) as bigcontainspropunion,
       ST_Equals(bigc, ST_Union(smallc, bigc)) as bigisunion,
       ST_Covers(bigc, ST_ExteriorRing(bigc)) As bigcoversexterior,
       ST_ContainsProperly(bigc, ST_ExteriorRing(bigc)) As bigcontainsexterior
FROM (SELECT ST_Buffer(ST_GeomFromText('POINT(1 2)'), 10) As smallc,
       ST_Buffer(ST_GeomFromText('POINT(1 2)'), 20) As bigc) As foo;
--Result
smallcontainspropbig | bigcontainspropsmall | bigcontainspropunion | bigisunion | ↔
bigcoversexterior | bigcontainsexterior
-----+-----+-----+-----+-----
f                | t                | f                | t                | ↔
                | f                |                  |                  |

--example demonstrating difference between contains and contains properly
SELECT ST_GeometryType(geomA) As geomtype, ST_Contains(geomA,geomA) AS acontainsa, ↔
       ST_ContainsProperly(geomA, geomA) AS acontainspropa,
       ST_Contains(geomA, ST_Boundary(geomA)) As acontainsba, ST_ContainsProperly(geomA, ↔
       ST_Boundary(geomA)) As acontainspropba
FROM (VALUES ( ST_Buffer(ST_Point(1,1), 5,1) ),
            ( ST_MakeLine(ST_Point(1,1), ST_Point(-1,-1) ) ),
            ( ST_Point(1,1) )
       ) As foo(geomA);

geomtype      | acontainsa | acontainspropa | acontainsba | acontainspropba
-----+-----+-----+-----+-----
ST_Polygon    | t          | f              | f           | f
ST_LineString | t          | f              | f           | f
ST_Point      | t          | t              | f           | f
```

**Siehe auch**

[ST\\_GeometryType](#), [ST\\_Boundary](#), [ST\\_Contains](#), [ST\\_Covers](#), [ST\\_CoveredBy](#), [ST\\_Equals](#), [ST\\_Relate](#), [ST\\_Within](#)



#### 7.11.1.4 ST\_CoveredBy

ST\_CoveredBy — Prüft, ob jeder Punkt von A in B liegt

#### Synopsis

```
boolean ST_CoveredBy(geometry geomA, geometry geomB);
boolean ST_CoveredBy(geography geogA, geography geogB);
```

#### Beschreibung

Gibt `true` zurück, wenn jeder Punkt in Geometrie/Geografie A innerhalb von Geometrie/Geografie B liegt (d. h. das Innere oder die Grenze von B schneidet). Gleichbedeutend wird getestet, dass kein Punkt von A außerhalb von B liegt.

Mathematisch ausgedrückt:  $ST\_CoveredBy(A, B) \Leftrightarrow A \cap B = A$

ST\_CoveredBy ist die Umkehrung von **ST\_Covers**.  $ST\_CoveredBy(A, B) = ST\_Covers(B, A)$ .

Im Allgemeinen sollte diese Funktion anstelle von **ST\_Within** verwendet werden, da sie eine einfachere Definition hat, die nicht die Eigenart hat, dass "Grenzen nicht innerhalb ihrer Geometrie liegen".



#### Note

Diese Funktion beinhaltet automatisch einen Bounding-Box-Vergleich, der alle räumlichen Indizes verwendet, die für die Geometrien verfügbar sind. Um die Verwendung von Indizes zu vermeiden, verwenden Sie die Funktion `_ST_CoveredBy`.



#### Important

Verbessert: 3.0.0 ermöglicht die Unterstützung von `GEOMETRYCOLLECTION`



#### Important

Verwenden Sie diese Funktion nicht mit ungültigen Geometrien. Sie werden unerwartete Ergebnisse erhalten.

Wird durch das GEOS Modul ausgeführt

Verfügbarkeit: 1.2.2

HINWEIS: Dies ist die "zulässige" Version, die einen booleschen Wert und keine ganze Zahl zurückgibt.

Kein OGC-Standard, aber Oracle hat ihn auch.

#### Beispiele

```
--a circle coveredby a circle
SELECT ST_CoveredBy(smallc,smallc) As smallinsmall,
       ST_CoveredBy(smallc, bigc) As smallcoveredbybig,
       ST_CoveredBy(ST_ExteriorRing(bigc), bigc) As exteriorcoveredbybig,
       ST_Within(ST_ExteriorRing(bigc),bigc) As exeriorwithinbig
FROM (SELECT ST_Buffer(ST_GeomFromText('POINT(1 2)'), 10) As smallc,
          ST_Buffer(ST_GeomFromText('POINT(1 2)'), 20) As bigc) As foo;
--Result
smallinsmall | smallcoveredbybig | exteriorcoveredbybig | exeriorwithinbig
-----+-----+-----+-----
t           | t                 | t                 | f
(1 row)
```

**Siehe auch**

[ST\\_Contains](#), [ST\\_Covers](#), [ST\\_ExteriorRing](#), [ST\\_Within](#)

**7.11.1.5 ST\_Covers**

ST\_Covers — Prüft, ob jeder Punkt von B in A liegt

**Synopsis**

```
boolean ST_Covers(geometry geomA, geometry geomB);  
boolean ST_Covers(geography geogpolyA, geography geogpointB);
```

**Beschreibung**

Gibt `true` zurück, wenn jeder Punkt in Geometrie/Geografie B innerhalb von Geometrie/Geografie A liegt (d. h. das Innere oder den Rand von A schneidet). Äquivalent dazu wird getestet, dass kein Punkt von B außerhalb (im Äußeren von) A liegt.

Mathematisch ausgedrückt:  $ST\_Covers(A, B) \Leftrightarrow A \cap B = B$

ST\_Covers ist die Umkehrung von [ST\\_CoveredBy](#).  $ST\_Covers(A, B) = ST\_CoveredBy(B, A)$ .

Im Allgemeinen sollte diese Funktion anstelle von [ST\\_Contains](#) verwendet werden, da sie eine einfachere Definition hat, die nicht die Eigenart hat, dass "Geometrien ihre Begrenzung nicht enthalten".

**Note**

Diese Funktion beinhaltet automatisch einen Bounding-Box-Vergleich, der alle räumlichen Indizes verwendet, die für die Geometrien verfügbar sind. Um die Verwendung von Indizes zu vermeiden, verwenden Sie die Funktion `_ST_Covers`.

**Important**

Verbessert: 3.0.0 ermöglicht die Unterstützung von `GEOMETRYCOLLECTION`

**Important**

Verwenden Sie diese Funktion nicht mit ungültigen Geometrien. Sie werden unerwartete Ergebnisse erhalten.

Wird durch das GEOS Modul ausgeführt

Verbessert: 2.4.0 Unterstützung für Polygon in Polygon und Linie in Polygon für Geografietypen hinzugefügt

Verbessert: 2.3.0 Verbesserung des PIP-Kurzschlusses für Geometrien, erweitert um die Unterstützung von MultiPoints mit wenigen Punkten. Frühere Versionen unterstützten nur Punkte in Polygonen.

Verfügbarkeit: 1.5 - Unterstützung von geographischen Koordinaten.

Verfügbarkeit: 1.2.2

HINWEIS: Dies ist die "zulässige" Version, die einen booleschen Wert und keine ganze Zahl zurückgibt.

Kein OGC-Standard, aber Oracle hat ihn auch.

## Beispiele

### Beispiel für Geometrie

```
--a circle covering a circle
SELECT ST_Covers(smallc,smallc) As smallinsmall,
       ST_Covers(smallc, bigc) As smallcoversbig,
       ST_Covers(bigc, ST_ExteriorRing(bigc)) As bigcoversexterior,
       ST_Contains(bigc, ST_ExteriorRing(bigc)) As bigcontainsexterior
FROM (SELECT ST_Buffer(ST_GeomFromText('POINT(1 2)'), 10) As smallc,
       ST_Buffer(ST_GeomFromText('POINT(1 2)'), 20) As bigc) As foo;
--Result
smallinsmall | smallcoversbig | bigcoversexterior | bigcontainsexterior
-----+-----+-----+-----
t            | f              | t                 | f
(1 row)
```

### 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](#)

### 7.11.1.6 ST\_Crosses

`ST_Crosses` — Prüft, ob zwei Geometrien einige, aber nicht alle, innere Punkte gemeinsam haben

## Synopsis

boolean `ST_Crosses`(geometry g1, geometry g2);

## Beschreibung

Vergleicht zwei Geometrieobjekte und gibt `true` zurück, wenn sich ihre Schnittpunkte "räumlich kreuzen"; das heißt, die Geometrien haben einige, aber nicht alle Innenpunkte gemeinsam. Die Schnittmenge der Innenräume der Geometrien darf nicht leer sein und muss eine Dimension haben, die kleiner ist als die maximale Dimension der beiden Eingabegeometrien, und die Schnittmenge der beiden Geometrien darf keiner der beiden Geometrien entsprechen. Andernfalls gibt es `false` zurück. Die Kreuzungsbeziehung ist symmetrisch und irreflexiv.

Mathematisch ausgedrückt:  $ST\_Crosses(A, B) \Leftrightarrow (dim(Int(A) \cap Int(B)) < \max(dim(Int(A)), dim(Int(B))) \wedge (A \cap B \neq A) \wedge (A \cap B \neq B))$

Geometrien kreuzen sich, wenn ihre DE-9IM Schnittpunktmatrix übereinstimmt:

- T\*T\*\*\*\*\* für die Situationen Punkt/Linie, Punkt/Fläche und Linie/Fläche
- T\*\*\*\*\*T\*\* für die Situationen Linie/Punkt, Bereich/Punkt und Bereich/Linie

- 0\*\*\*\*\* für Line/Line-Situationen
- das Ergebnis ist falsch für Punkt/Punkt und Bereich/Fläche Situationen

**Note**

Die OpenGIS Simple Features Specification definiert dieses Prädikat nur für die Situationen Punkt/Linie, Punkt/Fläche, Linie/Linie und Linie/Fläche. JTS / GEOS erweitert die Definition so, dass sie auch für die Situationen Linie/Punkt, Fläche/Punkt und Fläche/Linie gilt. Dadurch wird die Beziehung symmetrisch.

**Note**

Diese Funktion beinhaltet automatisch einen Bounding-Box-Vergleich, der alle räumlichen Indizes verwendet, die für die Geometrien verfügbar sind.

**Important**

Verbessert: 3.0.0 ermöglicht die Unterstützung von GEOMETRYCOLLECTION



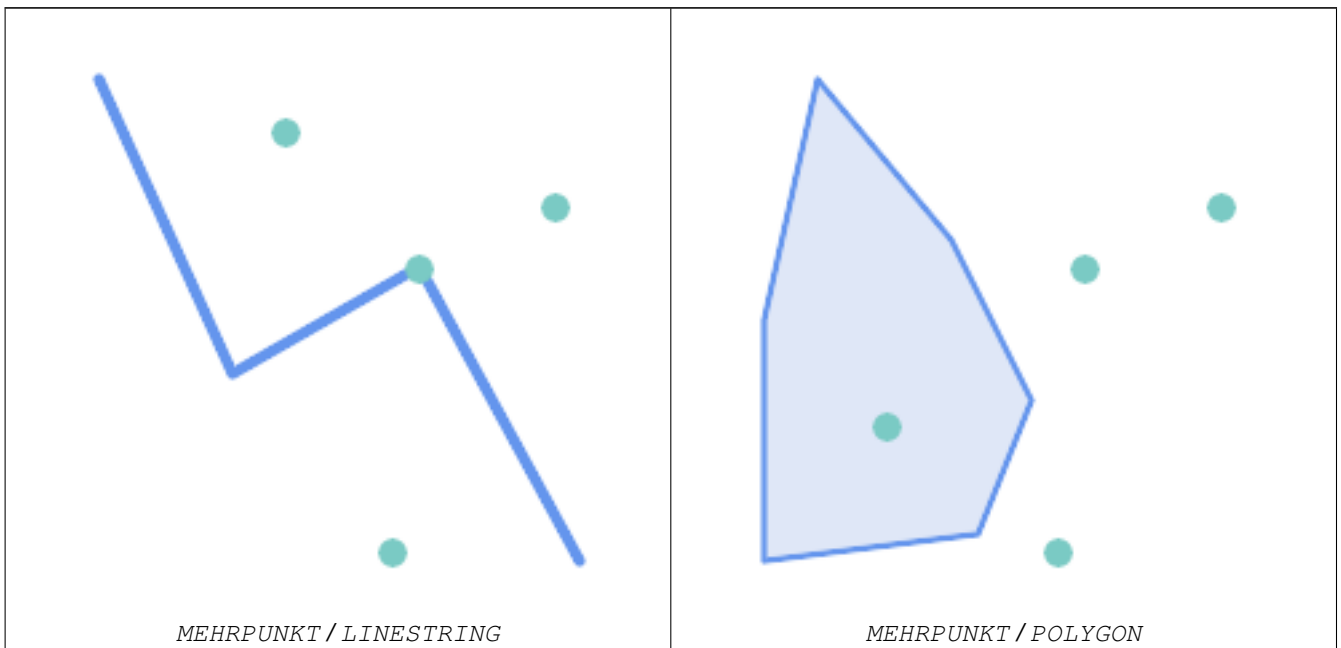
Diese Methode implementiert die [OGC Simple Features Implementation Specification for SQL 1.1](#). s2.1.13.3

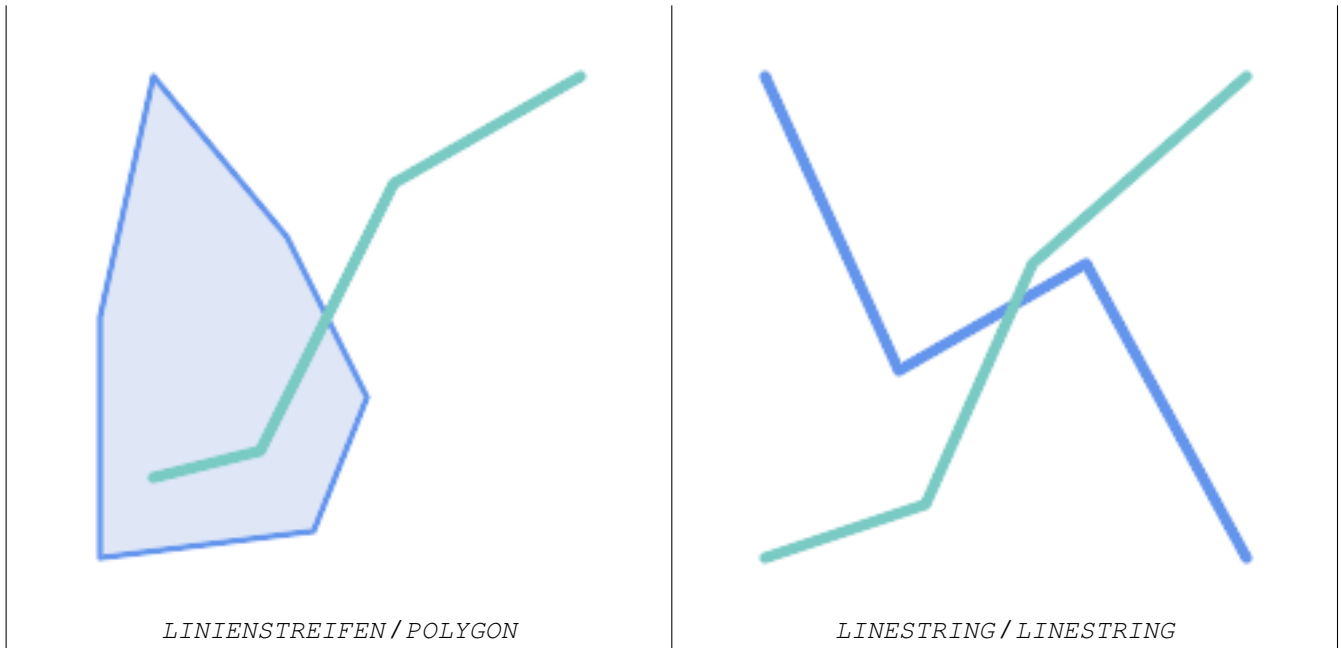


Diese Methode setzt die SQL/MM-Spezifikation um. SQL-MM 3: 5.1.29

**Beispiele**

Die folgenden Situationen ergeben alle `true`.





Stellen Sie sich eine Situation vor, in der ein Benutzer zwei Tabellen hat: eine Tabelle mit Straßen und eine Tabelle mit Autobahnen.

```
CREATE TABLE roads (
  id serial NOT NULL,
  geom geometry,
  CONSTRAINT roads_pkey PRIMARY KEY ( ↔
    road_id)
);
```

```
CREATE TABLE highways (
  id serial NOT NULL,
  the_geom geometry,
  CONSTRAINT roads_pkey PRIMARY KEY ( ↔
    road_id)
);
```

Um eine Liste von Straßen zu ermitteln, die eine Autobahn kreuzen, verwenden Sie eine Abfrage, die der folgenden ähnelt:

```
SELECT roads.id
FROM roads, highways
WHERE ST_Crosses(roads.geom, highways.geom);
```

### Siehe auch

[ST\\_Contains](#), [ST\\_Overlaps](#)

#### 7.11.1.7 ST\_Disjoint

**ST\_Disjoint** — Prüft, ob zwei Geometrien keine gemeinsamen Punkte haben

### Synopsis

boolean **ST\_Disjoint**( geometry A , geometry B );

## Beschreibung

Gibt `true` zurück, wenn zwei Geometrien disjunkt sind. Geometrien sind unzusammenhängend, wenn sie keinen gemeinsamen Punkt haben.

Wenn eine andere räumliche Beziehung für ein Paar von Geometrien gilt, sind sie nicht disjunkt. Disjunkt bedeutet, dass `ST_Intersects` falsch ist.

Mathematisch ausgedrückt:  $ST\_Disjoint(A, B) \Leftrightarrow A \cap B = \emptyset$



### Important

Verbessert: 3.0.0 ermöglicht die Unterstützung von `GEOMETRYCOLLECTION`

Wird durch das GEOS Modul ausgeführt



### Note

Dieser Funktionsaufruf verwendet keine Indizes. Ein negiertes `ST_Intersects` Prädikat kann als leistungsfähigere Alternative verwendet werden, die Indizes verwendet: `ST_Disjoint(A, B) = NOT ST_Intersects(A, B)`



### Note

HINWEIS: Dies ist die "zulässige" Version, die einen booleschen Wert und keine ganze Zahl zurückgibt.



Diese Methode implementiert die [OGC Simple Features Implementation Specification for SQL 1.1](#). `s2.1.1.2 //s2.1.13.3 - a.Relate(b, 'FF*FF*****')`



Diese Methode setzt die SQL/MM-Spezifikation um. SQL-MM 3: 5.1.26

## Beispiele

```
SELECT ST_Disjoint('POINT(0 0)::geometry, 'LINESTRING ( 2 0, 0 2 ) '::geometry);
st_disjoint
-----
t
(1 row)
SELECT ST_Disjoint('POINT(0 0)::geometry, 'LINESTRING ( 0 0, 0 2 ) '::geometry);
st_disjoint
-----
f
(1 row)
```

## Siehe auch

[ST\\_Intersects](#)

### 7.11.1.8 ST\_Equals

`ST_Equals` — Prüft, ob zwei Geometrien dieselbe Menge von Punkten enthalten

## Synopsis

boolean **ST\_Equals**(geometry A, geometry B);

## Beschreibung

Gibt `true` zurück, wenn die angegebenen Geometrien "topologisch gleich" sind. Verwenden Sie dies für eine "bessere" Antwort als `=`. Topologische Gleichheit bedeutet, dass die Geometrien die gleiche Dimension haben und ihre Punktmengen den gleichen Raum einnehmen. Das bedeutet, dass die Reihenfolge der Eckpunkte in topologisch gleichen Geometrien unterschiedlich sein kann. Um zu überprüfen, ob die Reihenfolge der Punkte konsistent ist, verwenden Sie **ST\_OrderingEquals** (es ist zu beachten, dass `ST_OrderingEquals` etwas strenger ist als die einfache Überprüfung, ob die Reihenfolge der Punkte gleich ist).

Mathematisch ausgedrückt:  $ST\_Equals(A, B) \Leftrightarrow A = B$

Es gilt die folgende Beziehung:  $ST\_Equals(A, B) \Leftrightarrow ST\_Within(A,B) \wedge ST\_Within(B,A)$



### Important

Verbessert: 3.0.0 ermöglicht die Unterstützung von `GEOMETRYCOLLECTION`



Diese Methode implementiert die [OGC Simple Features Implementation Specification for SQL 1.1](#). s2.1.1.2



Diese Methode setzt die SQL/MM-Spezifikation um. SQL-MM 3: 5.1.24

Geändert: 2.2.0 Gibt auch bei ungültigen Geometrien `true` zurück, wenn sie binär gleich sind

## Beispiele

```
SELECT ST_Equals(ST_GeomFromText('LINESTRING(0 0, 10 10)'),
  ST_GeomFromText('LINESTRING(0 0, 5 5, 10 10)'));
 st_equals
-----
t
(1 row)

SELECT ST_Equals(ST_Reverse(ST_GeomFromText('LINESTRING(0 0, 10 10)'),
  ST_GeomFromText('LINESTRING(0 0, 5 5, 10 10)'));
 st_equals
-----
t
(1 row)
```

## Siehe auch

[ST\\_IsValid](#), [ST\\_OrderingEquals](#), [ST\\_Reverse](#), [ST\\_Within](#)

### 7.11.1.9 ST\_Intersects

`ST_Intersects` — Prüft, ob sich zwei Geometrien schneiden (sie haben mindestens einen Punkt gemeinsam)

## Synopsis

boolean **ST\_Intersects**( geometry geomA , geometry geomB );  
 boolean **ST\_Intersects**( geography geogA , geography geogB );

## Beschreibung

Gibt `true` zurück, wenn sich zwei Geometrien überschneiden. Geometrien überschneiden sich, wenn sie einen Punkt gemeinsam haben.

Für die Geografie wird eine Entfernungstoleranz von 0,00001 Metern verwendet (d. h. Punkte, die sehr nahe beieinander liegen, werden als sich überschneidend betrachtet).

Mathematisch ausgedrückt:  $ST\_Intersects(A, B) \Leftrightarrow A \cap B \neq \emptyset$

Geometrien überschneiden sich, wenn ihre DE-9IM Schnittpunktmatrix mit einer der folgenden übereinstimmt:

- T\*\*\*\*\*
- \*T\*\*\*\*\*
- \*\*\*T\*\*\*\*\*
- \*\*\*\*T\*\*\*\*\*

Eine räumliche Überschneidung wird von allen anderen Tests für räumliche Beziehungen impliziert, mit Ausnahme von `ST_Disjoint`, das prüft, dass sich Geometrien NICHT überschneiden.



### Note

Diese Funktion beinhaltet automatisch einen Bounding-Box-Vergleich, der alle räumlichen Indizes verwendet, die für die Geometrien verfügbar sind.

Geändert: 3.0.0 SFCGAL Version entfernt und native Unterstützung für 2D TINS hinzugefügt.

Verbessert: 2.5.0 Unterstützt GEOMETRYCOLLECTION.

Verbessert: 2.3.0 Verbesserung des PIP-Kurzschlusses erweitert um die Unterstützung von MultiPoints mit wenigen Punkten. Frühere Versionen unterstützten nur Punkte in Polygonen.

Wird vom GEOS-Modul (für Geometrie) durchgeführt, Geographie ist nativ

Verfügbarkeit: Mit Version 1.5 wurde die Unterstützung für Geografie eingeführt.



### Note

Für die Geografie hat diese Funktion eine Entfernungstoleranz von etwa 0,00001 Metern und verwendet die Kugel statt der Sphäroidberechnung.



### Note

HINWEIS: Dies ist die "zulässige" Version, die einen booleschen Wert und keine ganze Zahl zurückgibt.



Diese Methode implementiert die [OGC Simple Features Implementation Specification for SQL 1.1](#). s2.1.1.2 //s2.1.13.3 - `ST_Intersects(g1, g2) --> Not (ST_Disjoint(g1, g2))`



Diese Methode setzt die SQL/MM-Spezifikation um. SQL-MM 3: 5.1.27



Diese Methode unterstützt kreisförmige Strings und Kurven.



Diese Funktion unterstützt Dreiecke und dreieckige unregelmäßige Netzoberflächen (TIN).



## Beispiele mit dem geometrischen Datentyp

```
SELECT ST_Intersects('POINT(0 0)::geometry, 'LINESTRING ( 2 0, 0 2 ) '::geometry);
st_intersects
-----
f
(1 row)
SELECT ST_Intersects('POINT(0 0)::geometry, 'LINESTRING ( 0 0, 0 2 ) '::geometry);
st_intersects
-----
t
(1 row)

-- Look up in table. Make sure table has a GiST index on geometry column for faster lookup.
SELECT id, name FROM cities WHERE ST_Intersects(geom, 'SRID=4326;POLYGON((28 53,27.707 ↵
52.293,27 52,26.293 52.293,26 53,26.293 53.707,27 54,27.707 53.707,28 53))');
id | name
----+-----
 2 | Minsk
(1 row)
```

## Beispiele für den geographischen Datentyp

```
SELECT ST_Intersects (
  'SRID=4326;LINESTRING(-43.23456 72.4567,-43.23456 72.4568) '::geography,
  'SRID=4326;POINT(-43.23456 72.4567772) '::geography
);

st_intersects
-----
t
```

### Siehe auch

[&&](#), [ST\\_3DIntersects](#), [ST\\_Disjoint](#)

#### 7.11.1.10 ST\_LineCrossingDirection

`ST_LineCrossingDirection` — Gibt eine Zahl zurück, die das Kreuzungsverhalten von zwei `LineStrings` angibt

### Synopsis

integer `ST_LineCrossingDirection`(geometry linestringA, geometry linestringB);

### Beschreibung

Bei zwei Linienstrings wird eine ganze Zahl zwischen -3 und 3 zurückgegeben, die angibt, welche Art von Kreuzungsverhalten zwischen ihnen besteht. 0 bedeutet keine Kreuzung. Dies wird nur für `LINESTRINGs` unterstützt.

Die Kreuzungsnummer hat die folgende Bedeutung:

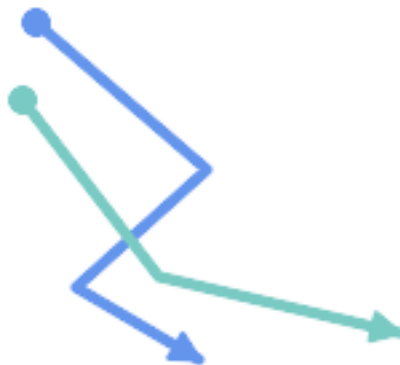
- 0: LINIE KEIN KREUZ
- -1: LINIENKREUZ LINKS
- 1: LINIE QUER RECHTS

- -2: LINIE MULTIKREUZ ENDE LINKS
- 2: LINIE MULTICROSS ENDE RECHTS
- -3: LINIE MULTIKREUZ ENDE GLEICH ERSTE LINKS
- 3: LINIE MULTICROSS ENDE GLEICH ERSTE RECHTS

Verfügbarkeit: 1.4

### Beispiele

**Beispiel:** LINIENKREUZUNG LINKS und LINIENKREUZUNG RECHTS

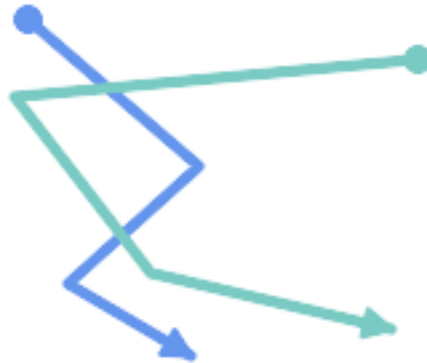


*Blau: Linie A; Grün: Linie B*

```
SELECT ST_LineCrossingDirection(lineA, lineB) As A_cross_B,
       ST_LineCrossingDirection(lineB, lineA) As B_cross_A
FROM (SELECT
      ST_GeomFromText('LINESTRING(25 169,89 114,40 70,86 43)') As lineA,
      ST_GeomFromText('LINESTRING (20 140, 71 74, 161 53)') As lineB
    ) As foo;
```

A_cross_B	B_cross_A
-1	1

**Beispiel:** LINE MULTICROSS END SAME FIRST LEFT und LINE MULTICROSS END SAME FIRST RIGHT

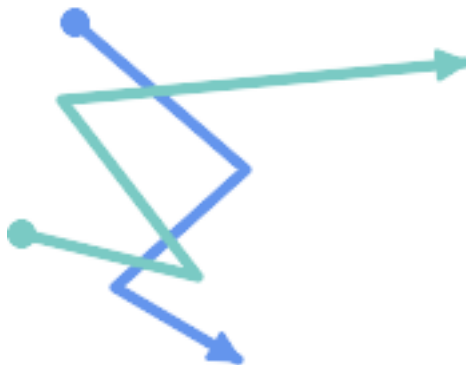


*Blau: Linie A; Grün: Linie B*

```
SELECT ST_LineCrossingDirection(lineA, lineB) As A_cross_B,
       ST_LineCrossingDirection(lineB, lineA) As B_cross_A
FROM (SELECT
      ST_GeomFromText('LINESTRING(25 169,89 114,40 70,86 43)') As lineA,
      ST_GeomFromText('LINESTRING(171 154,20 140,71 74,161 53)') As lineB
    ) As foo;
```

A_cross_B	B_cross_A
3	-3

**Beispiel:** LINE MULTICROSS END LEFT und LINE MULTICROSS END RIGHT



*Blau: Linie A; Grün: Linie B*

```
SELECT ST_LineCrossingDirection(lineA, lineB) As A_cross_B,
       ST_LineCrossingDirection(lineB, lineA) As B_cross_A
FROM (SELECT
      ST_GeomFromText('LINESTRING(25 169,89 114,40 70,86 43)') As lineA,
      ST_GeomFromText('LINESTRING(5 90, 71 74, 20 140, 171 154)') As lineB
    ) As foo;
```

```
A_cross_B | B_cross_A
-----+-----
      -2 |          2
```

### Beispiel: Findet alle Straßen, die sich kreuzen

```
SELECT s1.gid, s2.gid, ST_LineCrossingDirection(s1.geom, s2.geom)
   FROM streets s1 CROSS JOIN streets s2
      ON (s1.gid != s2.gid AND s1.geom && s2.geom )
WHERE ST_LineCrossingDirection(s1.geom, s2.geom)
> 0;
```

### Siehe auch

[ST\\_Crosses](#)

#### 7.11.1.11 ST\_OrderingEquals

`ST_OrderingEquals` — Prüft, ob zwei Geometrien die gleiche Geometrie darstellen und Punkte in der gleichen Richtungsreihenfolge haben

### Synopsis

boolean `ST_OrderingEquals`(geometry A, geometry B);

### Beschreibung

`ST_OrderingEquals` vergleicht zwei Geometrien und gibt t (TRUE) zurück, wenn die Geometrien gleich sind und die Koordinaten in der gleichen Reihenfolge sind; andernfalls gibt es f (FALSE) zurück.



#### Note

Diese Funktion ist gemäß der ArcSDE-SQL-Spezifikation und nicht gemäß SQL-MM implementiert. [http://edndoc.esri.com/arcscde/9.1/sql\\_api/sqlapi3.htm#ST\\_OrderingEquals](http://edndoc.esri.com/arcscde/9.1/sql_api/sqlapi3.htm#ST_OrderingEquals)



Diese Methode setzt die SQL/MM-Spezifikation um. 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](#)

**7.11.1.12 ST\_Overlaps**

**ST\_Overlaps** — Prüft, ob zwei Geometrien die gleiche Abmessung haben und sich schneiden, aber jede mindestens einen Punkt hat, der nicht in der anderen liegt

**Synopsis**

boolean **ST\_Overlaps**(geometry A, geometry B);

**Beschreibung**

Gibt TRUE zurück, wenn sich Geometrie A und B "räumlich überschneiden". Zwei Geometrien überlappen sich, wenn sie dieselbe Dimension haben, ihre Innenräume sich in dieser Dimension schneiden und jede Geometrie mindestens einen Punkt im Inneren der anderen hat (oder äquivalent dazu, keine der beiden Geometrien die andere überdeckt). Die Überlappungsbeziehung ist symmetrisch und irreflexiv.

Mathematisch ausgedrückt:  $ST\_Overlaps(A, B) \Leftrightarrow (dim(A) = dim(B) = dim(Int(A) \cap Int(B))) \wedge (A \cap B \neq A) \wedge (A \cap B \neq B)$

**Note**

Diese Funktion beinhaltet automatisch einen Bounding-Box-Vergleich, der alle räumlichen Indizes verwendet, die für die Geometrien verfügbar sind. Um die Verwendung des Index zu vermeiden, verwenden Sie die Funktion `_ST_Overlaps`.

Wird durch das GEOS Modul ausgeführt

**Important**

Verbessert: 3.0.0 ermöglicht die Unterstützung von `GEOMETRYCOLLECTION`

HINWEIS: Dies ist die "zulässige" Version, die einen booleschen Wert und keine ganze Zahl zurückgibt.



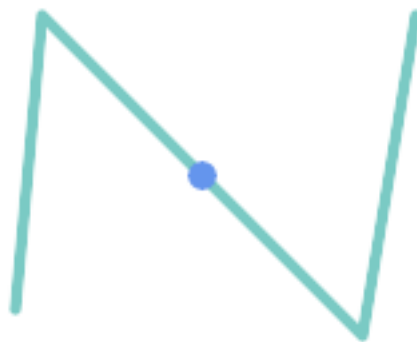
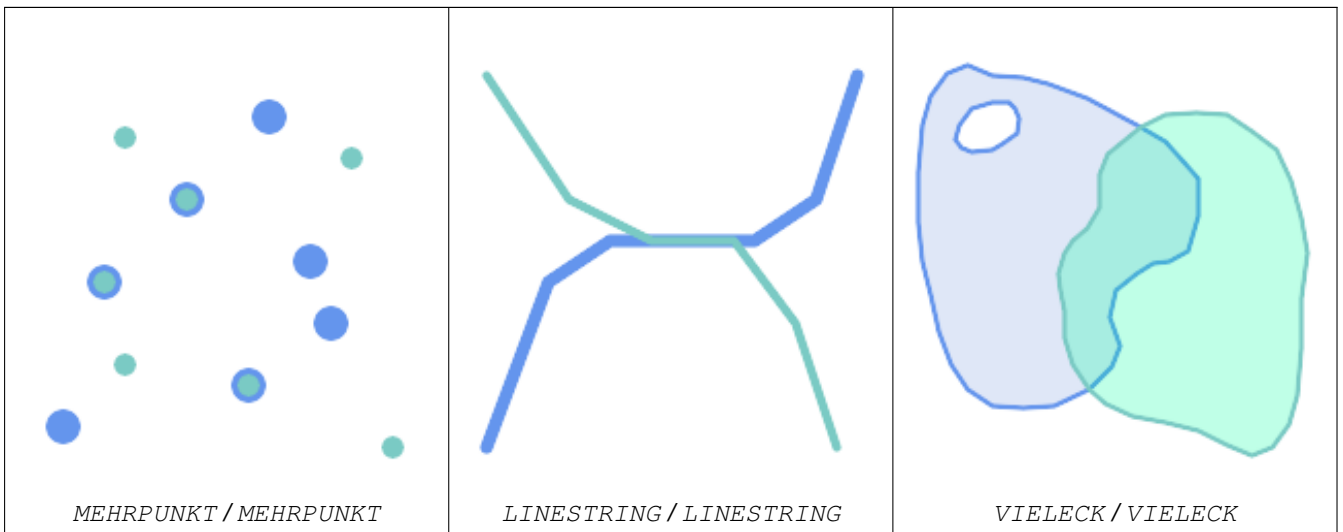
Diese Methode implementiert die [OGC Simple Features Implementation Specification for SQL 1.1](#). s2.1.1.2 // s2.1.13.3



Diese Methode setzt die SQL/MM-Spezifikation um. SQL-MM 3: 5.1.32

**Beispiele**

`ST_Overlaps` gibt TRUE in den folgenden Situationen zurück:



Ein Punkt auf einem LineString ist enthalten, aber da er eine geringere Dimension hat, überschneidet oder kreuzt er sich nicht.

```
SELECT ST_Overlaps(a,b) AS overlaps,          ST_Crosses(a,b) AS crosses,
       ST_Intersects(a, b) AS intersects,     ST_Contains(b,a) AS b_contains_a
FROM (SELECT ST_GeomFromText('POINT (100 100)') As a,
       ST_GeomFromText('LINESTRING (30 50, 40 160, 160 40, 180 160)') AS b) AS t
```

overlaps	crosses	intersects	b_contains_a
f	f	t	t



Ein LineString, der ein Polygon teilweise abdeckt, schneidet und kreuzt sich, überschneidet sich aber nicht, da er unterschiedliche Abmessungen hat.

```
SELECT ST_Overlaps(a,b) AS overlaps,      ST_Crosses(a,b) AS crosses,
       ST_Intersects(a, b) AS intersects,  ST_Contains(a,b) AS contains
FROM (SELECT ST_GeomFromText('POLYGON ((40 170, 90 30, 180 100, 40 170))') AS a,
       ST_GeomFromText('LINESTRING(10 10, 190 190)') AS b) AS t;
```

overlap	crosses	intersects	contains
f	t	t	f



Zwei Polygone, die sich schneiden, ohne dass das eine vom anderen enthalten ist, überschneiden sich, aber nicht, weil ihr Schnittpunkt die gleiche Dimension hat.

```
SELECT ST_Overlaps(a,b) AS overlaps,      ST_Crosses(a,b) AS crosses,
       ST_Intersects(a, b) AS intersects,  ST_Contains(b, a) AS b_contains_a,
       ST_Dimension(a) AS dim_a, ST_Dimension(b) AS dim_b,
       ST_Dimension(ST_Intersection(a,b)) AS dim_int
FROM (SELECT ST_GeomFromText('POLYGON ((40 170, 90 30, 180 100, 40 170))') AS a,
       ST_GeomFromText('POLYGON ((110 180, 20 60, 130 90, 110 180))') AS b) AS t;
```

overlaps	crosses	intersects	b_contains_a	dim_a	dim_b	dim_int
t	f	t	f	2	2	2

## Siehe auch

[ST\\_Contains](#), [ST\\_Crosses](#), [ST\\_Dimension](#), [ST\\_Intersects](#)

### 7.11.1.13 ST\_Relate

`ST_Relate` — Prüft, ob zwei Geometrien eine topologische Beziehung haben, die einem Schnittpunktmatrixmuster entspricht, oder berechnet ihre Schnittpunktmatrix

## Synopsis

```
boolean ST_Relate(geometry geomA, geometry geomB, text intersectionMatrixPattern);
text ST_Relate(geometry geomA, geometry geomB);
text ST_Relate(geometry geomA, geometry geomB, integer boundaryNodeRule);
```

## Beschreibung

Diese Funktionen ermöglichen die Prüfung und Bewertung der räumlichen (topologischen) Beziehung zwischen zwei Geometrien, wie sie durch das [Dimensionally Extended 9-Intersection Model](#) (DE-9IM) definiert sind.

DE-9IM wird als 9-Element-Matrix spezifiziert, die die Dimension der Schnittpunkte zwischen dem Inneren, dem Rand und dem Äußeren von zwei Geometrien angibt. Sie wird durch eine 9-stellige Zeichenkette mit den Symbolen "F", "0", "1", "2" dargestellt (z. B. "FF1FF0102").

Eine bestimmte Art von räumlicher Beziehung kann getestet werden, indem die Kreuzungsmatrix mit einem Muster der *Kreuzungsmatrix* verglichen wird. Muster können die zusätzlichen Symbole 'T' (bedeutet "Schnittpunkt ist nicht leer") und '\*' (bedeutet "beliebiger Wert") enthalten. Allgemeine räumliche Beziehungen werden durch die benannten Funktionen [ST\\_Contains](#), [ST\\_ContainsProperly](#), [ST\\_Covers](#), [ST\\_CoveredBy](#), [ST\\_Crosses](#), [ST\\_Disjoint](#), [ST\\_Equals](#), [ST\\_Intersects](#), [ST\\_Overlaps](#), [ST\\_Touches](#) und [ST\\_Within](#) bereitgestellt. Die Verwendung eines expliziten Musters ermöglicht es, mehrere Bedingungen für Überschneidungen, Kreuzungen usw. in einem Schritt zu testen. Es ermöglicht auch das Testen räumlicher Beziehungen, die keine benannte räumliche Beziehungsfunktion haben. Zum Beispiel hat die Beziehung "Interior-Intersects" das DE-9IM-Muster T\*\*\*\*\*, das von keinem benannten Prädikat ausgewertet wird.

Weitere Informationen finden Sie unter [Section 5.1](#).

**Variante 1:** Prüft, ob zwei Geometrien nach dem vorgegebenen `intersectionMatrixPattern` räumlich zusammenhängen.



#### Note

Im Gegensatz zu den meisten benannten räumlichen Beziehungsprädikaten enthält dieses NICHT automatisch einen Indexaufruf. Der Grund dafür ist, dass einige Beziehungen für Geometrien wahr sind, die sich NICHT schneiden (z. B. Disjoint). Wenn Sie ein Beziehungsmuster verwenden, das eine Überschneidung erfordert, dann schließen Sie den Indexaufruf `&&` ein.



#### Note

Es ist besser, eine benannte Beziehungsfunktion zu verwenden, wenn sie verfügbar ist, da sie automatisch einen räumlichen Index verwendet, wenn einer vorhanden ist. Außerdem können sie Leistungsoptimierungen implementieren, die bei einer vollständigen Beziehungsauswertung nicht verfügbar sind.



**Variante 2:** Gibt die DE-9IM-Matrixzeichenfolge für die räumliche Beziehung zwischen den beiden Eingabegeometrien zurück. Die Matrixzeichenfolge kann mit `ST_RelateMatch` auf Übereinstimmung mit einem DE-9IM-Muster getestet werden.

**Variante 3:** Wie Variante 2, erlaubt aber die Angabe einer **Boundary Node Rule**. Eine Boundary Node Rule erlaubt eine feinere Kontrolle darüber, ob die Endpunkte von MultiLineStrings als im DE-9IM Interior oder Boundary liegend betrachtet werden. Die `boundaryNodeRule` Werte sind:

- 1: **OGC-Mod2** - Linienendpunkte befinden sich in der Boundary, wenn sie eine ungerade Anzahl von Malen vorkommen. Dies ist die vom OGC SFS-Standard definierte Regel und ist die Vorgabe für `ST_Relate`.
- 2: **Endpunkt** - alle Endpunkte befinden sich in der Boundary.
- 3: **MultivalentEndpunkt** - Endpunkte befinden sich in der Boundary, wenn sie mehr als einmal vorkommen. Mit anderen Worten, die Grenze sind alle "verbundenen" oder "inneren" Endpunkte (aber nicht die "unverbundenen/äußeren").
- 4: **MonovalenterEndpunkt** - Endpunkte befinden sich in der Boundary, wenn sie nur einmal vorkommen. Mit anderen Worten, die Begrenzung sind alle "unverbundenen" oder "äußeren" Endpunkte.

Diese Funktion ist nicht in der OGC-Spezifikation enthalten, wird aber impliziert. siehe s2.1.13.2



Diese Methode implementiert die [OGC Simple Features Implementation Specification for SQL 1.1](#). s2.1.1.2 // s2.1.13.3



Diese Methode setzt die SQL/MM-Spezifikation um. SQL-MM 3: 5.1.25

Wird durch das GEOS Modul ausgeführt

Verbessert: 2.0.0 - Unterstützung für die Angabe von Grenzknotenregeln hinzugefügt.



#### Important

Verbessert: 3.0.0 ermöglicht die Unterstützung von `GEOMETRYCOLLECTION`

## Beispiele

Verwendung der booleschen Funktion zur Prüfung räumlicher Beziehungen.

```
SELECT ST_Relate('POINT(1 2)', ST_Buffer( 'POINT(1 2)', 2), '0FFFFFF212');
st_relate
-----
t

SELECT ST_Relate(POINT(1 2)', ST_Buffer( 'POINT(1 2)', 2), '*FF*FF212');
st_relate
-----
t
```

Testen eines benutzerdefinierten räumlichen Beziehungsmusters als Abfragebedingung, mit `&&`, um die Verwendung eines räumlichen Indexes zu ermöglichen.

```
-- Find compounds that properly intersect (not just touch) a poly (Interior Intersects)

SELECT c.* , p.name As poly_name
FROM polys AS p
INNER JOIN compounds As c
ON c.geom && p.geom
AND ST_Relate(p.geom, c.geom, 'T*****');
```

Berechnung der Kreuzungsmatrix für räumliche Beziehungen.

```

SELECT ST_Relate( 'POINT(1 2)',
                 ST_Buffer( 'POINT(1 2)', 2));
-----
0FFFFFF212

SELECT ST_Relate( 'LINESTRING(1 2, 3 4)',
                 'LINESTRING(5 6, 7 8)' );
-----
FF1FF0102

```

Verwendung verschiedener Boundary Node Rules zur Berechnung der räumlichen Beziehung zwischen einem LineString und einem MultiLineString mit einem doppelten Endpunkt (3 3):

- Unter Verwendung der **OGC-Mod2** Regel (1) liegt der doppelte Endpunkt im **Inneren** des MultiLineString, so dass der DE-9IM Matrixeintrag [aB:bI] 0 und [aB:bB] F ist.
- Bei Anwendung der Regel **Endpunkt** (2) liegt der doppelte Endpunkt in der **Grenze** des MultiLineString, so dass der DE-9IM Matrixeintrag [aB:bI] F und [aB:bB] 0 ist.

```

WITH data AS (SELECT
  'LINESTRING(1 1, 3 3)::geometry AS a_line,
  'MULTILINESTRING((3 3, 3 5), (3 3, 5 3)):: geometry AS b_multiline
)
SELECT ST_Relate( a_line, b_multiline, 1) AS bnr_mod2,
       ST_Relate( a_line, b_multiline, 2) AS bnr_endpoint
FROM data;

bnr_mod2 | bnr_endpoint
-----+-----
FF10F0102 | FF1F00102

```

### Siehe auch

Section 5.1, [ST\\_RelateMatch](#), [ST\\_Contains](#), [ST\\_ContainsProperly](#), [ST\\_Covers](#), [ST\\_CoveredBy](#), [ST\\_Crosses](#), [ST\\_Disjoint](#), [ST\\_Equals](#), [ST\\_Intersects](#), [ST\\_Overlaps](#), [ST\\_Touches](#), [ST\\_Within](#)

#### 7.11.1.14 ST\_RelateMatch

ST\_RelateMatch — Testet, ob eine DE-9IM Schnittpunktmatrix mit einem Schnittpunktmuster übereinstimmt

### Synopsis

boolean **ST\_RelateMatch**(text intersectionMatrix, text intersectionMatrixPattern);

### Beschreibung

Prüft, ob ein **Dimensionally Extended 9-Intersection Model** (DE-9IM) intersectionMatrix Wert ein intersectionMatrixPa erfüllt. Schnittpunktmatrixwerte können mit [ST\\_Relate](#) berechnet werden.

Weitere Informationen finden Sie unter Section 5.1.

Wird durch das GEOS Modul ausgeführt

Verfügbarkeit: 2.0.0

## Beispiele

```
SELECT ST_RelateMatch('101202FFF', 'TTTTTTFFF') ;
-- result --
t
```

Muster für gemeinsame räumliche Beziehungen, die mit den Werten der Kreuzungsmatrix abgeglichen werden, für eine Linie in verschiedenen Positionen relativ zu einem Polygon

```
SELECT pat.name AS relationship, pat.val AS pattern,
       mat.name AS position, mat.val AS matrix,
       ST_RelateMatch(mat.val, pat.val) AS match
FROM (VALUES ( 'Equality', 'T1FF1FFF1' ),
          ( 'Overlaps', 'T*T**T**' ),
          ( 'Within', 'T**F***' ),
          ( 'Disjoint', 'FF*FF***' )) AS pat(name,val)
CROSS JOIN
  (VALUES ('non-intersecting', 'FF1FF0212'),
         ('overlapping', '1010F0212'),
         ('inside', '1FF0FF212')) AS mat(name,val);
```

relationship	pattern	position	matrix	match
Equality	T1FF1FFF1	non-intersecting	FF1FF0212	f
Equality	T1FF1FFF1	overlapping	1010F0212	f
Equality	T1FF1FFF1	inside	1FF0FF212	f
Overlaps	T*T**T**	non-intersecting	FF1FF0212	f
Overlaps	T*T**T**	overlapping	1010F0212	t
Overlaps	T*T**T**	inside	1FF0FF212	f
Within	T**F***	non-intersecting	FF1FF0212	f
Within	T**F***	overlapping	1010F0212	f
Within	T**F***	inside	1FF0FF212	t
Disjoint	FF*FF***	non-intersecting	FF1FF0212	t
Disjoint	FF*FF***	overlapping	1010F0212	f
Disjoint	FF*FF***	inside	1FF0FF212	f

## Siehe auch

Section [5.1](#), [ST\\_Relate](#)

### 7.11.1.15 ST\_Touches

**ST\_Touches** — Prüft, ob zwei Geometrien mindestens einen Punkt gemeinsam haben, aber ihre Innenräume sich nicht schneiden

#### Synopsis

boolean **ST\_Touches**(geometry A, geometry B);

#### Beschreibung

Rückgabe TRUE wenn A und B sich schneiden, aber ihre Innenräume sich nicht schneiden. Äquivalent dazu haben A und B mindestens einen Punkt gemeinsam, und die gemeinsamen Punkte liegen in mindestens einer Begrenzung. Bei Punkt/Punkt-Eingaben ist die Beziehung immer FALSE, da Punkte keine Begrenzung haben.

In mathematical terms:  $ST\_Touches(A, B) \Leftrightarrow (Int(A) \cap Int(B) = \emptyset) \wedge (A \cap B \neq \emptyset)$

Diese Beziehung ist gegeben, wenn die DE-9IM-Schnittpunktmatrix für die beiden Geometrien einem der folgenden Werte entspricht:

- FT\*\*\*\*\*
- F\*\*T\*\*\*\*\*
- F\*\*\*T\*\*\*\*

**Note**

Diese Funktion beinhaltet automatisch einen Bounding-Box-Vergleich, der alle räumlichen Indizes verwendet, die für die Geometrien verfügbar sind. Um die Verwendung eines Indexes zu vermeiden, verwenden Sie stattdessen `_ST_Touches`.

**Important**

Verbessert: 3.0.0 ermöglicht die Unterstützung von `GEOMETRYCOLLECTION`



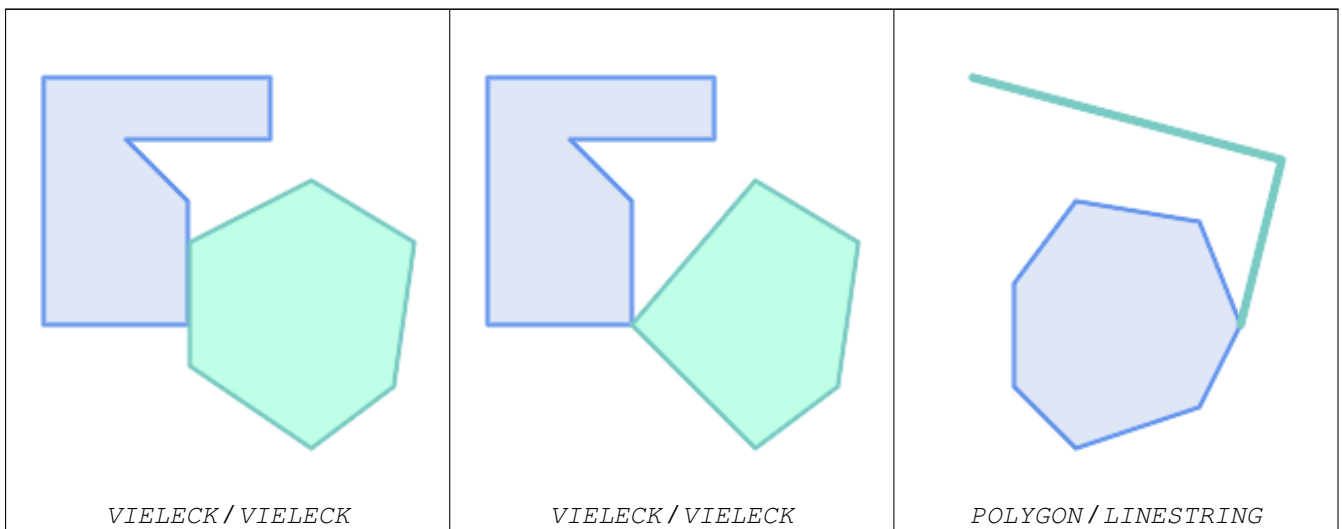
Diese Methode implementiert die [OGC Simple Features Implementation Specification for SQL 1.1](#). s2.1.1.2 // s2.1.13.3

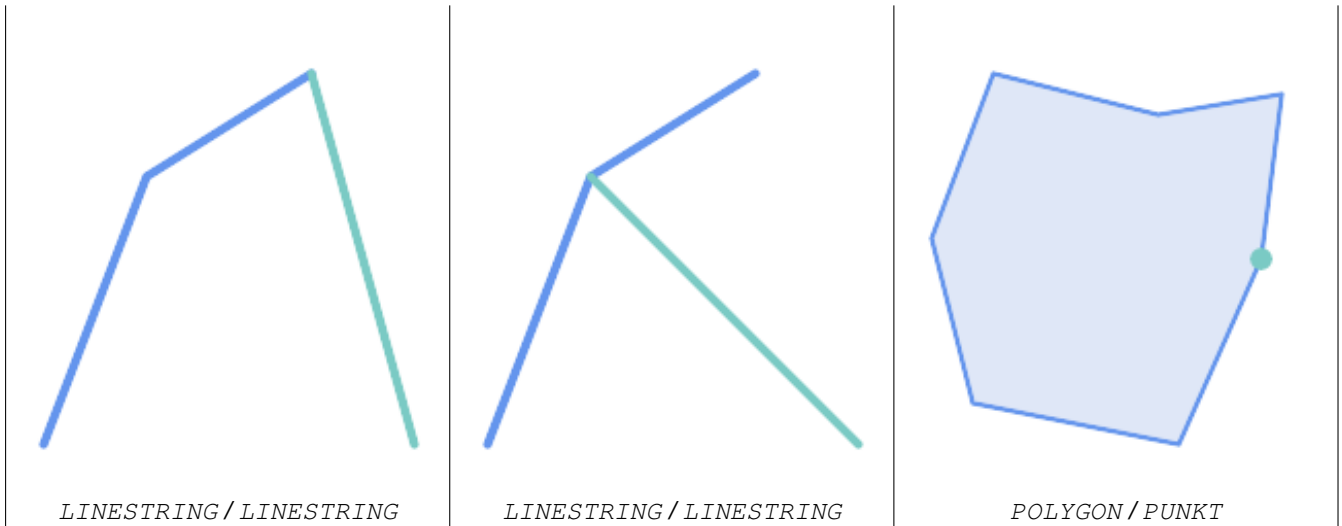


Diese Methode setzt die SQL/MM-Spezifikation um. SQL-MM 3: 5.1.28

**Beispiele**

Das Prädikat `ST_Touches` liefert `TRUE` in den folgenden Beispielen.





```
SELECT ST_Touches('LINestring(0 0, 1 1, 0 2)::geometry, 'POINT(1 1)::geometry');
st_touches
-----
f
(1 row)

SELECT ST_Touches('LINestring(0 0, 1 1, 0 2)::geometry, 'POINT(0 2)::geometry');
st_touches
-----
t
(1 row)
```

### 7.11.1.16 ST\_Within

**ST\_Within** — Tests, wenn jeder Punkt von A in B liegt und ihre Innenräume einen gemeinsamen Punkt haben

#### Synopsis

boolean **ST\_Within**(geometry A, geometry B);

#### Beschreibung

Gibt TRUE zurück, wenn Geometrie A innerhalb von Geometrie B liegt. A liegt nur dann innerhalb von B, wenn alle Punkte von A innerhalb (d. h. im Inneren oder am Rand) von B liegen (oder gleichwertig, keine Punkte von A liegen im Äußeren von B) und die Innenräume von A und B mindestens einen Punkt gemeinsam haben.

Damit diese Funktion sinnvoll ist, müssen die Ausgangsgeometrien beide die gleiche Koordinatenprojektion und den gleichen SRID haben.

Mathematisch ausgedrückt:  $ST\_Within(A, B) \Leftrightarrow (A \cap B = A) \wedge (Int(A) \cap Int(B) \neq \emptyset)$

Die Within-Relation ist reflexiv: jede Geometrie ist in sich selbst. Die Beziehung ist antisymmetrisch: Wenn  $ST\_Within(A, B) = true$  und  $ST\_Within(B, A) = true$ , dann müssen die beiden Geometrien topologisch gleich sein ( $ST\_Equals(A, B) = true$ ).

**ST\_Within** ist die Umkehrung von **ST\_Contains**.  $ST\_Within(A, B) = ST\_Contains(B, A)$ .



**Note**

Da die Innenräume einen gemeinsamen Punkt haben müssen, besteht eine Feinheit der Definition darin, dass Linien und Punkte, die vollständig in der Begrenzung von Polygonen oder Linien liegen, *nicht* innerhalb der Geometrie sind. Für weitere Details siehe [Subtleties of OGC Covers, Contains, Within](#). Das Prädikat `ST_CoveredBy` bietet eine umfassendere Beziehung.



**Note**

Diese Funktion beinhaltet automatisch einen Bounding-Box-Vergleich, der alle räumlichen Indizes verwendet, die für die Geometrien verfügbar sind. Um die Verwendung von Indizes zu vermeiden, verwenden Sie die Funktion `_ST_Within`.

Wird durch das GEOS Modul ausgeführt

Verbessert: 2.3.0 Verbesserung des PIP-Kurzschlusses für Geometrien, erweitert um die Unterstützung von MultiPoints mit wenigen Punkten. Frühere Versionen unterstützten nur Punkte in Polygonen.



**Important**

Verbessert: 3.0.0 ermöglicht die Unterstützung von `GEOMETRYCOLLECTION`



**Important**

Verwenden Sie diese Funktion nicht mit ungültigen Geometrien. Sie werden unerwartete Ergebnisse erhalten.

HINWEIS: Dies ist die "zulässige" Version, die einen booleschen Wert und keine ganze Zahl zurückgibt.



Diese Methode implementiert die [OGC Simple Features Implementation Specification for SQL 1.1](#). `s2.1.1.2 // s2.1.13.3 - a.Relate(b, "T*F**F**")`



Diese Methode setzt die SQL/MM-Spezifikation um. `SQL-MM 3: 5.1.30`

**Beispiele**

```
--a circle within a circle
SELECT ST_Within(smallc,smallc) As smallinsmall,
       ST_Within(smallc, bigc) As smallinbig,
       ST_Within(bigc,smallc) As biginsmall,
       ST_Within(ST_Union(smallc, bigc), bigc) as unioninbig,
       ST_Within(bigc, ST_Union(smallc, bigc)) as 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\\_CoveredBy](#), [ST\\_Equals](#), [ST\\_IsValid](#)

**7.11.2 Fernbeziehungen****7.11.2.1 ST\_3DDWithin**

ST\_3DDWithin — Prüft, ob zwei 3D-Geometrien innerhalb eines bestimmten 3D-Abstands liegen

**Synopsis**

```
boolean ST_3DDWithin(geometry g1, geometry g2, double precision distance_of_srid);
```

**Beschreibung**

Gibt true zurück, wenn der 3D-Abstand zwischen zwei Geometriewerten nicht größer ist als `distance_of_srid`. Der Abstand wird in Einheiten angegeben, die durch das räumliche Bezugssystem der Geometrien definiert sind. Damit diese Funktion sinnvoll ist, müssen sich die Ausgangsgeometrien im selben Koordinatensystem befinden (denselben SRID haben).

**Note**

Diese Funktion beinhaltet automatisch einen Bounding-Box-Vergleich, der alle räumlichen Indizes verwendet, die für die Geometrien verfügbar sind.



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



Diese Funktion unterstützt polyedrische Flächen.



Diese Methode setzt die SQL/MM-Spezifikation um. SQL-MM ?

Verfügbarkeit: 2.0.0

## Beispiele

```
-- Geometry example - units in meters (SRID: 2163 US National Atlas Equal area) (3D point ↔
  and line compared 2D point and line)
-- Note: currently no vertical datum support so Z is not transformed and assumed to be same ↔
  units as final.
SELECT ST_3DDWithin(
  ST_Transform(ST_GeomFromEWKT('SRID=4326;POINT(-72.1235 42.3521 4)'),2163),
  ST_Transform(ST_GeomFromEWKT('SRID=4326;LINESTRING(-72.1260 42.45 15, -72.123 42.1546 ↔
    20)'),2163),
  126.8
) As within_dist_3d,
ST_DWithin(
  ST_Transform(ST_GeomFromEWKT('SRID=4326;POINT(-72.1235 42.3521 4)'),2163),
  ST_Transform(ST_GeomFromEWKT('SRID=4326;LINESTRING(-72.1260 42.45 15, -72.123 42.1546 ↔
    20)'),2163),
  126.8
) As within_dist_2d;

within_dist_3d | within_dist_2d
-----+-----
f              | t
```

## Siehe auch

[ST\\_3DDFullyWithin](#), [ST\\_DWithin](#), [ST\\_DFullyWithin](#), [ST\\_3DDistance](#), [ST\\_Distance](#), [ST\\_3DMaxDistance](#), [ST\\_Transform](#)

### 7.11.2.2 ST\_3DDFullyWithin

ST\_3DDFullyWithin — Prüft, ob zwei 3D-Geometrien vollständig innerhalb eines bestimmten 3D-Abstands liegen

## Synopsis

boolean **ST\_3DDFullyWithin**(geometry g1, geometry g2, double precision distance);

## Beschreibung

Gibt true zurück, wenn die 3D-Geometrien vollständig innerhalb des angegebenen Abstands zueinander liegen. Der Abstand wird in Einheiten angegeben, die durch das räumliche Bezugssystem der Geometrien definiert sind. Damit diese Funktion sinnvoll ist, müssen die Ausgangsgeometrien beide der gleichen Koordinatenprojektion angehören und den gleichen SRID haben.



### Note

Diese Funktion beinhaltet automatisch einen Bounding-Box-Vergleich, der alle räumlichen Indizes verwendet, die für die Geometrien verfügbar sind.

Verfügbarkeit: 2.0.0



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



Diese Funktion unterstützt polyedrische Flächen.



**Beispiele**

```
-- This compares the difference between fully within and distance within as well
-- as the distance fully within for the 2D footprint of the line/point vs. the 3d fully
-- within
SELECT ST_3DDFullyWithin(geom_a, geom_b, 10) as D3DFullyWithin10, ST_3DDWithin(geom_a,
geom_b, 10) as D3DWithin10,
ST_DFullyWithin(geom_a, geom_b, 20) as D2DFullyWithin20,
ST_3DDFullyWithin(geom_a, geom_b, 20) as D3DFullyWithin20 from
(select ST_GeomFromEWKT('POINT(1 1 2)') as geom_a,
ST_GeomFromEWKT('LINESTRING(1 5 2, 2 7 20, 1 9 100, 14 12 3)') as geom_b) t1;
d3dfullywithin10 | d3dwithin10 | d2dfullywithin20 | d3dfullywithin20
-----+-----+-----+-----
f | t | t | f
```

**Siehe auch**

[ST\\_3DDWithin](#), [ST\\_DWithin](#), [ST\\_DFullyWithin](#), [ST\\_3DMaxDistance](#)

**7.11.2.3 ST\_DFullyWithin**

ST\_DFullyWithin — Tests if a geometry is entirely inside a distance of another

**Synopsis**

boolean **ST\_DFullyWithin**(geometry g1, geometry g2, double precision distance);

**Beschreibung**

Returns true if g2 is entirely within distance of g1. Visually, the condition is true if g2 is contained within a distance buffer of g1. The distance is specified in units defined by the spatial reference system of the geometries.



**Note**

Diese Funktion beinhaltet automatisch einen Bounding-Box-Vergleich, der alle räumlichen Indizes verwendet, die für die Geometrien verfügbar sind.

Verfügbarkeit: 1.5.0

Changed: 3.5.0 : the logic behind the function now uses a test of containment within a buffer, rather than the ST\_MaxDistance algorithm. Results will differ from prior versions, but should be closer to user expectations.

**Beispiele**

```
SELECT
  ST_DFullyWithin(geom_a, geom_b, 10) AS DFullyWithin10,
  ST_DWithin(geom_a, geom_b, 10) AS DWithin10,
  ST_DFullyWithin(geom_a, geom_b, 20) AS DFullyWithin20
FROM (VALUES
  ('POINT(1 1)', 'LINESTRING(1 5, 2 7, 1 9, 14 12)')
) AS v(geom_a, geom_b)

dfullywithin10 | dwithin10 | dfullywithin20
-----+-----+-----
f | t | t
```

## Siehe auch

[ST\\_MaxDistance](#), [ST\\_DWithin](#), [ST\\_3DDWithin](#), [ST\\_3DDFullyWithin](#)

### 7.11.2.4 ST\_DWithin

ST\_DWithin — Prüft, ob zwei Geometrien innerhalb eines bestimmten Abstands liegen

## Synopsis

boolean **ST\_DWithin**(geometry g1, geometry g2, double precision distance\_of\_srid);

boolean **ST\_DWithin**(geography gg1, geography gg2, double precision distance\_meters, boolean use\_spheroid = true);

## Beschreibung

Gibt true zurück, wenn die Geometrien innerhalb eines bestimmten Abstands liegen

Für geometry: Der Abstand wird in Einheiten angegeben, die durch das räumliche Bezugssystem der Geometrien definiert sind. Damit diese Funktion sinnvoll ist, müssen sich die Ausgangsgeometrien im selben Koordinatensystem befinden (denselben SRID haben).

Für Geografie: Die Einheiten sind in Metern und die Entfernungsmessung ist standardmäßig auf `use_spheroid = true` eingestellt. Für eine schnellere Auswertung verwenden Sie `use_spheroid = false`, um auf der Kugel zu messen.



### Note

Verwenden Sie [ST\\_3DDWithin](#) für 3D-Geometrien.



### Note

Dieser Funktionsaufruf beinhaltet einen Bounding-Box-Vergleich, der alle für die Geometrien verfügbaren Indizes verwendet.



Diese Methode implementiert die [OGC Simple Features Implementation Specification for SQL 1.1](#).

Verfügbarkeit: Mit Version 1.5.0 wurde die Unterstützung für Geografie eingeführt.

Verbessert: 2.1.0 verbesserte die Geschwindigkeit für Geographie. Siehe [Geografie schneller machen](#) für Details.

Verbessert: 2.1.0 Unterstützung für gekrümmte Geometrien wurde eingeführt.

Vor 1.3 wurde [ST\\_Expand](#) üblicherweise in Verbindung mit `&&` und `ST_Distance` verwendet, um den Abstand zu testen, und vor 1.3.4 verwendete diese Funktion diese Logik. Ab 1.3.4 verwendet `ST_DWithin` eine schnellere Kurzschluss-Abstandsfunktion.

## Beispiele

```
-- Find the nearest hospital to each school
-- that is within 3000 units of the school.
-- We do an ST_DWithin search to utilize indexes to limit our search list
-- that the non-indexable ST_Distance needs to process
-- If the units of the spatial reference is meters then units would be meters
SELECT DISTINCT ON (s.gid) s.gid, s.school_name, s.geom, h.hospital_name
FROM schools s
LEFT JOIN hospitals h ON ST_DWithin(s.geom, h.geom, 3000)
ORDER BY s.gid, ST_Distance(s.geom, h.geom);
```

```
-- The schools with no close hospitals
-- Find all schools with no hospital within 3000 units
-- away from the school. Units is in units of spatial ref (e.g. meters, feet, degrees)
SELECT s.gid, s.school_name
FROM schools s
LEFT JOIN hospitals h ON ST_DWithin(s.geom, h.geom, 3000)
WHERE h.gid IS NULL;

-- Find broadcasting towers that receiver with limited range can receive.
-- Data is geometry in Spherical Mercator (SRID=3857), ranges are approximate.

-- Create geometry index that will check proximity limit of user to tower
CREATE INDEX ON broadcasting_towers using gist (geom);

-- Create geometry index that will check proximity limit of tower to user
CREATE INDEX ON broadcasting_towers using gist (ST_Expand(geom, sending_range));

-- Query towers that 4-kilometer receiver in Minsk Hackerspace can get
-- Note: two conditions, because shorter LEAST(b.sending_range, 4000) will not use index.
SELECT b.tower_id, b.geom
FROM broadcasting_towers b
WHERE ST_DWithin(b.geom, 'SRID=3857;POINT(3072163.4 7159374.1)', 4000)
AND ST_DWithin(b.geom, 'SRID=3857;POINT(3072163.4 7159374.1)', b.sending_range);
```

## Siehe auch

[ST\\_Distance](#), [ST\\_3DDWithin](#)

### 7.11.2.5 ST\_PointInsideCircle

**ST\_PointInsideCircle** — Prüft, ob ein geometrischer Punkt innerhalb eines Kreises liegt, der durch einen Mittelpunkt und einen Radius definiert ist

#### Synopsis

boolean **ST\_PointInsideCircle**(geometry a\_point, float center\_x, float center\_y, float radius);

#### Beschreibung

Gibt true zurück, wenn die Geometrie ein Punkt ist und innerhalb des Kreises mit Zentrum `center_x`, `center_y` und Radius `radius` liegt.



#### Warning

Verwendet keine räumlichen Indizes. Verwenden Sie stattdessen [ST\\_DWithin](#).

---

Verfügbarkeit: 1.2

Geändert: 2.2.0 In früheren Versionen hieß dies `ST_Point_Inside_Circle`

---

## Beispiele

```
SELECT ST_PointInsideCircle(ST_Point(1,2), 0.5, 2, 3);
  st_pointinsidecircle
-----
t
```

## Siehe auch

[ST\\_DWithin](#)

## 7.12 Messfunktionen

### 7.12.1 ST\_Area

`ST_Area` — Gibt den geometrischen Schwerpunkt einer Geometrie zurück.

#### Synopsis

```
float ST_Area(geometry g1);
float ST_Area(geography geog, boolean use_spheroid = true);
```

#### Beschreibung

Gibt den Flächeninhalt von Polygonen und Mehrfachpolygonen zurück. Gibt den Flächeninhalt der Datentypen "ST\_Surface" und "ST\_MultiSurface" zurück. Beim geometrischen Datentyp wird die kartesische 2D-Fläche ermittelt und in den Einheiten des SRID ausgegeben. Beim geographischen Datentyp wird die Fläche standardmäßig auf einem Referenzellipsoid ermittelt und in Quadratmeter ausgegeben. Mit `ST_Area(geog,false)` kann der Flächeninhalt auf einer Kugel ermittelt werden; dies ist zwar schneller aber auch weniger genau.

Erweiterung: Mit 2.0.0 wurde 2D-Unterstützung für polyedrische Oberflächen eingeführt.

Erweiterung: 2.2.0 - die Messung auf dem Referenzellipsoid wird mit der Bibliothek "GeographicLib" durchgeführt. Dadurch wurde die Genauigkeit und die Robustheit erhöht. Um die Vorteile dieser neuen Funktionalität zu nutzen, benötigen Sie Proj >= 4.9.0.

Geändert: 3.0.0 - hängt nicht mehr von SFCGAL ab.



Diese Methode implementiert die [OGC Simple Features Implementation Specification for SQL 1.1](#).



Diese Methode setzt die SQL/MM-Spezifikation um. SQL-MM 3: 8.1.2, 9.5.3



Diese Funktion unterstützt polyedrische Flächen.



#### Note

Bei polyedrischen Oberflächen wird nur 2D (nicht 2.5D) unterstützt. Bei 2.5D kann ein Ergebnis ungleich null geliefert werden, wenn die Oberflächen vollständig in der XY-Ebene liegen.

## Beispiele

Gibt den Flächeninhalt eines Grundstücks in Massachusetts - in Quadratfuß und konvertiert in Quadratmeter - zurück. Anmerkung: Wegen "Massachusetts State Plane Feet" (EPSG:2249) wird der Flächeninhalt in Quadratfuß ausgegeben

```
select ST_Area(geom) sqft,
       ST_Area(geom) * 0.3048 ^ 2 sqm
from (
  select 'SRID=2249;POLYGON((743238 2967416,743238 2967450,
                            743265 2967450,743265.625 2967416,743238 2967416))' :: geometry geom
) subquery;
sqft      sqm
928.625   86.27208552
928.625   86.27208552
```

Gibt den Flächeninhalt in Quadratfuß aus und transformiert nach "Massachusetts state plane meters" (EPSG:26986) um Quadratmeter zu erhalten. Da die Fläche in "Massachusetts State Plane Feet" (EPSG:2249) vorliegt, wird der Flächeninhalt in Quadratfuß ausgegeben. Die transformierte Fläche ist in Quadratmeter, da sie in EPSG:26986 "Massachusetts state plane meters" (EPSG:26986) vorliegt.

```
select ST_Area(geom) sqft,
       ST_Area(ST_Transform(geom, 26986)) As sqm
from (
  select
    'SRID=2249;POLYGON((743238 2967416,743238 2967450,
                        743265 2967450,743265.625 2967416,743238 2967416))' :: geometry geom
) subquery;
sqft      sqm
928.625   86.272430607008
928.625   86.272430607008
```

Gibt den Flächeninhalt in Quadratfuß und in Quadratmeter für den geographischen Datentyp zurück. Beachten Sie bitte, dass wir den geometrischen in den geographischen Datentyp umwandeln (dafür muss die Geometrie in WGS84 lon lat 4326 vorliegen). Beim geographischen Datentyp wird immer in Meter gemessen. Dies ist nur für Vergleichszwecke gedacht, da Ihre Tabelle üblicherweise bereits den geographischen Datentyp aufweisen wird.

```
select ST_Area(geog) / 0.3048 ^ 2 sqft_spheroid,
       ST_Area(geog, false) / 0.3048 ^ 2 sqft_sphere,
       ST_Area(geog) sqm_spheroid
from (
  select ST_Transform(
    'SRID=2249;POLYGON((743238 2967416,743238 2967450,743265
                        2967450,743265.625 2967416,743238 2967416))' :: geometry,
    4326
  ) :: geography geog
) as subquery;
sqft_spheroid sqft_sphere sqm_spheroid
928.684405784452 927.049336105925 86.2776044979692
928.684405784452 927.049336105925 86.2776044979692
```

Wenn Ihre Daten bereits in der Geografie enthalten sind:

```
select ST_Area(geog) / 0.3048 ^ 2 sqft,
       ST_Area(the_geog) sqm
from somegeogtable;
```

**Siehe auch**

[ST\\_3DArea](#), [ST\\_GeomFromEWKT](#), [ST\\_LengthSpheroid](#), [ST\\_Perimeter](#), [ST\\_Transform](#)

**7.12.2 ST\_Azimuth**

`ST_Azimuth` — Gibt die 2-dimensionale kürzeste Strecke zwischen zwei Geometrien als Linie zurück

**Synopsis**

```
float ST_Azimuth(geometry origin, geometry target);
float ST_Azimuth(geography origin, geography target);
```

**Beschreibung**

Gibt den Azimutwinkel des Zielpunkts vom Ursprungspunkt in Bogenmaß zurück oder NULL, wenn die beiden Punkte übereinstimmen. Der Azimutwinkel ist ein positiver Winkel im Uhrzeigersinn, der sich auf die positive Y-Achse (Geometrie) oder den Nordmeridian (Geografie) bezieht: Nord = 0; Nordost =  $\pi/4$ ; Ost =  $\pi/2$ ; Südost =  $3\pi/4$ ; Süd =  $\pi$ ; Südwest  $5\pi/4$ ; West =  $3\pi/2$ ; Nordwest =  $7\pi/4$ .

Für den Typ Geografie ist die Azimutlösung als [inverses geodätisches Problem](#) bekannt.

Der Azimut ist ein mathematisches Konzept, das als der Winkel zwischen einem Referenzvektor und einem Punkt definiert ist, mit Winkleinheiten im Bogenmaß. Der Ergebniswert in Bogenmaß kann mit der PostgreSQL-Funktion `degrees()` in Grad umgerechnet werden.

Der Azimut ist in Verbindung mit `ST_Translate` besonders nützlich, weil damit ein Objekt entlang seiner rechtwinkligen Achse verschoben werden kann. Siehe dazu die Funktion "upgis\_lineshift", in dem Abschnitt [Ppgsqfunctions des PostGIS Wiki](#), für ein Beispiel.

Verfügbarkeit: 1.1.0

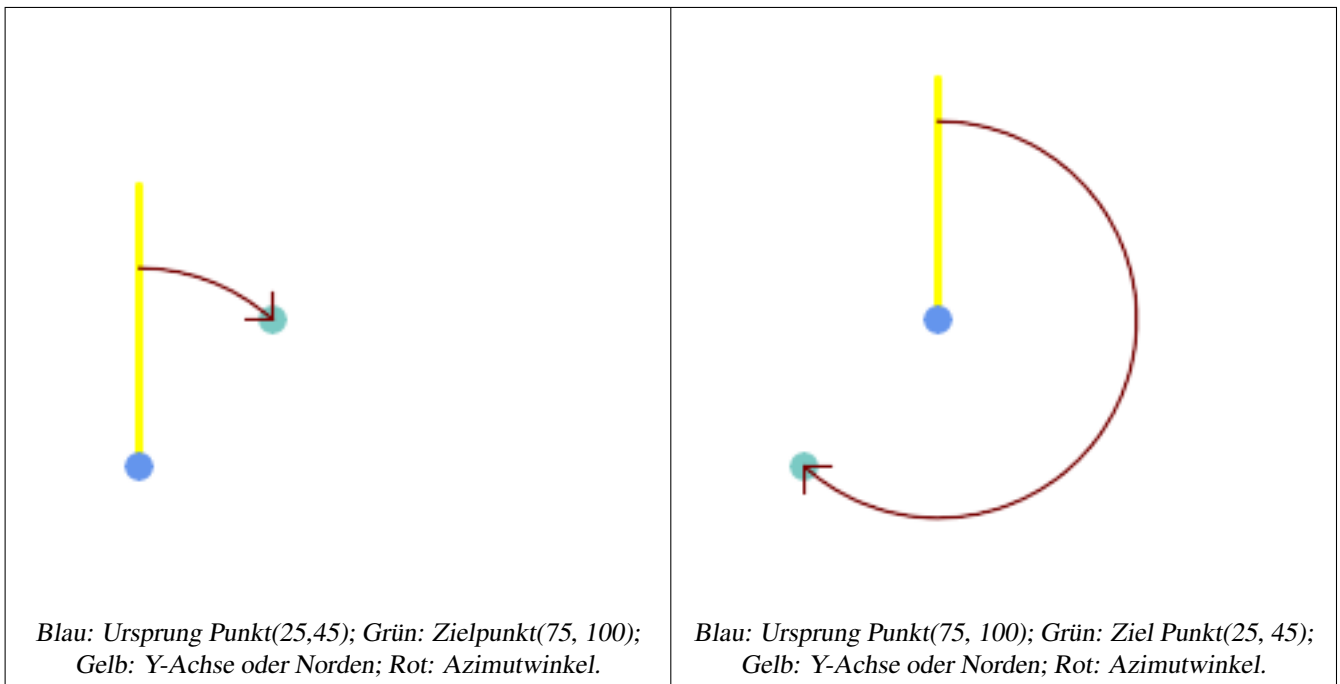
Erweiterung: mit 2.0.0 wurde die Unterstützung des geographischen Datentyps eingeführt.

Erweiterung: 2.2.0 die Messungen auf dem Referenzellipsoid werden mit der Bibliothek "GeographicLib" durchgeführt. Dadurch wurde die Genauigkeit und die Robustheit erhöht. Um die Vorteile dieser neuen Funktionalität zu nutzen, benötigen Sie Proj >= 4.9.0.

**Beispiele****Geometrischer Datentyp - Azimut in Grad**

```
SELECT degrees(ST_Azimuth( ST_Point(25, 45), ST_Point(75, 100))) AS degA_B,
       degrees(ST_Azimuth( ST_Point(75, 100), ST_Point(25, 45) )) AS degB_A;
```

dega_b	degb_a
42.2736890060937	222.273689006094

**Siehe auch**

[ST\\_Angle](#), [ST\\_Translate](#), [ST\\_Project](#), [PostgreSQL Math Functions](#)

**7.12.3 ST\_Angle**

`ST_Angle` — Gibt den Winkel zwischen 3 Punkten oder zwischen 2 Vektoren (4 Punkte oder 2 Linien) zurück.

**Synopsis**

```
float ST_Angle(geometry point1, geometry point2, geometry point3, geometry point4);
float ST_Angle(geometry line1, geometry line2);
```

**Beschreibung**

Gibt die größte 3-dimensionale Distanz zwischen zwei geometrischen Objekten als Linie zurück

**Variante 1:** berechnet den Winkel, der von den Punkten P1-P2-P3 eingeschlossen wird. Wenn ein 4. Punkt vorgesehen ist, werden die Winkel P1-P2 und P3-P4 berechnet

**Variante 2:** berechnet den Winkel zwischen zwei Vektoren S1-E1 und S2-E2, die durch die Anfangs- und Endpunkte der Eingabelinien definiert sind

Das Ergebnis wird in Radiant ausgegeben. Wie im folgenden Beispiel gezeigt, kann mit der in PostgreSQL integrierten Funktion "degrees()" von der Einheit Radiant auf die Einheit Grad umgerechnet werden.

```
ST_Angle(P1,P2,P3) = ST_Angle(P2,P1,P2,P3)
```

Verfügbarkeit: 2.5.0

## Beispiele

### Längste Strecke zwischen Polygon und Polygon

```
SELECT degrees( ST_Angle('POINT(0 0)', 'POINT(10 10)', 'POINT(20 0)') );

degrees
-----
      270
```

### Winkel zwischen Vektoren, die durch vier Punkte definiert sind

```
SELECT degrees( ST_Angle('POINT (10 10)', 'POINT (0 0)', 'POINT(90 90)', 'POINT (100 80)') ←
);

degrees
-----
269.99999999999999
```

### Winkel zwischen Vektoren, die durch die Anfangs- und Endpunkte von Linien definiert sind

```
SELECT degrees( ST_Angle('LINESTRING(0 0, 0.3 0.7, 1 1)', 'LINESTRING(0 0, 0.2 0.5, 1 0)') ←
);

degrees
-----
      45
```

## Siehe auch

[ST\\_Azimuth](#)

## 7.12.4 ST\_ClosestPoint

`ST_ClosestPoint` — Gibt den 2D-Punkt auf `g1` zurück, der `g2` am nächsten ist. Dies ist der erste Punkt der kürzesten Linie von einer Geometrie zur anderen.

### Synopsis

```
geometry ST_ClosestPoint(geometry geom1, geometry geom2);
geography ST_ClosestPoint(geography geom1, geography geom2, boolean use_spheroid = true);
```

### Beschreibung

Gibt den 2-dimensionalen Punkt auf `geom1` zurück, der `geom2` am nächsten liegt. Dies ist der erste Punkt der kürzesten Linie zwischen den Geometrien (wie von [ST\\_ShortestLine](#) berechnet).



#### Note

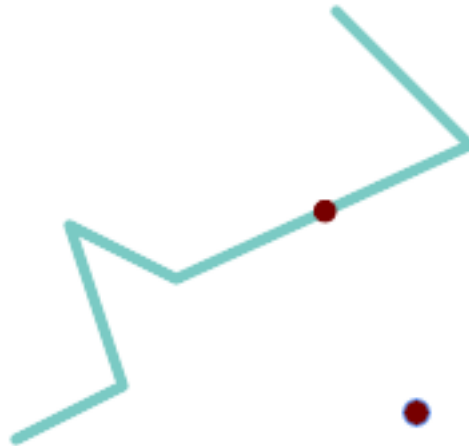
Falls es sich um eine 3D-Geometrie handelt, sollten Sie [ST\\_3DClosestPoint](#) vorziehen.

Verbessert: 3.4.0 - Unterstützung für Geographie.

Verfügbarkeit: 1.5.0



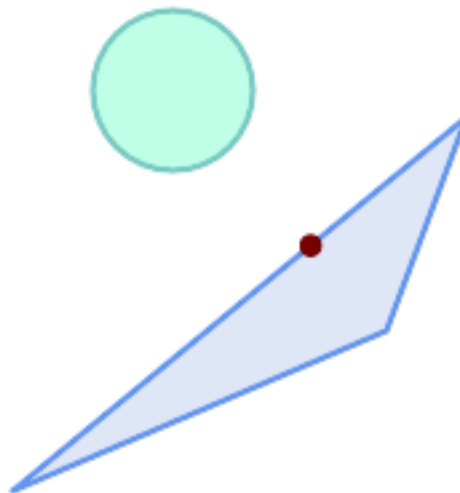
## Beispiele



*Der nächstgelegene Punkt für einen Point und einen LineString ist der Punkt selbst. Der nächstgelegene Punkt für einen LineString und einen Punkt ist ein Punkt auf der Linie.*

```
SELECT ST_AsText( ST_ClosestPoint(pt,line)) AS cp_pt_line,
       ST_AsText( ST_ClosestPoint(line,pt)) AS cp_line_pt
FROM (SELECT 'POINT (160 40)::geometry AS pt,
            'LINESTRING (10 30, 50 50, 30 110, 70 90, 180 140, 130 190)::geometry AS line ) AS t;
```

cp_pt_line	cp_line_pt
POINT(160 40)	POINT(125.75342465753425 115.34246575342466)



*Der Punkt auf Polygon A, der Polygon B am nächsten liegt*

```
SELECT ST_AsText( ST_ClosestPoint(
    'POLYGON ((190 150, 20 10, 160 70, 190 150))',
    ST_Buffer('POINT(80 160)', 30)
)) AS ptwkt;
-----
POINT(131.59149149528952 101.89887534906197)
```

**Siehe auch**

[ST\\_3DClosestPoint](#), [ST\\_Distance](#), [ST\\_LongestLine](#), [ST\\_ShortestLine](#), [ST\\_MaxDistance](#)

**7.12.5 ST\_3DClosestPoint**

ST\_3DClosestPoint — Gibt den 3-dimensionalen Punkt auf g1 zurück, der den kürzesten Abstand zu g2 hat. Dies ist der Anfangspunkt des kürzesten Abstands in 3D.

**Synopsis**

geometry **ST\_3DClosestPoint**(geometry g1, geometry g2);

**Beschreibung**

Gibt den 3-dimensionalen Punkt auf g1 zurück, der den kürzesten Abstand zu g2 hat. Dies ist der Anfangspunkt des kürzesten Abstands in 3D. Die Länge des kürzesten Abstands in 3D ergibt sich aus der 3D-Distanz.



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



Diese Funktion unterstützt polyedrische Flächen.

Verfügbarkeit: 2.0.0

Änderung: 2.2.0 - Wenn 2 geometrische Objekte in 2D übergeben werden, wird ein 2D-Punkt zurückgegeben (anstelle wie früher 0 für ein fehlendes Z). Im Falle von 2D und 3D, wird für fehlende Z nicht länger 0 angenommen.

**Beispiele**

```

Linienstück und Punkt -- Punkt mit kürzestem Abstand; in 3D und in 2D

SELECT ST_AsEWKT(ST_3DClosestPoint(line,pt)) AS cp3d_line_pt,
       ST_AsEWKT(ST_ClosestPoint(line,pt)) As cp2d_line_pt
FROM (SELECT 'POINT(100 100 30)::geometry As pt,
            'LINESTRING (20 80 20, 98 190 1, 110 180 3, 50 75 1000)::' ←
       geometry As line
       ) As foo;

cp3d_line_pt | cp2d_line_pt
-----+-----
POINT(54.69937988867619 128.935022917228 11.5475869506606) | POINT(73.0769230769231 ←
115.384615384615)

Linienstück und Mehrfachpunkt - Punkt mit kürzestem Abstand; in 3D und in 2D

SELECT ST_AsEWKT(ST_3DClosestPoint(line,pt)) AS cp3d_line_pt,
       ST_AsEWKT(ST_ClosestPoint(line,pt)) As cp2d_line_pt
FROM (SELECT 'MULTIPOINT(100 100 30, 50 74 1000)::geometry As pt,
            'LINESTRING (20 80 20, 98 190 1, 110 180 3, 50 75 900)::' ←
       geometry As line
       ) As foo;

cp3d_line_pt | cp2d_line_pt
-----+-----
POINT(54.69937988867619 128.935022917228 11.5475869506606) | POINT(50 75)
    
```

**Mehrfachlinie und Polygon - Punkt mit kürzestem Abstand; in 3D und in 2D**

```

SELECT ST_AsEWKT(ST_3DClosestPoint(poly, mline)) As cp3d,
       ST_AsEWKT(ST_ClosestPoint(poly, mline)) As cp2d
  FROM (SELECT ST_GeomFromEWKT('POLYGON((175 150 5, 20 40 5, 35 45 5, 50 60 5, ←
        100 100 5, 175 150 5))') As poly,
        ST_GeomFromEWKT('MULTILINESTRING((175 155 2, 20 40 20, 50 60 -2, 125 ←
        100 1, 175 155 1),
        (1 10 2, 5 20 1))') As mline ) As foo;
-----+-----
POINT(39.993580415989 54.1889925532825 5) | POINT(20 40)

```

**Siehe auch**

[ST\\_AsEWKT](#), [ST\\_ClosestPoint](#), [ST\\_3DDistance](#), [ST\\_3DShortestLine](#)

**7.12.6 ST\_Distance**

`ST_Distance` — Gibt die größte 3-dimensionale Distanz zwischen zwei geometrischen Objekten als Linie zurück

**Synopsis**

```

float ST_Distance(geometry g1, geometry g2);
float ST_Distance(geography geog1, geography geog2, boolean use_spheroid = true);

```

**Beschreibung**

Für den geometrischen Datentyp. Es wird der geringste 3-dimensionale kartesische Abstand zwischen zwei geometrischen Objekten in projizierten Einheiten (Einheiten des Koordinatenreferenzsystem) zurückgegeben.

Beim geometrischen Datentyp **geometry** wird die geringste kartesische Distanz in 2D zwischen zwei geometrischen Objekten - in projizierten Einheiten (Einheiten des Koordinatenreferenzsystem) - zurückgegeben. Beim geographischen Datentyp **geography** wird standardmäßig die geringste geodätische Distanz zwischen zwei geographischen Objekten in Meter zurückgegeben. Wenn `use_spheroid FALSE` ist, erfolgt eine schnellere Berechnung auf einer Kugel anstatt auf dem Referenzellipsoid.



Diese Methode implementiert die [OGC Simple Features Implementation Specification for SQL 1.1](#).



Diese Methode setzt die SQL/MM-Spezifikation um. SQL-MM 3: 5.1.23



Diese Methode unterstützt kreisförmige Strings und Kurven.

Verfügbarkeit: 1.5.0 die Unterstützung des geographischen Datentyps wurde eingeführt. Geschwindigkeitsverbesserungen bei einer umfangreichen Geometrie und bei einer Geometrie mit vielen Knoten

Enhanced: 2.1.0 Geschwindigkeitsverbesserung beim geographischen Datentyp. Siehe [Making Geography faster](#) für Details.

Erweiterung: 2.1.0 - Unterstützung für Kurven beim geometrischen Datentyp eingeführt.

Erweiterung: 2.2.0 - die Messung auf dem Referenzellipsoid wird mit der Bibliothek "GeographicLib" durchgeführt. Dadurch wurde die Genauigkeit und die Robustheit erhöht. Um die Vorteile dieser neuen Funktionalität zu nutzen, benötigen Sie Proj >= 4.9.0.

Geändert: 3.0.0 - hängt nicht mehr von SFCGAL ab.

## Beispiele mit dem geometrischen Datentyp

Geometriebeispiel - Einheiten in Grad 4326 ist WGS 84 long lat, Einheiten sind Grad.

```
SELECT ST_Distance(
  'SRID=4326;POINT(-72.1235 42.3521)::geometry,
  'SRID=4326;LINESTRING(-72.1260 42.45, -72.123 42.1546)::geometry );
-----
0.00150567726382282
```

Geometriebeispiel - Einheiten in Metern (SRID: 3857, proportional zu Pixeln auf gängigen Webkarten). Obwohl der Wert nicht stimmt, können nahe gelegene Werte korrekt verglichen werden, was ihn zu einer guten Wahl für Algorithmen wie KNN oder KMeans macht.

```
SELECT ST_Distance(
  ST_Transform('SRID=4326;POINT(-72.1235 42.3521)::geometry, 3857),
  ST_Transform('SRID=4326;LINESTRING(-72.1260 42.45, -72.123 42.1546)::geometry, 3857) ) ←
  ;
-----
167.441410065196
```

Geometriebeispiel - Einheiten in Metern (SRID: 3857 wie oben, aber korrigiert um  $\cos(\text{lat})$ , um die Verzerrung zu berücksichtigen)

```
SELECT ST_Distance(
  ST_Transform('SRID=4326;POINT(-72.1235 42.3521)::geometry, 3857),
  ST_Transform('SRID=4326;LINESTRING(-72.1260 42.45, -72.123 42.1546)::geometry, 3857)
  ) * cosd(42.3521);
-----
123.742351254151
```

Geometriebeispiel - Einheiten in Metern (SRID: 26986 Massachusetts state plane meters) (am genauesten für Massachusetts)

```
SELECT ST_Distance(
  ST_Transform('SRID=4326;POINT(-72.1235 42.3521)::geometry, 26986),
  ST_Transform('SRID=4326;LINESTRING(-72.1260 42.45, -72.123 42.1546)::geometry, 26986) ←
  );
-----
123.797937878454
```

Geometriebeispiel - Einheiten in Metern (SRID: 2163 US National Atlas Equal area) (am wenigsten genau)

```
SELECT ST_Distance(
  ST_Transform('SRID=4326;POINT(-72.1235 42.3521)::geometry, 2163),
  ST_Transform('SRID=4326;LINESTRING(-72.1260 42.45, -72.123 42.1546)::geometry, 2163) ) ←
  ;
-----
126.664256056812
```

## Beispiele für den geographischen Datentyp

Wie im Geometrie-Beispiel, aber beachten Sie die Einheiten in Metern - verwenden Sie die Kugel für eine etwas schnellere und weniger genaue Berechnung.

```
SELECT ST_Distance(gg1, gg2) As spheroid_dist, ST_Distance(gg1, gg2, false) As sphere_dist
FROM (SELECT
  'SRID=4326;POINT(-72.1235 42.3521)::geography as gg1,
  'SRID=4326;LINESTRING(-72.1260 42.45, -72.123 42.1546)::geography as gg2
  ) As foo ;
```

spheroid_dist		sphere_dist
123.802076746848		123.475736916397

**Siehe auch**

[ST\\_3DDistance](#), [ST\\_DWithin](#), [ST\\_DistanceSphere](#), [ST\\_DistanceSpheroid](#), [ST\\_MaxDistance](#), [ST\\_HausdorffDistance](#), [ST\\_FrechetDistance](#), [ST\\_Transform](#)

**7.12.7 ST\_3DDistance**

**ST\_3DDistance** — Für den geometrischen Datentyp. Es wird der geringste 3-dimensionale kartesische Abstand (basierend auf dem Koordinatenreferenzsystem) zwischen zwei geometrischen Objekten in projizierten Einheiten zurückgegeben.

**Synopsis**

```
float ST_3DDistance(geometry g1, geometry g2);
```

**Beschreibung**

Für den geometrischen Datentyp. Es wird der geringste 3-dimensionale kartesische Abstand zwischen zwei geometrischen Objekten in projizierten Einheiten (Einheiten des Koordinatenreferenzsystem) zurückgegeben.



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



Diese Funktion unterstützt polyedrische Flächen.



Diese Methode setzt die SQL/MM-Spezifikation um. SQL-MM ISO/IEC 13249-3

Verfügbarkeit: 2.0.0

Änderung: 2.2.0 - Im Falle von 2D und 3D wird für ein fehlendes Z nicht mehr 0 angenommen.

Geändert: 3.0.0 - SFCGAL-Version entfernt

**Beispiele**

```
-- Geometry example - units in meters (SRID: 2163 US National Atlas Equal area) (3D point ←
  and line compared 2D point and line)
-- Note: currently no vertical datum support so Z is not transformed and assumed to be same ←
  units as final.
SELECT ST_3DDistance(
    ST_Transform('SRID=4326;POINT(-72.1235 42.3521 4) '::geometry,2163),
    ST_Transform('SRID=4326;LINESTRING(-72.1260 42.45 15, -72.123  ←
      42.1546 20) '::geometry,2163)
  ) As dist_3d,
  ST_Distance(
    ST_Transform('SRID=4326;POINT(-72.1235 42.3521) '::geometry,2163),
    ST_Transform('SRID=4326;LINESTRING(-72.1260 42.45, -72.123 42.1546) ←
      '::geometry,2163)
  ) As dist_2d;

dist_3d      |      dist_2d
-----+-----
127.295059324629 | 126.66425605671
```

```

-- Multilinestring and polygon both 3d and 2d distance
-- Same example as 3D closest point example
SELECT ST_3DDistance(poly, mline) As dist3d,
       ST_Distance(poly, mline) As dist2d
FROM (SELECT 'POLYGON((175 150 5, 20 40 5, 35 45 5, 50 60 5, 100 100 5, 175 150 5) ←
)>:::geometry as poly,
      'MULTILINESTRING((175 155 2, 20 40 20, 50 60 -2, 125 100 1, 175 155 1), (1 ←
10 2, 5 20 1))':::geometry as mline) as foo;
-----+-----
dist3d   | dist2d
-----+-----
0.716635696066337 | 0

```

## Siehe auch

[ST\\_Distance](#), [ST\\_3DClosestPoint](#), [ST\\_3DDWithin](#), [ST\\_3DMaxDistance](#), [ST\\_3DShortestLine](#), [ST\\_Transform](#)

## 7.12.8 ST\_DistanceSphere

**ST\_DistanceSphere** — Gibt die kürzeste Distanz zwischen zwei geometrischen Objekten zurück, die über Länge, Breite und ein bestimmtes Referenzellipsoid gegeben sind. Vorgängerversionen von PostGIS 1.5 unterstützten nur Punkte.

### Synopsis

```
float ST_DistanceSphere(geometry geom1lonlatA, geometry geom1lonlatB, float8 radius=6371008);
```

### Beschreibung

Gibt die kürzeste Distanz zwischen zwei Punkten zurück, die über Länge und Breite gegeben sind. Verwendet die Kugelform für die Erde und den Radius des Referenzellipsoids, der durch die SRID festgelegt ist. Ist schneller als [ST\\_DistanceSpheroid](#), aber weniger genau. Vorgängerversionen von PostGIS 1.5 unterstützten nur Punkte.

Verfügbarkeit: 1.5 die Unterstützung für weitere geometrische Datentypen neben Punkten eingeführt. Bei Vorgängerversionen wurden nur Punkte unterstützt.

Änderung: 2.2.0 In Vorgängerversionen als `ST_Distance_Sphere` bezeichnet.

### Beispiele

```

SELECT round(CAST(ST_DistanceSphere(ST_Centroid(geom), ST_GeomFromText('POINT(-118 38) ←
',4326)) As numeric),2) As dist_meters,
round(CAST(ST_Distance(ST_Transform(ST_Centroid(geom),32611),
ST_Transform(ST_GeomFromText('POINT(-118 38)', 4326),32611)) As numeric),2) ←
As dist_utm11_meters,
round(CAST(ST_Distance(ST_Centroid(geom), ST_GeomFromText('POINT(-118 38)', 4326)) As ←
numeric),5) As dist_degrees,
round(CAST(ST_Distance(ST_Transform(geom,32611),
ST_Transform(ST_GeomFromText('POINT(-118 38)', 4326),32611)) As numeric),2) ←
As min_dist_line_point_meters
FROM
(SELECT ST_GeomFromText('LINESTRING(-118.584 38.374,-118.583 38.5)', 4326) As geom) ←
as foo;
-----+-----+-----+-----
dist_meters | dist_utm11_meters | dist_degrees | min_dist_line_point_meters
-----+-----+-----+-----
70424.47 | 70438.00 | 0.72900 | 65871.18

```

**Siehe auch**[ST\\_Distance](#), [ST\\_DistanceSpheroid](#)**7.12.9 ST\_DistanceSpheroid**

`ST_DistanceSpheroid` — Gibt die kürzeste Distanz zwischen zwei geometrischen Objekten zurück, die über Länge, Breite und ein bestimmtes Referenzellipsoid gegeben sind. Vorgängerversionen von PostGIS 1.5 unterstützten nur Punkte.

**Synopsis**

```
float ST_DistanceSpheroid(geometry geomlonlatA, geometry geomlonlatB, spheroid measurement_spheroid=WGS84);
```

**Beschreibung**

Gibt die kürzeste Distanz zwischen zwei geometrischen Objekten in Meter zurück, die über Länge, Breite und ein bestimmtes Referenzellipsoid gegeben sind. Siehe die Erklärung zu Referenzellipsoiden unter [ST\\_LengthSpheroid](#). Vorgängerversionen von PostGIS 1.5 unterstützten nur Punkte.

**Note**

Aktuell schaut diese Funktion nicht auf die SRID der Geometrie, sondern nimmt an, dass die Geometrie in den Koordinaten des gegebenen Referenzellipsoids vorliegt. Vorgängerversionen von PostGIS 1.5 unterstützten nur Punkte.

Verfügbarkeit: 1.5 die Unterstützung für weitere geometrische Datentypen neben Punkten eingeführt. Bei Vorgängerversionen wurden nur Punkte unterstützt.

Änderung: 2.2.0 In Vorgängerversionen als `ST_Distance_Spheroid` bezeichnet.

**Beispiele**

```
SELECT round(CAST (
    ST_DistanceSpheroid(ST_Centroid(geom), ST_GeomFromText('POINT(-118 38) ←
    ',4326), 'SPHEROID["WGS 84",6378137,298.257223563]')
    As numeric),2) As dist_meters_spheroid,
    round(CAST(ST_DistanceSphere(ST_Centroid(geom), ST_GeomFromText('POINT(-118 ←
    38)',4326)) As numeric),2) As dist_meters_sphere,
    round(CAST(ST_Distance(ST_Transform(ST_Centroid(geom),32611),
    ST_Transform(ST_GeomFromText('POINT(-118 38)',4326),32611)) As numeric),2) ←
    As dist_utm11_meters
FROM
    (SELECT ST_GeomFromText('LINESTRING(-118.584 38.374,-118.583 38.5)',4326) As geom) ←
    as foo;
dist_meters_spheroid | dist_meters_sphere | dist_utm11_meters
-----+-----+-----
70454.92 | 70424.47 | 70438.00
```

**Siehe auch**[ST\\_Distance](#), [ST\\_DistanceSphere](#)

### 7.12.10 ST\_FrechetDistance

ST\_FrechetDistance — Gibt den kürzesten 3-dimensionalen Abstand zwischen zwei geometrischen Objekten als Linie zurück

#### Synopsis

```
float ST_FrechetDistance(geometry g1, geometry g2, float densifyFrac = -1);
```

#### Beschreibung

Implementiert einen Algorithmus zur Berechnung der Fréchet-Metrik, der für beide geometrischen Objekte auf diskrete Punkte beschränkt ist und auf [Computing Discrete Fréchet Distance](#) beruht. Die Fréchet-Metrik ist ein Maß für die Ähnlichkeit von Kurven, welches die Position und die Reihenfolge der Kurvenstützpunkte mit einbezieht. Daher ist sie oft besser geeignet als die Hausdorff-Metrik.

Wenn der optionale Parameter "densifyFrac" vorgegeben wird, dann führt diese Funktion eine Verdichtung der Linienstücke durch, bevor die diskrete Fréchet-Metrik berechnet wird. Der Parameter "densifyFrac" bestimmt um welchen Anteil die Linienstücke verdichtet werden. Jedes Linienstück wird in gleichlange Teilstücke zerlegt, deren Anteil an der Gesamtlänge am nächsten an den vorgegebenen Anteil herankommt.

Die Einheiten sind in den Einheiten des räumlichen Bezugssystems der Geometrien angegeben.



#### Note

Bei der aktuellen Implementierung können die diskreten Punkte nur Knoten sein. Dies könnte erweitert werden, um eine beliebige Punktdichte zu ermöglichen.



#### Note

Umso kleiner wir densifyFrac wählen, umso genauer wird die Fréchet-Metrik. Aber die Rechenzeit und der Speicherplatzbedarf steigen quadratisch mit der Anzahl der Teilabschnitte.

Wird vom GEOS Modul ausgeführt

Verfügbarkeit: 2.4.0 - benötigt GEOS >= 3.7.0

#### Beispiele

```
postgres=# SELECT st_frechetdistance('LINESTRING (0 0, 100 0)::geometry, 'LINESTRING (0 0, ↵
    50 50, 100 0)::geometry');
 st_frechetdistance
-----
    70.7106781186548
(1 row)
```

```
SELECT st_frechetdistance('LINESTRING (0 0, 100 0)::geometry, 'LINESTRING (0 0, 50 50, 100 ↵
    0)::geometry, 0.5);
 st_frechetdistance
-----
                    50
(1 row)
```

#### Siehe auch

[ST\\_HausdorffDistance](#)



### 7.12.11 ST\_HausdorffDistance

ST\_HausdorffDistance — Gibt den kürzesten 3-dimensionalen Abstand zwischen zwei geometrischen Objekten als Linie zurück

#### Synopsis

```
float ST_HausdorffDistance(geometry g1, geometry g2);
float ST_HausdorffDistance(geometry g1, geometry g2, float densifyFrac);
```

#### Beschreibung

Liefert den **Hausdorff-Abstand** zwischen zwei Geometrien. Der Hausdorff-Abstand ist ein Maß dafür, wie ähnlich oder unähnlich 2 Geometrien sind.

Die Funktion berechnet die "Diskrete Hausdorff-Distanz". Dies ist der Hausdorff-Abstand, der an diskreten Punkten der Geometrien berechnet wird. Der Parameter *densifyFrac* kann angegeben werden, um durch Verdichtung der Segmente vor der Berechnung des diskreten Hausdorff-Abstands eine genauere Antwort zu erhalten. Jedes Segment wird in eine Anzahl von Untersegmenten gleicher Länge aufgeteilt, deren Anteil an der Segmentlänge dem angegebenen Anteil am nächsten kommt.

Die Einheiten sind in den Einheiten des räumlichen Bezugssystems der Geometrien angegeben.



#### Note

Dieser Algorithmus ist NICHT gleichwertig mit dem Standard-Hausdorff-Abstand. Er berechnet jedoch eine Annäherung, die für eine große Teilmenge nützlicher Fälle korrekt ist. Ein wichtiger Fall sind Linien, die ungefähr parallel zueinander verlaufen und ungefähr gleich lang sind. Dies ist eine nützliche Metrik für die Anpassung von Linien.

Verfügbarkeit: 1.5.0

#### Beispiele



*Hausdorff-Abstand (rot) und Abstand (gelb) zwischen zwei Linien*

```
SELECT ST_HausdorffDistance(geomA, geomB),
       ST_Distance(geomA, geomB)
FROM (SELECT 'LINESTRING (20 70, 70 60, 110 70, 170 70)::geometry AS geomA,
            'LINESTRING (20 90, 130 90, 60 100, 190 100)::geometry AS geomB) AS t;
st_hausdorffdistance | st_distance
-----+-----
37.26206567625497 |          20
```

**Beispiel:** Hausdorff-Abstand mit Verdichtung.

```
SELECT ST_HausdorffDistance(
    'LINESTRING (130 0, 0 0, 0 150)::geometry,
    'LINESTRING (10 10, 10 150, 130 10)::geometry,
    0.5);
-----
70
```

**Beispiel:** Finden Sie für jedes Gebäude die Parzelle, die es am besten repräsentiert. Zunächst muss sich die Parzelle mit der Geometrie des Gebäudes schneiden. `DISTINCT ON` garantiert, dass jedes Gebäude nur einmal aufgeführt wird. `ORDER BY .. ST_HausdorffDistance` wählt die Parzelle aus, die dem Gebäude am ähnlichsten ist.

```
SELECT DISTINCT ON (buildings.gid) buildings.gid, parcels.parcel_id
FROM buildings
INNER JOIN parcels
ON ST_Intersects(buildings.geom, parcels.geom)
ORDER BY buildings.gid, ST_HausdorffDistance(buildings.geom, parcels.geom);
```

**Siehe auch**

[ST\\_FrechetDistance](#)

**7.12.12 ST\_Length**

`ST_Length` — Gibt den geometrischen Schwerpunkt einer Geometrie zurück.

**Synopsis**

```
float ST_Length(geometry a_2dlinestring);
float ST_Length(geography geog, boolean use_spheroid = true);
```

**Beschreibung**

Beim geometrischen Datentyp wird die kartesische 2D Länge der Geometrie zurückgegeben. Dabei muss es sich um einen LineString, einen MultiLineString, eine `ST_Curve` oder eine `ST_MultiCurve` handeln. Bei einer flächigen Geometrie wird 0 zurückgegeben. Für eine flächige Geometrie können Sie [ST\\_Perimeter](#) verwenden. Bei den geometrischen Datentypen sind die Einheiten für die Längenmessungen durch das Koordinatenreferenzsystem festgelegt.

Für Geografiertypen: Die Berechnung erfolgt über die inverse geodätische Berechnung. Die Längeneinheiten sind in Meter. Wenn PostGIS mit PROJ Version 4.8.0 oder später kompiliert wurde, wird das Sphäroid durch den SRID angegeben, andernfalls ausschließlich WGS84. Wenn `use_spheroid = false`, dann basiert die Berechnung auf einer Kugel anstelle eines Sphäroids.

Zur Zeit ein Synonym für `ST_Length2D`; dies kann sich allerdings ändern, wenn höhere Dimensionen unterstützt werden.

**Warning**

Änderung: 2.0.0 Wesentliche Änderung -- In früheren Versionen ergab die Anwendung auf ein MULTI/POLYGON vom geographischen Datentyp den Umfang des POLYGON/MULTIPOLYGON. In 2.0.0 wurde dies geändert und es wird jetzt 0 zurückgegeben, damit es mit der Verhaltensweise beim geometrischen Datentyp übereinstimmt. Verwenden Sie bitte `ST_Perimeter`, wenn Sie den Umfang eines Polygons wissen wollen

**Note**

Beim geographischer Datentyp werden die Messungen standardmäßig am Referenzellipsoid durchgeführt. Für die schnellere, aber ungenauere Berechnung auf einer Kugel können Sie `ST_Length(gg,false)` verwenden;



Diese Methode implementiert die [OGC Simple Features Implementation Specification for SQL 1.1](#). s2.1.5.1



Diese Methode setzt die SQL/MM-Spezifikation um. SQL-MM 3: 7.1.2, 9.3.4

Verfügbarkeit: 1.5.0 Unterstützung von geographischen Koordinaten.

**Beispiele mit dem geometrischen Datentyp**

Gibt die Länge eines Liniensegments zurück. Beachten Sie, dass die Einheit Fuß ist, da EPSG:2249 "Massachusetts State Plane Feet" ist

```
SELECT ST_Length(ST_GeomFromText('LINESTRING(743238 2967416,743238 2967450,743265 2967450,743265.625 2967416,743238 2967416)',2249));
```

```
st_length
```

```
-----
122.630744000095
```

```
--Transforming WGS 84 LineString to Massachusetts state plane meters
```

```
SELECT ST_Length(
  ST_Transform(
    ST_GeomFromEWKT('SRID=4326;LINESTRING(-72.1260 42.45, -72.1240 42.45666, ↔
      -72.123 42.1546)'),
    26986
  )
);
```

```
st_length
```

```
-----
34309.4563576191
```

**Beispiele für den geographischen Datentyp**

Gibt die Länge einer Linie zurück, die in geographischen WGS 84 Koordinaten vorliegt.

```
-- the default calculation uses a spheroid
SELECT ST_Length(the_geog) As length_spheroid, ST_Length(the_geog,false) As length_sphere
FROM (SELECT ST_GeographyFromText(
'SRID=4326;LINESTRING(-72.1260 42.45, -72.1240 42.45666, -72.123 42.1546)') As the_geog)
As foo;
```

```
length_spheroid | length_sphere
-----+-----
34310.5703627288 | 34346.2060960742
```

**Siehe auch**

[ST\\_GeographyFromText](#), [ST\\_GeomFromEWKT](#), [ST\\_LengthSpheroid](#), [ST\\_Perimeter](#), [ST\\_Transform](#)

### 7.12.13 ST\_Length2D

ST\_Length2D — Gibt die 2-dimensionale Länge einer Linie oder einer Mehrfachlinie zurück. Dies ist ein Alias für ST\_Length

#### Synopsis

```
float ST_Length2D(geometry a_2dlinestring);
```

#### Beschreibung

Gibt die 2-dimensionale Länge einer Linie oder einer Mehrfachlinie zurück. Dies ist ein Alias für ST\_Length

#### Siehe auch

[ST\\_Length](#), [ST\\_3DLength](#)

### 7.12.14 ST\_3DLength

ST\_3DLength — Gibt den geometrischen Schwerpunkt einer Geometrie zurück.

#### Synopsis

```
float ST_3DLength(geometry a_3dlinestring);
```

#### Beschreibung

Gibt die 2- oder 3-dimensionale Länge einer Linie oder einer Mehrfachlinie zurück. Bei einer 2-D Linie wird die Länge nur in 2D zurückgegeben (gleich wie ST\_Length und ST\_Length2D)



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



Diese Methode setzt die SQL/MM-Spezifikation um. SQL-MM IEC 13249-3: 7.1, 10.3

Änderung: 2.0.0 In Vorgängerversionen als ST\_Length3D bezeichnet.

#### Beispiele

Gibt die Länge eines 3D-Kabels zurück. Beachten Sie, dass die Einheit Fuß ist, da EPSG:2249 "Massachusetts State Plane Feet" ist

```
SELECT ST_3DLength(ST_GeomFromText('LINESTRING(743238 2967416 1,743238 2967450 1,743265 2967450 3,743265.625 2967416 3,743238 2967416 3)',2249));
ST_3DLength
-----
122.704716741457
```

#### Siehe auch

[ST\\_Length](#), [ST\\_Length2D](#)

## 7.12.15 ST\_LengthSpheroid

ST\_LengthSpheroid — Gibt den geometrischen Schwerpunkt einer Geometrie zurück.

### Synopsis

```
float ST_LengthSpheroid(geometry a_geometry, spheroid a_spheroid);
```

### Beschreibung

Berechnet die/den Länge/Umfang einer Geometrie auf einem Ellipsoid. Dies ist nützlich wenn die Koordinaten der Geometrie in Länge und Breite vorliegen, und die Länge der Geometrie ohne benötigt wird, ohne dass umprojiziert werden muss. Das Ellipsoid ist ein eigener Datentyp und kann wie folgt erstellt werden:

```
SPHEROID [<NAME
>, <SEMI-MAJOR AXIS
>, <INVERSE FLATTENING
>]
```

### Geometrie Beispiel

```
SPHEROID["GRS_1980", 6378137, 298.257222101]
```

Verfügbarkeit: 1.2.2

Änderung: 2.2.0 In Vorgängerversionen als ST\_Length\_Spheroid bezeichnet und mit dem Alias "ST\_3DLength\_Spheroid" versehen



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

### Beispiele

```
SELECT ST_LengthSpheroid( geometry_column,
                          'SPHEROID["GRS_1980", 6378137, 298.257222101]' )
FROM geometry_table;

SELECT ST_LengthSpheroid( geom, sph_m ) As tot_len,
ST_LengthSpheroid(ST_GeometryN(geom,1), sph_m) As len_line1,
ST_LengthSpheroid(ST_GeometryN(geom,2), sph_m) As len_line2
FROM (SELECT ST_GeomFromText ('MULTILINESTRING((-118.584  ←
                          38.374, -118.583 38.5),
                          (-71.05957 42.3589 , -71.061 43))') As geom,
CAST('SPHEROID["GRS_1980", 6378137, 298.257222101]' As spheroid) As sph_m) as foo;
  tot_len      |   len_line1   |   len_line2
-----+-----+-----
  85204.5207562955 | 13986.8725229309 | 71217.6482333646

--3D
SELECT ST_LengthSpheroid( geom, sph_m ) As tot_len,
ST_LengthSpheroid(ST_GeometryN(geom,1), sph_m) As len_line1,
ST_LengthSpheroid(ST_GeometryN(geom,2), sph_m) As len_line2
FROM (SELECT ST_GeomFromEWKT ('MULTILINESTRING((-118.584 38.374  ←
                          20, -118.583 38.5 30),
                          (-71.05957 42.3589 75, -71.061 43 90))') As geom,
CAST('SPHEROID["GRS_1980", 6378137, 298.257222101]' As spheroid) As sph_m) as foo;
  tot_len      |   len_line1   |   len_line2
-----+-----+-----
```

85204.5259107402 | 13986.876097711 | 71217.6498130292

## Siehe auch

[ST\\_GeometryN](#), [ST\\_Length](#)

## 7.12.16 ST\_LongestLine

`ST_LongestLine` — Gibt die größte 3-dimensionale Distanz zwischen zwei geometrischen Objekten als Linie zurück

### Synopsis

```
geometry ST_LongestLine(geometry g1, geometry g2);
```

### Beschreibung

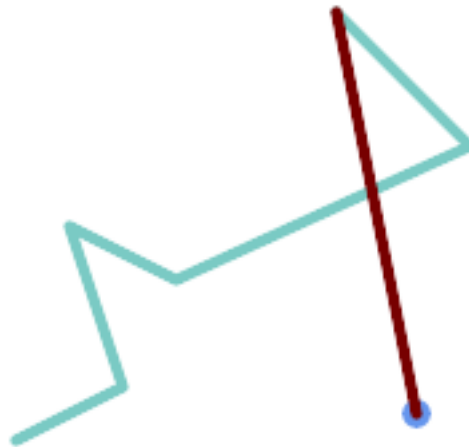
Liefert die 2-dimensionale längste Linie zwischen den Punkten zweier Geometrien. Die zurückgegebene Linie beginnt auf `g1` und endet auf `g2`.

Die längste Linie liegt immer zwischen zwei Scheitelpunkten. Die Funktion gibt die erste längste Linie zurück, wenn mehr als eine gefunden wird. Die Länge der Linie ist gleich dem von [ST\\_MaxDistance](#) zurückgegebenen Abstand.

Wenn `g1` und `g2` dieselbe Geometrie sind, wird die Linie zwischen den beiden am weitesten voneinander entfernten Scheitelpunkten der Geometrie zurückgegeben. Die Endpunkte der Linie liegen auf dem von [ST\\_MinimumBoundingCircle](#) berechneten Kreis.

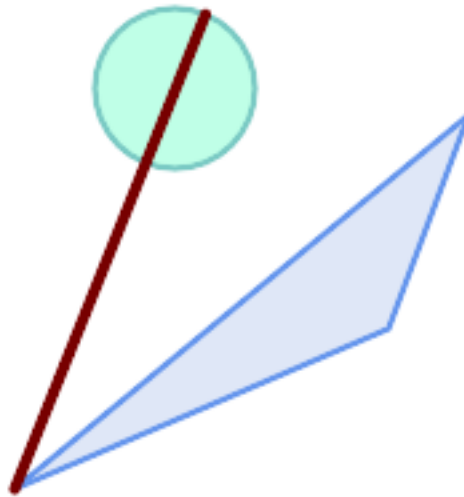
Verfügbarkeit: 1.5.0

### Beispiele



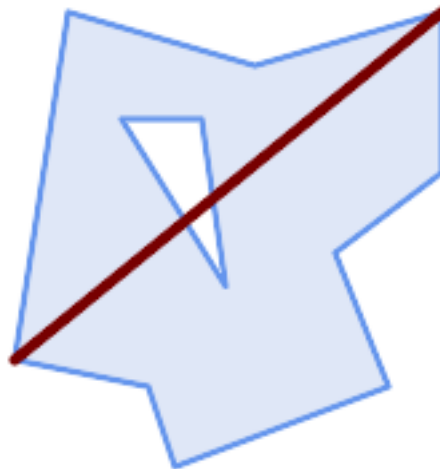
*Längste Strecke zwischen Punkt und Linie*

```
SELECT ST_AsText( ST_LongestLine(
    'POINT (160 40)',
    'LINESTRING (10 30, 50 50, 30 110, 70 90, 180 140, 130 190)' )
) AS lline;
-----
LINESTRING(160 40,130 190)
```



*Längste Strecke zwischen Polygon und Polygon*

```
SELECT ST_AsText( ST_LongestLine(
    'POLYGON ((190 150, 20 10, 160 70, 190 150))',
    ST_Buffer('POINT(80 160)', 30)
) ) AS llinewkt;
-----
LINESTRING(20 10,105.3073372946034 186.95518130045156)
```



*Längste Linie durch eine einzelne Geometrie. Die Länge der Linie ist gleich dem Maximalen Abstand. Die Endpunkte der Linie liegen auf dem Minimalen Begrenzungskreis.*

```
SELECT ST_AsText( ST_LongestLine( geom, geom)) AS llinewkt,
    ST_MaxDistance( geom, geom) AS max_dist,
    ST_Length( ST_LongestLine(geom, geom)) AS lenl1
FROM (SELECT 'POLYGON ((40 180, 110 160, 180 180, 180 120, 140 90, 160 40, 80 10, 70 40, 20 ←
    50, 40 180),
    (60 140, 99 77.5, 90 140, 60 140))'::geometry AS geom) AS t;

-----
llinewkt          |          max_dist          |          lenl1
-----+-----+-----
LINESTRING(20 50,180 180) | 206.15528128088303 | 206.15528128088303
```

**Siehe auch**

[ST\\_MaxDistance](#), [ST\\_ShortestLine](#), [ST\\_3DLongestLine](#), [ST\\_MinimumBoundingCircle](#)

**7.12.17 ST\_3DLongestLine**

`ST_3DLongestLine` — Gibt die größte 3-dimensionale Distanz zwischen zwei geometrischen Objekten als Linie zurück

**Synopsis**

```
geometry ST_3DLongestLine(geometry g1, geometry g2);
```

**Beschreibung**

Gibt den größten 3-dimensionalen Abstand zwischen zwei geometrischen Objekten als Linie zurück. Wenn es mehr als einen größten Abstand gibt, dann wird nur die erste zurückgegeben. Die zurückgegebene Linie fängt immer mit "g1" an und endet mit "g2". Die Länge der 3D-Linie die von dieser Funktion zurückgegeben wird ist immer ident mit der von [ST\\_3DMaxDistance](#) für "g1" und "g2" zurückgegebenen Distanz.

Verfügbarkeit: 2.0.0

Änderung: 2.2.0 - Wenn 2 geometrische Objekte in 2D übergeben werden, wird ein 2D-Punkt zurückgegeben (anstelle wie früher 0 für ein fehlendes Z). Im Falle von 2D und 3D, wird für fehlende Z nicht länger 0 angenommen.



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



Diese Funktion unterstützt polyedrische Flächen.

**Beispiele****Linienstück und Punkt -- größter Abstand in 3D und in 2D**

```
SELECT ST_AsEWKT(ST_3DLongestLine(line,pt)) AS lol3d_line_pt,
       ST_AsEWKT(ST_LongestLine(line,pt)) As lol2d_line_pt
FROM (SELECT 'POINT(100 100 30)::geometry As pt,
            'LINESTRING (20 80 20, 98 190 1, 110 180 3, 50 75 1000)>::: ←
       geometry As line
       ) As foo;
```

```
lol3d_line_pt      | lol2d_line_pt
-----+-----
LINESTRING(50 75 1000,100 100 30) | LINESTRING(98 190,100 100)
```

**Linienstück und Mehrfachpunkt -- größter Abstand in 3D und in 2D**

```
SELECT ST_AsEWKT(ST_3DLongestLine(line,pt)) AS lol3d_line_pt,
       ST_AsEWKT(ST_LongestLine(line,pt)) As lol2d_line_pt
FROM (SELECT 'MULTIPOINT(100 100 30, 50 74 1000)>:::geometry As pt,
            'LINESTRING (20 80 20, 98 190 1, 110 180 3, 50 75 900)>::: ←
       geometry As line
       ) As foo;
```

```
lol3d_line_pt      | lol2d_line_pt
-----+-----
LINESTRING(98 190 1,50 74 1000) | LINESTRING(98 190,50 74)
```



**Mehrfachlinie und Polygon - größter Abstand in 3D und in 2D**

```

SELECT ST_AsEWKT(ST_3DLongestLine(poly, mline)) As lol3d,
       ST_AsEWKT(ST_LongestLine(poly, mline)) As lol2d
  FROM (SELECT ST_GeomFromEWKT('POLYGON((175 150 5, 20 40 5, 35 45 5, 50 60 5, ←
100 100 5, 175 150 5))') As poly,
       ST_GeomFromEWKT('MULTILINESTRING((175 155 2, 20 40 20, 50 60 -2, 125 ←
100 1, 175 155 1),
       (1 10 2, 5 20 1))') As mline ) As foo;
-----+-----
lol3d          |          lol2d
-----+-----
LINESTRING(175 150 5,1 10 2) | LINESTRING(175 150,1 10)

```

**Siehe auch**

[ST\\_3DClosestPoint](#), [ST\\_3DDistance](#), [ST\\_LongestLine](#), [ST\\_3DShortestLine](#), [ST\\_3DMaxDistance](#)

**7.12.18 ST\_MaxDistance**

**ST\_MaxDistance** — Gibt die größte 2-dimensionale Distanz zwischen zwei geometrischen Objekten in projizierten Einheiten zurück.

**Synopsis**

```
float ST_MaxDistance(geometry g1, geometry g2);
```

**Beschreibung**

Gibt die größte 2-dimensionale Distanz zwischen zwei geometrischen Objekten in projizierten Einheiten zurück. Wenn g1 und g2 dieselbe Geometrie sind, dann gibt die Funktion die Distanz zwischen den beiden am weitesten entfernten Knoten in dieser Geometrie zurück.

Gibt die größte 2-dimensionale Distanz zwischen zwei geometrischen Objekten in projizierten Einheiten zurück. Wenn g1 und g2 dieselbe Geometrie sind, dann gibt die Funktion die Distanz zwischen den beiden am weitesten entfernten Knoten in dieser Geometrie zurück.

Verfügbarkeit: 1.5.0

**Beispiele****Längste Strecke zwischen Punkt und Linie**

```
SELECT ST_MaxDistance('POINT(0 0)::geometry, 'LINESTRING ( 2 0, 0 2 ) '::geometry);
```

```
-----
2
```

```
SELECT ST_MaxDistance('POINT(0 0)::geometry, 'LINESTRING ( 2 2, 2 2 ) '::geometry);
```

```
-----
2.82842712474619
```

**Maximaler Abstand zwischen Scheitelpunkten einer einzelnen Geometrie.**

```
SELECT ST_MaxDistance('POLYGON ((10 10, 10 0, 0 0, 10 10)) '::geometry,
                       'POLYGON ((10 10, 10 0, 0 0, 10 10)) '::geometry);
```

```
-----
14.142135623730951
```

**Siehe auch**

[ST\\_Distance](#), [ST\\_LongestLine](#), [ST\\_DFullyWithin](#)

**7.12.19 ST\_3DMaxDistance**

**ST\_3DMaxDistance** — Für den geometrischen Datentyp. Gibt die maximale 3-dimensionale kartesische Distanz (basierend auf dem Koordinatenreferenzsystem) zwischen zwei geometrischen Objekten in projizierten Einheiten zurück.

**Synopsis**

```
float ST_3DMaxDistance(geometry g1, geometry g2);
```

**Beschreibung**

Für den geometrischen Datentyp. Gibt die maximale 3-dimensionale kartesische Distanz zwischen zwei geometrischen Objekten in projizierten Einheiten (Einheiten des Koordinatenreferenzsystem) zurück.



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



Diese Funktion unterstützt polyedrische Flächen.

Verfügbarkeit: 2.0.0

Änderung: 2.2.0 - Im Falle von 2D und 3D wird für ein fehlendes Z nicht mehr 0 angenommen.

**Beispiele**

```
-- Geometry example - units in meters (SRID: 2163 US National Atlas Equal area) (3D point ↔
  and line compared 2D point and line)
-- Note: currently no vertical datum support so Z is not transformed and assumed to be same ↔
  units as final.
SELECT ST_3DMaxDistance(
    ST_Transform(ST_GeomFromEWKT('SRID=4326;POINT(-72.1235 42.3521 10000)'),2163),
    ST_Transform(ST_GeomFromEWKT('SRID=4326;LINESTRING(-72.1260 42.45 15, -72.123 42.1546 20)'),2163)
) As dist_3d,
ST_MaxDistance(
    ST_Transform(ST_GeomFromEWKT('SRID=4326;POINT(-72.1235 42.3521 10000)'),2163),
    ST_Transform(ST_GeomFromEWKT('SRID=4326;LINESTRING(-72.1260 42.45 15, -72.123 42.1546 20)'),2163)
) As dist_2d;

dist_3d      |      dist_2d
-----+-----
24383.7467488441 | 22247.8472107251
```

**Siehe auch**

[ST\\_Distance](#), [ST\\_3DDWithin](#), [ST\\_3DMaxDistance](#), [ST\\_Transform](#)

### 7.12.20 ST\_MinimumClearance

`ST_MinimumClearance` — Gibt das Mindestabstandsmaß für eine Geometrie zurück; ein Maß für die Robustheit einer Geometrie.

#### Synopsis

```
float ST_MinimumClearance(geometry g);
```

#### Beschreibung

Es ist nicht ungewöhnlich dass eine Geometrie, welche die Kriterien für Validität gemäß `ST_IsValid` (Polygone) oder `ST_IsSimple` (Linien) erfüllt, durch eine geringe Verschiebung von einem Knoten invalid wird. Dies kann während einer Konvertierung in Textformate (wie WKT, KML, GML und GeoJSON) vorkommen, oder bei binären Formaten, welche die Koordinaten nicht als Gleitpunkt-Zahl mit doppelter Genauigkeit (double-precision floating point) abspeichern (MapInfo TAB).

Der Mindestabstand ist ein quantitatives Maß für die Robustheit einer Geometrie gegenüber Änderungen der Koordinatengenauigkeit. Es ist der größte Abstand, um den die Eckpunkte der Geometrie verschoben werden können, ohne dass eine ungültige Geometrie entsteht. Größere Werte des Mindestabstands bedeuten eine größere Robustheit.

Wenn eine Geometrie ein Mindestabstandsmaß von  $\epsilon$  hat, dann gilt:

- Zwei sich unterscheidende Knoten der Geometrie sind nicht weniger als  $\epsilon$  voneinander entfernt.
- Kein Knoten liegt näher als  $\epsilon$  bei einem Liniensegment, außer es ist ein Endpunkt.

Wenn für eine Geometrie kein Mindestabstandsmaß existiert (zum Beispiel ein einzelner Punkt, oder ein Mehrfachpunkt, dessen Punkte identisch sind), dann gibt `ST_MinimumClearance` unendlich zurück.

Um durch Präzisionsverluste verursachte Gültigkeitsprobleme zu vermeiden, kann `ST_ReducePrecision` die Koordinatengenauigkeit verringern und gleichzeitig sicherstellen, dass die polygonale Geometrie gültig bleibt.

Verfügbarkeit: 2.3.0

#### Beispiele

```
SELECT ST_MinimumClearance('POLYGON ((0 0, 1 0, 1 1, 0.5 3.2e-4, 0 0))');
st_minimumclearance
-----
0.00032
```

#### Siehe auch

[ST\\_MinimumClearanceLine](#), [ST\\_Crosses](#), [ST\\_Dimension](#), [ST\\_Intersects](#)

### 7.12.21 ST\_MinimumClearanceLine

`ST_MinimumClearanceLine` — Gibt ein Liniensegment mit zwei Punkten zurück, welche sich über das Mindestabstandsmaß erstreckt.

#### Synopsis

```
Geometry ST_MinimumClearanceLine(geometry g);
```

## Beschreibung

Gibt den Zweipunkt-LineString zurück, der den Mindestabstand einer Geometrie umspannt. Wenn die Geometrie keinen Mindestabstand hat, wird `LINestring EMPTY` zurückgegeben.

Wird vom GEOS Modul ausgeführt

Verfügbarkeit: 2.3.0 - benötigt GEOS >= 3.6.0

## Beispiele

```
SELECT ST_AsText(ST_MinimumClearanceLine('POLYGON ((0 0, 1 0, 1 1, 0.5 3.2e-4, 0 0))'));
-----
LINestring(0.5 0.00032,0.5 0)
```

## Siehe auch

[ST\\_MinimumClearance](#)

## 7.12.22 ST\_Perimeter

`ST_Perimeter` — Gibt die Länge der Begrenzung einer polygonalen Geometrie oder Geografie zurück.

## Synopsis

```
float ST_Perimeter(geometry g1);
float ST_Perimeter(geography geog, boolean use_spheroid = true);
```

## Beschreibung

Gibt für die geometrischen/geographischen Datentypen `ST_Surface`, `ST_MultiSurface` (Polygon, MultiPolygon) den Umfang in 2D zurück. Bei einer nicht flächigen Geometrie wird 0 zurückgegeben. Für eine lineare Geometrie können Sie [ST\\_Length](#) verwenden. Beim geometrischen Datentyp sind die Einheiten der Umfangsmessung durch das Koordinatenreferenzsystem der Geometrie festgelegt.

Bei geographischen Typen werden die Berechnungen mit dem inversen geodätischen Problem durchgeführt, wobei die Perimeterereinheiten in Metern angegeben werden. Wenn PostGIS mit PROJ Version 4.8.0 oder später kompiliert wurde, wird das Sphäroid durch den SRID spezifiziert, andernfalls ist es ausschließlich WGS84. Wenn `use_spheroid = false`, dann werden die Berechnungen eine Kugel anstelle eines Sphäroids approximieren.

Zur Zeit ein Synonym für `ST_Perimeter2D`; dies kann sich allerdings ändern, wenn höhere Dimensionen unterstützt werden.



Diese Methode implementiert die [OGC Simple Features Implementation Specification for SQL 1.1](#). s2.1.5.1



Diese Methode setzt die SQL/MM-Spezifikation um. SQL-MM 3: 8.1.3, 9.5.4

Verfügbarkeit: Mit 2.0.0 wurde die Unterstützung für geograpischen Koordinaten eingeführt

## Beispiele: Geometrie

Den Umfang eines Polygons und eines Mehrfachpolygons in Fuß ausgeben. Beachten Sie bitte, dass die Einheit Fuß ist, da EPSG:2249 "Massachusetts State Plane Feet" ist

```
SELECT ST_Perimeter(ST_GeomFromText('POLYGON((743238 2967416,743238 2967450,743265 2967450,
743265.625 2967416,743238 2967416))', 2249));
st_perimeter
-----
122.630744000095
(1 row)

SELECT ST_Perimeter(ST_GeomFromText('MULTIPOLYGON(((763104.471273676 2949418.44119003,
763104.477769673 2949418.42538203,
763104.189609677 2949418.22343004,763104.471273676 2949418.44119003)),
((763104.471273676 2949418.44119003,763095.804579742 2949436.33850239,
763086.132105649 2949451.46730207,763078.452329651 2949462.11549407,
763075.354136904 2949466.17407812,763064.362142565 2949477.64291974,
763059.953961626 2949481.28983009,762994.637609571 2949532.04103014,
762990.568508415 2949535.06640477,762986.710889563 2949539.61421415,
763117.237897679 2949709.50493431,763235.236617789 2949617.95619822,
763287.718121842 2949562.20592617,763111.553321674 2949423.91664605,
763104.471273676 2949418.44119003)))', 2249));
st_perimeter
-----
845.227713366825
(1 row)
```

**Beispiele: Geographie**

Gibt den Umfang eines Polygons und eines Mehrfachpolygons in Meter und in Fuß aus. Beachten Sie, dass diese in geographischen Koordinaten (WGS 84 Länge/Breite) vorliegen.

```
SELECT ST_Perimeter(geog) As per_meters, ST_Perimeter(geog)/0.3048 As per_ft
FROM ST_GeogFromText('POLYGON((-71.1776848522251 42.3902896512902,-71.1776843766326 ↵
42.3903829478009,
-71.1775844305465 42.3903826677917,-71.1775825927231 42.3902893647987,-71.1776848522251 ↵
42.3902896512902)))' As geog;

per_meters | per_ft
-----+-----
37.3790462565251 | 122.634666195949

-- MultiPolygon example --
SELECT ST_Perimeter(geog) As per_meters, ST_Perimeter(geog,false) As per_sphere_meters, ↵
ST_Perimeter(geog)/0.3048 As per_ft
FROM ST_GeogFromText('MULTIPOLYGON((( -71.1044543107478 42.340674480411,-71.1044542869917 ↵
42.3406744369506,
-71.1044553562977 42.340673886454,-71.1044543107478 42.340674480411)),
((-71.1044543107478 42.340674480411,-71.1044860600303 42.3407237015564,-71.1045215770124 ↵
42.3407653385914,
-71.1045498002983 42.3407946553165,-71.1045611902745 42.3408058316308,-71.1046016507427 ↵
42.340837442371,
-71.104617893173 42.3408475056957,-71.1048586153981 42.3409875993595,-71.1048736143677 ↵
42.3409959528211,
-71.1048878050242 42.3410084812078,-71.1044020965803 42.3414730072048,
-71.1039672113619 42.3412202916693,-71.1037740497748 42.3410666421308,
-71.1044280218456 42.3406894151355,-71.1044543107478 42.340674480411)))' As geog;

per_meters | per_sphere_meters | per_ft
-----+-----+-----
257.634283683311 | 257.412311446337 | 845.256836231335
```

**Siehe auch**

[ST\\_GeogFromText](#), [ST\\_GeomFromText](#), [ST\\_Length](#)

**7.12.23 ST\_Perimeter2D**

ST\_Perimeter2D — Gibt den 2D-Umfang einer polygonalen Geometrie zurück. Alias für ST\_Perimeter.

**Synopsis**

```
float ST_Perimeter2D(geometry geomA);
```

**Beschreibung**

Gibt den 2-dimensionalen Umfang eines Polygons oder eines Mehrfachpolygons zurück.

**Note**

Zurzeit ein Alias für ST\_Perimeter. In zukünftigen Versionen dürfte ST\_Perimeter den Umfang in der höchsten Dimension einer Geometrie zurückgeben. Dies befindet sich jedoch noch im Aufbau.

**Siehe auch**

[ST\\_Perimeter](#)

**7.12.24 ST\_3DPerimeter**

ST\_3DPerimeter — Gibt den geometrischen Schwerpunkt einer Geometrie zurück.

**Synopsis**

```
float ST_3DPerimeter(geometry geomA);
```

**Beschreibung**

Gibt den 3-dimensionalen Umfang eines Polygons oder eines Mehrfachpolygons zurück. Wenn es sich um eine 2-dimensionale Geometrie handelt wird der 2-dimensionale Umfang zurückgegeben.



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



Diese Methode setzt die SQL/MM-Spezifikation um. SQL-MM ISO/IEC 13249-3: 8.1, 10.5

Änderung: 2.0.0 In Vorgängerversionen als ST\_Perimeter3D bezeichnet.

## Beispiele

Umfang eines leicht erhöhten Polygons in "Massachusetts state plane feet"

```
SELECT ST_3DPerimeter(geom), ST_Perimeter2d(geom), ST_Perimeter(geom) FROM
      (SELECT ST_GeomFromEWKT('SRID=2249;POLYGON((743238 2967416 2,743238 ←
                2967450 1,
743265.625 2967416 1,743238 2967416 2))') As geom) As foo;

  ST_3DPerimeter | st_perimeter2d | st_perimeter
-----+-----+-----
105.465793597674 | 105.432997272188 | 105.432997272188
```

## Siehe auch

[ST\\_GeomFromEWKT](#), [ST\\_Perimeter](#), [ST\\_Perimeter2D](#)

### 7.12.25 ST\_ShortestLine

`ST_ShortestLine` — Gibt die 2-dimensionale kürzeste Strecke zwischen zwei Geometrien als Linie zurück

#### Synopsis

```
geometry ST_ShortestLine(geometry geom1, geometry geom2);
geography ST_ShortestLine(geography geom1, geography geom2, boolean use_spheroid = true);
```

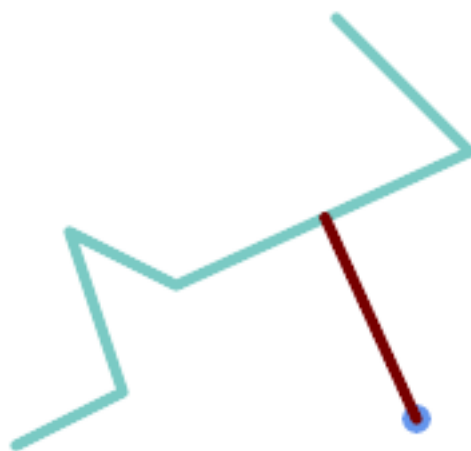
#### Beschreibung

Gibt die 2-dimensionale kürzeste Linie zwischen zwei Geometrien zurück. Die zurückgegebene Linie beginnt in `geom1` und endet in `geom2`. Wenn `geom1` und `geom2` sich schneiden, ist das Ergebnis eine Linie mit Start und Ende am Schnittpunkt. Die Länge der Linie ist dieselbe, die [ST\\_Distance](#) für `g1` und `g2` zurückgibt.

Verbessert: 3.4.0 - Unterstützung für Geographie.

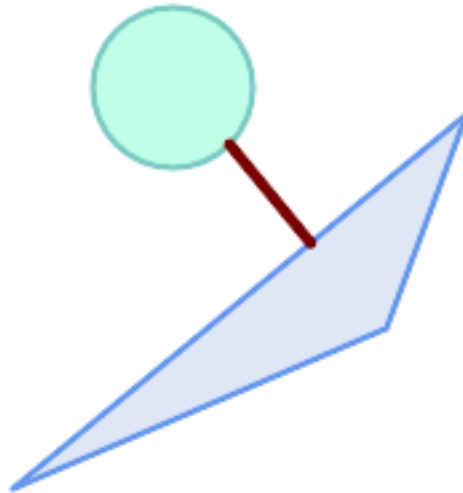
Verfügbarkeit: 1.5.0

#### Beispiele



*Kürzeste Linie zwischen Punkt und LineString*

```
SELECT ST_AsText( ST_ShortestLine(
  'POINT (160 40)',
  'LINESTRING (10 30, 50 50, 30 110, 70 90, 180 140, 130 190)')
) As sline;
-----
LINESTRING(160 40,125.75342465753425 115.34246575342466)
```



*Kürzeste Linie zwischen Polygonen*

```
SELECT ST_AsText( ST_ShortestLine(
  'POLYGON ((190 150, 20 10, 160 70, 190 150))',
  ST_Buffer('POINT(80 160)', 30)
) ) AS llinewkt;
-----
LINESTRING(131.59149149528952 101.89887534906197,101.21320343559644 138.78679656440357)
```

#### Siehe auch

[ST\\_ClosestPoint](#), [ST\\_Distance](#), [ST\\_LongestLine](#), [ST\\_MaxDistance](#)

### 7.12.26 ST\_3DShortestLine

**ST\_3DShortestLine** — Gibt den kürzesten 3-dimensionalen Abstand zwischen zwei geometrischen Objekten als Linie zurück

#### Synopsis

```
geometry ST_3DShortestLine(geometry g1, geometry g2);
```

#### Beschreibung

Gibt den kürzesten 3-dimensionalen Abstand zwischen zwei geometrischen Objekten als Linie zurück. Wenn es mehrere kürzeste Abstände gibt, dann wird nur der erste zurückgegeben, der von der Funktion gefunden wurde. Wenn sich g1 und g2 nur in einem Punkt schneiden, dann gibt die Funktion eine Linie zurück, die ihren Anfang und ihr Ende in dem Schnittpunkt hat. Wenn sich g1 und g2 in mehreren Punkten schneiden, dann gibt die Funktion eine Linie zurück, die Anfang und Ende in irgendeinem der



Schnittpunkte hat. Die zurückgegebene Linie beginnt immer mit g1 und endet mit g2. Die Länge der 3D-Linie die von dieser Funktion zurückgegeben wird ist immer ident mit der von **ST\_3DDistance** für g1 und g2 zurückgegebenen Distanz.

Verfügbarkeit: 2.0.0

Änderung: 2.2.0 - Wenn 2 geometrische Objekte in 2D übergeben werden, wird ein 2D-Punkt zurückgegeben (anstelle wie früher 0 für ein fehlendes Z). Im Falle von 2D und 3D, wird für fehlende Z nicht länger 0 angenommen.



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



Diese Funktion unterstützt polyedrische Flächen.

## Beispiele

### Linienzug und Punkt -- kürzester Abstand in 3D und in 2D

```
SELECT ST_AsEWKT(ST_3DShortestLine(line,pt)) AS shl3d_line_pt,
       ST_AsEWKT(ST_ShortestLine(line,pt)) As shl2d_line_pt
FROM (SELECT 'POINT(100 100 30)>:::geometry As pt,
            'LINESTRING (20 80 20, 98 190 1, 110 180 3, 50 75 1000)>::: ←
      geometry As line
      ) As foo;
```

```
shl3d_line_pt                                     | ←
-----
shl2d_line_pt
```

```
LINESTRING(54.6993798867619 128.935022917228 11.5475869506606,100 100 30) | ←
LINESTRING(73.0769230769231 115.384615384615,100 100)
```

### Linienstück und Mehrfachpunkt -- kürzester Abstand in 3D und in 2D

```
SELECT ST_AsEWKT(ST_3DShortestLine(line,pt)) AS shl3d_line_pt,
       ST_AsEWKT(ST_ShortestLine(line,pt)) As shl2d_line_pt
FROM (SELECT 'MULTIPOINT(100 100 30, 50 74 1000)>:::geometry As pt,
            'LINESTRING (20 80 20, 98 190 1, 110 180 3, 50 75 900)>::: ←
      geometry As line
      ) As foo;
```

```
shl2d_line_pt                                     shl3d_line_pt         | ←
-----
```

```
LINESTRING(54.6993798867619 128.935022917228 11.5475869506606,100 100 30) | LINESTRING ←
(50 75,50 74)
```

### Mehrfachlinienzug und Polygon - kürzester Abstand in 3D und in 2D

```
SELECT ST_AsEWKT(ST_3DShortestLine(poly, mline)) As shl3d,
       ST_AsEWKT(ST_ShortestLine(poly, mline)) As shl2d
FROM (SELECT ST_GeomFromEWKT('POLYGON((175 150 5, 20 40 5, 35 45 5, 50 60 5, ←
100 100 5, 175 150 5))') As poly,
       ST_GeomFromEWKT('MULTILINESTRING((175 155 2, 20 40 20, 50 60 -2, 125 ←
100 1, 175 155 1),
       (1 10 2, 5 20 1))') As mline ) As foo;
shl3d ←
```

```
| shl2d
```

```
LINESTRING(39.993580415989 54.1889925532825 5,40.4078575708294 53.6052383805529 ←
5.03423778139177) | LINESTRING(20 40,20 40)
```

**Siehe auch**

[ST\\_3DClosestPoint](#), [ST\\_3DDistance](#), [ST\\_LongestLine](#), [ST\\_ShortestLine](#), [ST\\_3DMaxDistance](#)

## 7.13 Overlay-Funktionen

### 7.13.1 ST\_ClipByBox2D

`ST_ClipByBox2D` — Berechnet den Teil einer Geometrie, der innerhalb eines Rechtecks liegt.

**Synopsis**

```
geometry ST_ClipByBox2D(geometry geom, box2d box);
```

**Beschreibung**

Beschneidet eine Geometrie durch eine 2D-Box auf eine schnelle und tolerante, aber möglicherweise ungültige Weise. Topologisch ungültige Eingabegeometrien führen nicht zum Auslösen von Exceptions. Es ist nicht garantiert, dass die Ausgabegeometrie gültig ist (insbesondere können Selbstüberschneidungen für ein Polygon eingeführt werden).

Wird vom GEOS Modul ausgeführt

Verfügbarkeit: 2.2.0

**Beispiele**

```
-- Rely on implicit cast from geometry to box2d for the second parameter
SELECT ST_ClipByBox2D(geom, ST_MakeEnvelope(0,0,10,10)) FROM mytab;
```

**Siehe auch**

[ST\\_Intersection](#), [ST\\_MakeBox2D](#), [ST\\_MakeEnvelope](#)

### 7.13.2 ST\_Difference

`ST_Difference` — Berechnet eine Geometrie, die den Teil der Geometrie A darstellt, der die Geometrie B nicht schneidet.

**Synopsis**

```
geometry ST_Difference(geometry geomA, geometry geomB, float8 gridSize = -1);
```

**Beschreibung**

Gibt eine Geometrie zurück, die den Teil der Geometrie A darstellt, der die Geometrie B nicht schneidet. Dies entspricht  $A - \text{ST\_Intersection}(A, B)$ . Wenn A vollständig in B enthalten ist, wird eine leere atomare Geometrie des entsprechenden Typs zurückgegeben.

**Note**

Dies ist die einzige Überlagerungsfunktion, bei der die Reihenfolge der Eingabe eine Rolle spielt. `ST_Difference(A, B)` gibt immer einen Teil von A zurück.

Wenn das optionale Argument `gridSize` angegeben wird, werden die Eingaben auf ein Gitter der angegebenen Größe gerastert, und die Ergebnispunkte werden auf demselben Gitter berechnet. (Benötigt GEOS-3.9.0 oder höher)

Wird durch das GEOS Modul ausgeführt

Verbessert: 3.1.0 akzeptiert einen `gridSize`-Parameter.

Erfordert GEOS >= 3.9.0 zur Verwendung des Parameters `gridSize`.

- ✔ Diese Methode implementiert die [OGC Simple Features Implementation Specification for SQL 1.1](#). s2.1.1.3
- ✔ Diese Methode setzt die SQL/MM-Spezifikation um. SQL-MM 3: 5.1.20
- ✔ Diese Funktion unterstützt 3d und lässt den Z-Index nicht fallen. Das Ergebnis wird jedoch nur mit XY berechnet. Die Z-Werte des Ergebnisses werden kopiert, gemittelt oder interpoliert.

## Beispiele



### Die Differenz der 2D-Linien.

```
SELECT ST_AsText (
  ST_Difference(
    'LINESTRING(50 100, 50 200)::geometry',
    'LINESTRING(50 50, 50 150)::geometry'
  )
);
```

```
st_astext
-----
LINESTRING(50 150,50 200)
```

### Die Differenz der 3D-Punkte.

```
SELECT ST_AsEWKT( ST_Difference(
  'MULTIPOINT(-118.58 38.38 5,-118.60 38.329 6,-118.614 38.281 7)' :: geometry,
  'POINT(-118.614 38.281 5)' :: geometry
));
```

```

) );

st_asewkt
-----
MULTIPOINT(-118.6 38.329 6,-118.58 38.38 5)

```

## Siehe auch

[ST\\_SymDifference](#), [ST\\_Intersection](#), [ST\\_Union](#)

### 7.13.3 ST\_Intersection

`ST_Intersection` — Berechnet eine Geometrie, die den gemeinsamen Teil der Geometrien A und B darstellt.

#### Synopsis

```

geometry ST_Intersection( geometry geomA , geometry geomB , float8 gridSize = -1 );
geography ST_Intersection( geography geogA , geography geogB );

```

#### Beschreibung

Gibt eine Geometrie zurück, die die Punktmengenüberschneidung zweier Geometrien darstellt. Mit anderen Worten: der Teil von Geometrie A und Geometrie B, der von den beiden Geometrien gemeinsam genutzt wird.

Wenn die Geometrien keine gemeinsamen Punkte haben (d. h. disjunkt sind), wird eine leere atomare Geometrie des entsprechenden Typs zurückgegeben.

Wenn das optionale Argument `gridSize` angegeben wird, werden die Eingaben auf ein Gitter der angegebenen Größe gerastert, und die Ergebnispunkte werden auf demselben Gitter berechnet. (Benötigt GEOS-3.9.0 oder höher)

`ST_Intersection` in Verbindung mit [ST\\_Intersects](#) ist nützlich, um Geometrien zu beschneiden, z. B. in Bounding Box-, Puffer- oder Regionsabfragen, bei denen Sie nur den Teil einer Geometrie benötigen, der sich innerhalb eines Landes oder einer Region von Interesse befindet.

#### Note



Für die Geografie ist dies eine dünne Hülle um die Geometrieimplementierung. Zunächst wird der beste SRID ermittelt, der in die Bounding Box der beiden geografischen Objekte passt (wenn die geografischen Objekte innerhalb einer halben UTM-Zone liegen, aber nicht in der gleichen UTM-Zone, wird eines davon ausgewählt) (wobei UTM oder der Nord-/Südpol von Lambert Azimuthal Equal Area (LAEA) bevorzugt werden und im schlimmsten Fall auf Mercator zurückgegriffen wird), und dann wird die Schnittmenge in diesem am besten passenden planaren Raumbezug ermittelt und zurück in WGS84-Geografie transformiert.



#### Warning

Mit dieser Funktion werden die M-Koordinatenwerte, falls vorhanden, gelöscht.



#### Warning

Wenn Sie mit 3D-Geometrien arbeiten, sollten Sie die auf SFGCAL basierende Funktion [ST\\_3DIntersection](#) verwenden, die für 3D-Geometrien eine korrekte 3D-Schnittmenge erstellt. Obwohl diese Funktion mit der Z-Koordinate arbeitet, führt sie eine Mittelung der Z-Koordinate durch.

Wird durch das GEOS Modul ausgeführt

Verbessert: 3.1.0 akzeptiert einen gridSize Parameter

Erfordert GEOS >= 3.9.0 zur Verwendung des Parameters gridSize

Geändert: 3.0.0 ist nicht von SFCGAL abhängig.

Verfügbarkeit: Mit Version 1.5 wurde die Unterstützung für den Datentyp Geographie eingeführt.

- ✔ Diese Methode implementiert die [OGC Simple Features Implementation Specification for SQL 1.1. s2.1.1.3](#)
- ✔ Diese Methode setzt die SQL/MM-Spezifikation um. SQL-MM 3: 5.1.18
- ✔ Diese Funktion unterstützt 3d und lässt den Z-Index nicht fallen. Das Ergebnis wird jedoch nur mit XY berechnet. Die Z-Werte des Ergebnisses werden kopiert, gemittelt oder interpoliert.

### Beispiele

```
SELECT ST_AsText(ST_Intersection('POINT(0 0)::geometry, 'LINESTRING ( 2 0, 0 2 )':: ↵
    geometry));
st_astext
-----
GEOMETRYCOLLECTION EMPTY

SELECT ST_AsText(ST_Intersection('POINT(0 0)::geometry, 'LINESTRING ( 0 0, 0 2 )':: ↵
    geometry));
st_astext
-----
POINT(0 0)
```

Schneiden Sie alle Linien (Trails) nach Land. Hier nehmen wir an, dass die Ländergeometrien POLYGON oder MULTIPOLYGON sind. HINWEIS: Wir behalten nur Schnittpunkte, die zu einem LINESTRING oder MULTILINESTRING führen, da wir uns nicht um Wege kümmern, die nur einen Punkt teilen. Der Dump wird benötigt, um eine Geometriesammlung in einzelne MULT\*-Teile zu zerlegen. Das folgende Beispiel ist recht allgemein gehalten und funktioniert auch für Polygone usw., indem einfach die Where-Klausel geändert wird.

```
select clipped.gid, clipped.f_name, clipped_geom
from (
    select trails.gid, trails.f_name,
        (ST_Dump(ST_Intersection(country.geom, trails.geom))).geom clipped_geom
    from country
        inner join trails on ST_Intersects(country.geom, trails.geom)
    ) as clipped
where ST_Dimension(clipped.clipped_geom) = 1;
```

Für Polygone, z. B. Polygon-Landmarken, können Sie auch den manchmal schnelleren Hack verwenden, dass das Puffern von allem um 0,0 außer einem Polygon zu einer leeren Geometriesammlung führt. (Eine Geometriesammlung mit Polygonen, Linien und Punkten, die mit 0,0 gepuffert wird, würde also nur die Polygone übrig lassen und die Sammlungshülle auflösen).

```
select poly.gid,
    ST_Multi(
        ST_Buffer(
            ST_Intersection(country.geom, poly.geom),
            0.0
        )
    ) clipped_geom
from country
    inner join poly on ST_Intersects(country.geom, poly.geom)
where not ST_IsEmpty(ST_Buffer(ST_Intersection(country.geom, poly.geom), 0.0));
```

**Beispiele: 2.5Gericht**

Beachten Sie, dass es sich hierbei nicht um eine echte Schnittmenge handelt, vergleichen Sie das gleiche Beispiel mit [ST\\_3DIntersection](#).

```
select ST_AsText(ST_Intersection(linestring, polygon)) As wkt
from ST_GeomFromText('LINESTRING Z (2 2 6,1.5 1.5 7,1 1 8,0.5 0.5 8,0 0 10)') AS ↔
  linestring
CROSS JOIN ST_GeomFromText('POLYGON((0 0 8, 0 1 8, 1 1 8, 1 0 8, 0 0 8))') AS polygon;

          st_astext
-----
LINESTRING Z (1 1 8,0.5 0.5 8,0 0 10)
```

**Siehe auch**

[ST\\_3DIntersection](#), [ST\\_Difference](#), [ST\\_Union](#), [ST\\_Dimension](#), [ST\\_Dump](#), [ST\\_Force2D](#), [ST\\_SymDifference](#), [ST\\_Intersects](#), [ST\\_Multi](#)

**7.13.4 ST\_MemUnion**

`ST_MemUnion` — Aggregatfunktion, die Geometrien auf eine speichereffiziente, aber langsamere Weise zusammenfasst

**Synopsis**

```
geometry ST_MemUnion(geometry set geomfield);
```

**Beschreibung**

Eine Aggregatfunktion, die die Eingabegeometrien vereinigt und zu einer überschneidungsfreien Ergebnisgeometrie zusammenführt. Die Ausgabe kann eine einzelne Geometrie, eine MultiGeometry oder eine Geometriesammlung sein.

**Note**

Ergibt das gleiche Ergebnis wie [ST\\_Union](#), benötigt aber weniger Speicher und mehr Prozessorzeit. Bei dieser Aggregatfunktion werden die Geometrien inkrementell vereinigt, im Gegensatz zum Aggregat `ST_Union`, das zunächst ein Array akkumuliert und dann den Inhalt mit einem schnellen Algorithmus vereinigt.



Diese Funktion unterstützt 3d und lässt den Z-Index nicht fallen. Das Ergebnis wird jedoch nur mit XY berechnet. Die Z-Werte des Ergebnisses werden kopiert, gemittelt oder interpoliert.

**Beispiele**

```
SELECT id,
       ST_MemUnion(geom) as singlegeom
FROM sometable f
GROUP BY id;
```

**Siehe auch**

[ST\\_Union](#)

### 7.13.5 ST\_Node

ST\_Node — Knoten eine Sammlung von Linien.

#### Synopsis

```
geometry ST_Node(geometry geom);
```

#### Beschreibung

Gibt einen (Multi)LineString zurück, der die vollständig nodierte Version einer Sammlung von Linestrings darstellt. Bei der Kodierung bleiben alle Eingabeknoten erhalten, und es werden so wenig neue Knoten wie möglich hinzugefügt. Das resultierende Linienwerk wird aufgelöst (doppelte Linien werden entfernt).

Dies ist ein guter Weg, um ein vollständig codiertes Linienwerk zu erstellen, das als Input für [ST\\_Polygonize](#) verwendet werden kann.

[ST\\_UnaryUnion](#) kann auch zum Verknüpfen und Auflösen von Linien verwendet werden. Es bietet die Möglichkeit, eine gridSize anzugeben, die eine einfachere und robustere Ausgabe ermöglichen kann. Siehe auch [ST\\_Union](#) für eine aggregierte Variante.



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

Wird vom GEOS Modul ausgeführt

Verfügbarkeit: 2.0.0

Geändert: 2.4.0 verwendet diese Funktion intern GEOSNode anstelle von GEOSUnaryUnion. Dies kann dazu führen, dass die resultierenden Linestrings eine andere Reihenfolge und Richtung haben als in PostGIS < 2.4.

#### Beispiele

Kodierung eines 3D-LineStrings, der sich selbst schneidet

```
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))
```

Kodierung zweier LineStrings, die ein gemeinsames Liniengerüst haben. Beachten Sie, dass das Liniengerüst des Ergebnisses aufgelöst wird.

```
SELECT ST_AsText (
    ST_Node('MULTILINESTRING ((2 5, 2 1, 7 1), (6 1, 4 1, 2 3, 2 5))::geometry')
) As output;
output
-----
MULTILINESTRING((2 5,2 3),(2 3,2 1,4 1),(4 1,2 3),(4 1,6 1),(6 1,7 1))
```

#### Siehe auch

[ST\\_UnaryUnion](#), [ST\\_AsBinary](#)

### 7.13.6 ST\_Split

ST\_Split — Gibt eine Sammlung von Geometrien zurück, die durch Aufteilung einer Geometrie durch eine andere Geometrie entstanden sind.

## Synopsis

geometry **ST\_Split**(geometry input, geometry blade);

## Beschreibung

Die Funktion unterstützt die Aufteilung eines LineStrings durch eine (Multi)Point-, (Multi)LineString- oder (Multi)Polygon-Grenze, oder eines (Multi)Polygons durch einen LineString. Wenn ein (Multi)Polygon als Klinge verwendet wird, werden seine linearen Komponenten (die Begrenzung) für die Aufteilung der Eingabe verwendet. Die Ergebnisgeometrie ist immer eine Sammlung.

Diese Funktion ist in gewisser Weise das Gegenteil von **ST\_Union**. Die Anwendung von **ST\_Union** auf die zurückgegebene Sammlung sollte theoretisch die ursprüngliche Geometrie ergeben (obwohl dies aufgrund numerischer Rundungen nicht unbedingt der Fall ist).



### Note

Wenn sich Eingabe und Schaufel aufgrund von Problemen mit der numerischen Präzision nicht überschneiden, wird die Eingabe möglicherweise nicht wie erwartet geteilt. Um dies zu vermeiden, kann es notwendig sein, die Eingabe zuerst an der Klinge zu fangen, indem Sie **ST\_Snap** mit einer kleinen Toleranz verwenden.

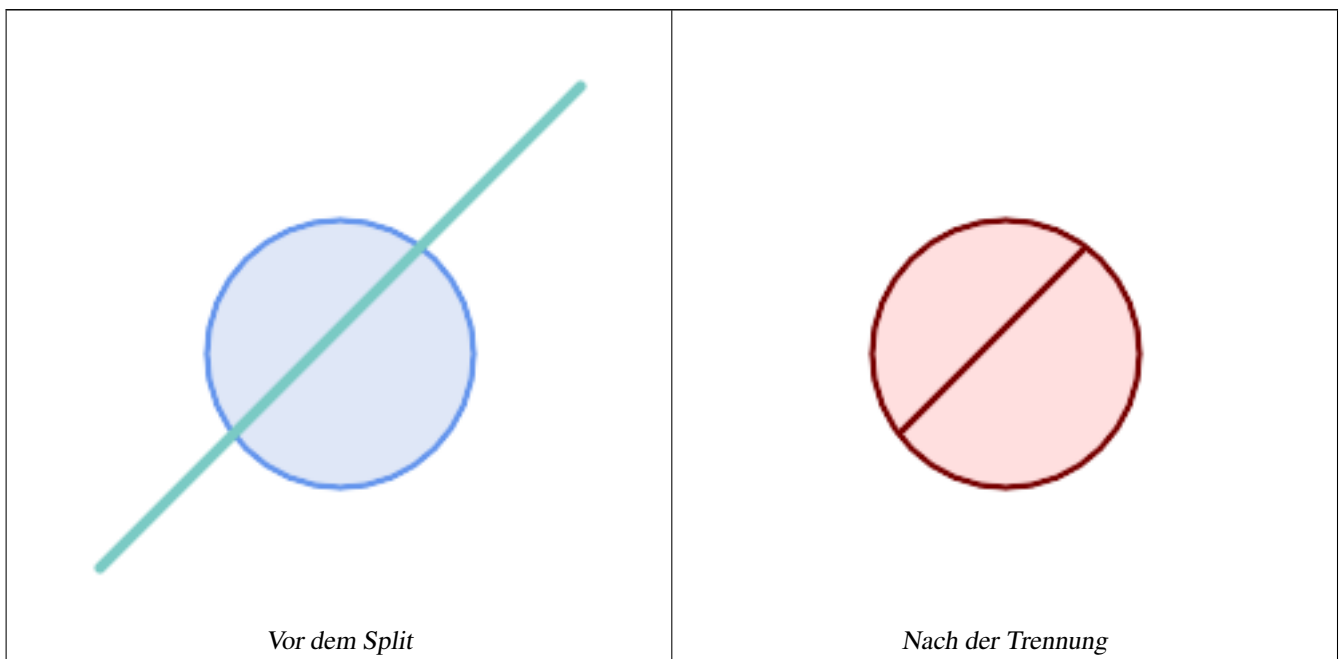
Verfügbarkeit: 2.0.0 erfordert GEOS

Verbessert: In Version 2.2.0 wurde die Unterstützung für die Aufteilung einer Linie durch eine Mehrlinien-, eine Mehrpunkt- oder eine (Mehr-)Polygonbegrenzung eingeführt.

Verbessert: In Version 2.5.0 wurde die Unterstützung für die Aufteilung eines Polygons durch eine Mehrlinie eingeführt.

## Beispiele

Ein Polygon durch eine Linie teilen.



```
SELECT ST_AsText( ST_Split(
                    ST_Buffer(ST_GeomFromText('POINT(100 90)'), 50), -- circle
```

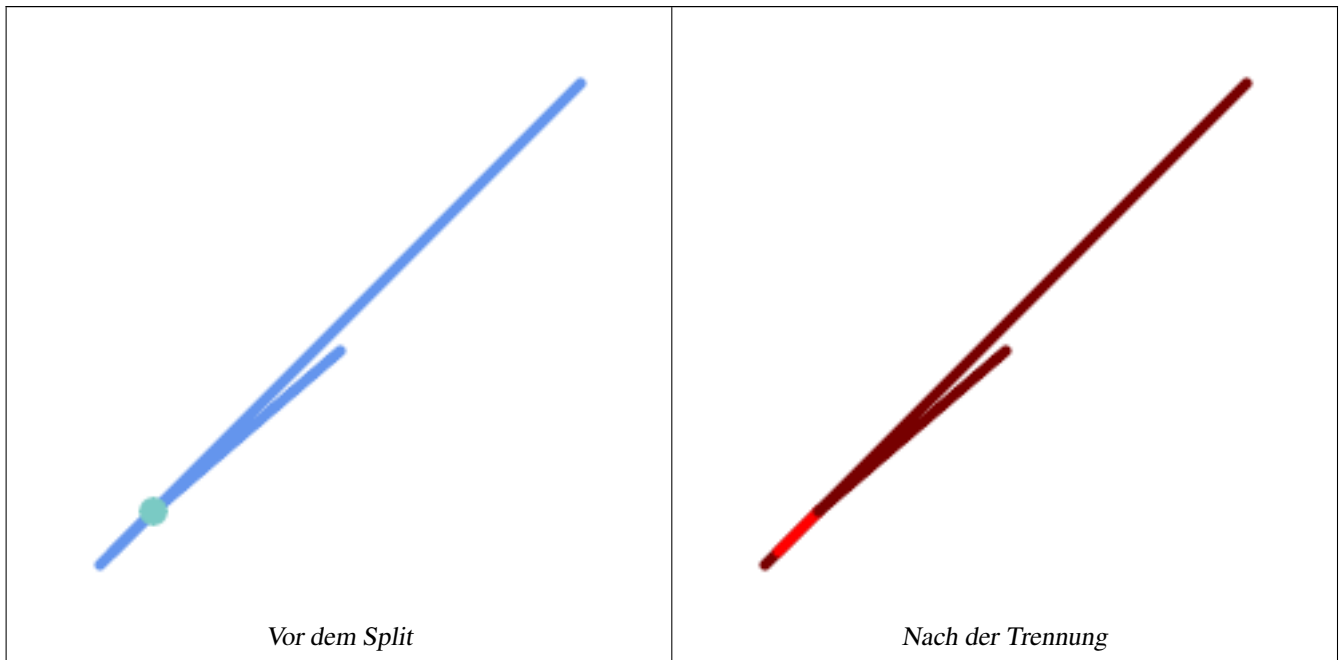


```

        ST_MakeLine(ST_Point(10, 10),ST_Point(190, 190)) -- line
    ));
-- result --
GEOMETRYCOLLECTION(
  POLYGON((150 90,149.039264020162 80.2454838991936,146.193976625564 ↵
    70.8658283817455,..),
  POLYGON(..)
)

```

Teilt einen MultiLineString durch einen Punkt, wobei der Punkt genau auf beiden LineString-Elementen liegt.



```

SELECT ST_AsText(ST_Split(
  'MULTILINESTRING((10 10, 190 190), (15 15, 30 30, 100 90))',
  ST_Point(30,30))) As split;

split
-----
GEOMETRYCOLLECTION(
  LINESTRING(10 10,30 30),
  LINESTRING(30 30,190 190),
  LINESTRING(15 15,30 30),
  LINESTRING(30 30,100 90)
)

```

Aufteilung eines LineString durch einen Punkt, wobei der Punkt nicht genau auf der Linie liegt. Zeigt die Verwendung von **ST\_Snap**, um die Linie an dem Punkt zu fangen, damit sie geteilt werden kann.

```

WITH data AS (SELECT
  'LINESTRING(0 0, 100 100)::geometry AS line,
  'POINT(51 50):: geometry AS point
)
SELECT ST_AsText( ST_Split( ST_Snap(line, point, 1), point)) AS snapped_split,
  ST_AsText( ST_Split(line, point)) AS not_snapped_not_split
FROM data;

                snapped_split |                               ↵
                not_snapped_not_split

```

```
GEOMETRYCOLLECTION(LINESTRING(0 0,51 50),LINESTRING(51 50,100 100)) | GEOMETRYCOLLECTION(↵  
LINESTRING(0 0,100 100))
```

## Siehe auch

[ST\\_Snap](#), [ST\\_AsBinary](#)

## 7.13.7 ST\_Subdivide

`ST_Subdivide` — Berechnet eine geradlinige Unterteilung einer Geometrie.

### Synopsis

```
setof geometry ST_Subdivide(geometry geom, integer max_vertices=256, float8 gridSize = -1);
```

### Beschreibung

Gibt eine Menge von Geometrien zurück, die das Ergebnis der Unterteilung von `geom` in Teile mit geradlinigen Linien sind, wobei jeder Teil nicht mehr als `max_vertices` enthält.

`max_vertices` muss 5 oder mehr sein, da 5 Punkte benötigt werden, um eine geschlossene Box darzustellen. `gridSize` kann angegeben werden, damit das Clipping im Raum mit fester Genauigkeit funktioniert (erfordert GEOS-3.9.0+).

Punkt-in-Polygon und andere räumliche Operationen sind bei indizierten, unterteilten Datensätzen normalerweise schneller. Da die Boundingboxen für die Teile in der Regel einen kleineren Bereich abdecken als die ursprüngliche Geometriebox, führen Indexabfragen zu weniger "Treffer"-Fällen. Die "Treffer"-Fälle sind schneller, weil die räumlichen Operationen, die von der Indexnachprüfung ausgeführt werden, weniger Punkte verarbeiten.



#### Note

Dies ist eine **set-returning function** (SRF), die eine Reihe von Zeilen mit einzelnen Geometriewerten zurückgibt. Sie kann in einer SELECT-Liste oder einer FROM-Klausel verwendet werden, um eine Ergebnismenge mit einem Datensatz für jede Ergebnisgeometrie zu erzeugen.

Wird vom GEOS Modul ausgeführt

Verfügbarkeit: 2.2.0

Verbessert: 2.5.0 verwendet vorhandene Punkte bei der Polygonaufteilung wieder, die Anzahl der Scheitelpunkte wurde von 8 auf 5 gesenkt.

Verbessert: 3.1.0 akzeptiert einen `gridSize`-Parameter.

Erfordert GEOS >= 3.9.0 zur Verwendung des Parameters `gridSize`

### Beispiele

**Beispiel:** Unterteilen Sie ein Polygon in Teile mit nicht mehr als 10 Scheitelpunkten, und weisen Sie jedem Teil eine eindeutige ID zu.



```
SELECT ST_AsText( ST_Subdivide(
    ST_Segmentize('LINESTRING(0 0, 85 85)::geography,
    1200000)::geometry, 8));
```

```
LINESTRING(0 0,0.487578359029357 5.57659056746196,0.984542144675897 ↔
    11.1527721155093,1.50101059639722 16.7281035483571,1.94532113630331 21.25)
LINESTRING(1.94532113630331 21.25,2.04869538062779 22.3020741387339,2.64204641967673 ↔
    27.8740533545155,3.29994062412787 33.443216802941,4.04836719489742 ↔
    39.0084282520239,4.59890468420694 42.5)
LINESTRING(4.59890468420694 42.5,4.92498503922732 44.5680389206321,5.98737409390639 ↔
    50.1195229244701,7.3290919767674 55.6587646879025,8.79638749938413 60.1969505994924)
LINESTRING(8.79638749938413 60.1969505994924,9.11375579533779 ↔
    61.1785363177625,11.6558166691368 66.6648504160202,15.642041247655 ↔
    72.0867690601745,22.8716627200212 77.3609628116894,24.6991785131552 77.8939011989848)
LINESTRING(24.6991785131552 77.8939011989848,39.4046096622744 ↔
    82.1822848017636,44.7994523421035 82.5156766227011)
LINESTRING(44.7994523421035 82.5156766227011,85 85)
```

**Beispiel:** Die komplexen Geometrien einer Tabelle werden an Ort und Stelle unterteilt. Die ursprünglichen Geometriedatensätze werden aus der Quelltable gelöscht, und neue Datensätze für jede unterteilte Ergebnisgeometrie werden eingefügt.

```
WITH complex_areas_to_subdivide AS (
    DELETE from polygons_table
    WHERE ST_NPoints(geom)
> 255
    RETURNING id, column1, column2, column3, geom
)
INSERT INTO polygons_table (fid, column1, column2, column3, geom)
SELECT fid, column1, column2, column3,
    ST_Subdivide(geom, 255) as geom
FROM complex_areas_to_subdivide;
```

**Beispiel:** Erstellen einer neuen Tabelle mit unterteilten Geometrien unter Beibehaltung des Schlüssels der ursprünglichen Geometrie, so dass die neue Tabelle mit der Quelltable verbunden werden kann. Da `ST_Subdivide` eine mengenrückgebende (Tabellen-)Funktion ist, die eine Menge von Einzelwertzeilen zurückgibt, erzeugt diese Syntax automatisch eine Tabelle mit einer Zeile für jedes Ergebnisteil.

```
CREATE TABLE subdivided_geoms AS
SELECT pkey, ST_Subdivide(geom) AS geom
FROM original_geoms;
```

## Siehe auch

[ST\\_ClipByBox2D](#), [ST\\_Segmentize](#), [ST\\_Split](#), [ST\\_NPoints](#)

## 7.13.8 ST\_SymDifference

`ST_SymDifference` — Berechnet eine Geometrie, die die Teile der Geometrien A und B darstellt, die sich nicht überschneiden.

### Synopsis

geometry **ST\_SymDifference**(geometry geomA, geometry geomB, float8 gridSize = -1);

## Beschreibung

Gibt eine Geometrie zurück, die die Teile der Geodäten A und B darstellt, die sich nicht schneiden. Dies ist äquivalent zu  $ST\_Union(A,B) - ST\_Intersection(A,B)$ . Es wird als symmetrische Differenz bezeichnet, weil  $ST\_SymDifference(A,B) = ST\_SymDifference(B,A)$ .

Wenn das optionale Argument `gridSize` angegeben wird, werden die Eingaben auf ein Gitter der angegebenen Größe gerastert, und die Ergebnispunkte werden auf demselben Gitter berechnet. (Benötigt GEOS-3.9.0 oder höher)

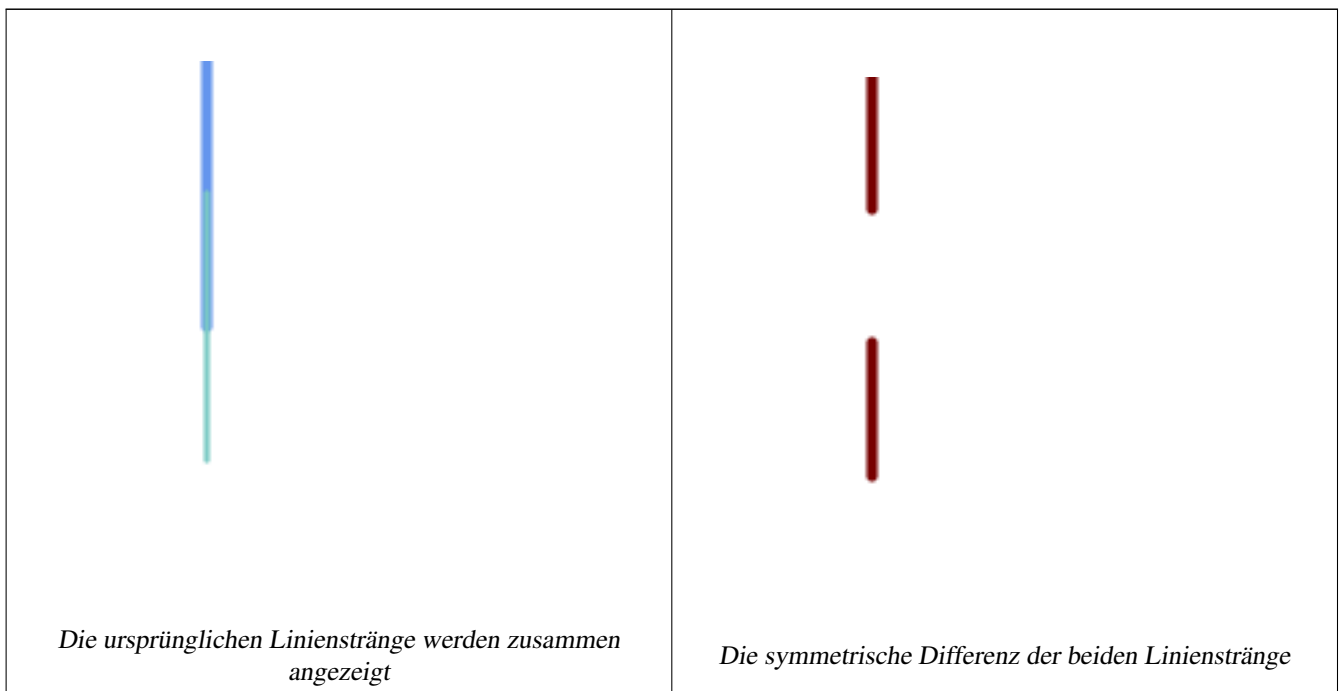
Wird durch das GEOS Modul ausgeführt

Verbessert: 3.1.0 akzeptiert einen `gridSize`-Parameter.

Erfordert GEOS  $\geq$  3.9.0 zur Verwendung des Parameters `gridSize`

- ✔ Diese Methode implementiert die [OGC Simple Features Implementation Specification for SQL 1.1](#). s2.1.1.3
- ✔ Diese Methode setzt die SQL/MM-Spezifikation um. SQL-MM 3: 5.1.21
- ✔ Diese Funktion unterstützt 3d und lässt den Z-Index nicht fallen. Das Ergebnis wird jedoch nur mit XY berechnet. Die Z-Werte des Ergebnisses werden kopiert, gemittelt oder interpoliert.

## Beispiele



```
--Safe for 2d - symmetric difference of 2 linestrings
SELECT ST_AsText (
  ST_SymDifference (
    ST_GeomFromText ('LINESTRING(50 100, 50 200)'),
    ST_GeomFromText ('LINESTRING(50 50, 50 150)')
  )
);

st_astext
-----
MULTILINESTRING((50 150,50 200),(50 50,50 100))
```

```
--When used in 3d doesn't quite do the right thing
SELECT ST_AsEWKT(ST_SymDifference(ST_GeomFromEWKT('LINESTRING(1 2 1, 1 4 2)'),
    ST_GeomFromEWKT('LINESTRING(1 1 3, 1 3 4)'))

st_astext
-----
MULTILINESTRING((1 3 2.75,1 4 2), (1 1 3,1 2 2.25))
```

**Siehe auch**

[ST\\_Difference](#), [ST\\_Intersection](#), [ST\\_Union](#)

**7.13.9 ST\_UnaryUnion**

`ST_UnaryUnion` — Berechnet die Vereinigung der Komponenten einer einzelnen Geometrie.

**Synopsis**

```
geometry ST_UnaryUnion(geometry geom, float8 gridSize = -1);
```

**Beschreibung**

Eine Variante von [ST\\_Union](#) mit einer einzigen Eingabe. Die Eingabe kann eine einzelne Geometrie, eine MultiGeometry oder eine GeometryCollection sein. Die Vereinigung wird auf die einzelnen Elemente der Eingabe angewendet.

Diese Funktion kann verwendet werden, um MultiPolygone zu korrigieren, die aufgrund von überlappenden Komponenten ungültig sind. Allerdings müssen die Eingabekomponenten jeweils gültig sein. Eine ungültige Eingabekomponente, wie z.B. ein Schleifenpolygon, kann einen Fehler verursachen. Aus diesem Grund kann es besser sein, [ST\\_MakeValid](#) zu verwenden.

Eine weitere Anwendung dieser Funktion ist das Verknüpfen und Auflösen einer Sammlung von Linien, die sich kreuzen oder überlappen, um sie **einfach** zu machen. ([ST\\_Node](#) tut dies auch, bietet aber nicht die Option `gridSize`.)

Es ist möglich, `ST_UnaryUnion` mit [ST\\_Collect](#) zu kombinieren, um eine Feinabstimmung darüber vorzunehmen, wie viele Geometrien auf einmal vereinigt werden sollen. Dies ermöglicht eine Abwägung zwischen Speichernutzung und Rechenzeit, wobei ein Gleichgewicht zwischen `ST_Union` und [ST\\_MemUnion](#) hergestellt wird.

Wenn das optionale Argument `gridSize` angegeben wird, werden die Eingaben auf ein Gitter der angegebenen Größe gerastert, und die Ergebnispunkte werden auf demselben Gitter berechnet. (Benötigt GEOS-3.9.0 oder höher)



Diese Funktion unterstützt 3d und lässt den Z-Index nicht fallen. Das Ergebnis wird jedoch nur mit XY berechnet. Die Z-Werte des Ergebnisses werden kopiert, gemittelt oder interpoliert.

Verbessert: 3.1.0 akzeptiert einen `gridSize`-Parameter.

Erfordert GEOS >= 3.9.0 zur Verwendung des Parameters `gridSize`

Verfügbarkeit: 2.0.0

**Siehe auch**

[ST\\_Union](#), [ST\\_MemUnion](#), [ST\\_MakeValid](#), [ST\\_Collect](#), [ST\\_Node](#)

**7.13.10 ST\_Union**

`ST_Union` — Berechnet eine Geometrie, die die Punktmengenvereinigung der Eingabegeometrien darstellt.

## Synopsis

```
geometry ST_Union(geometry g1, geometry g2);  
geometry ST_Union(geometry g1, geometry g2, float8 gridSize);  
geometry ST_Union(geometry[] g1_array);  
geometry ST_Union(geometry set g1field);  
geometry ST_Union(geometry set g1field, float8 gridSize);
```

## Beschreibung

Vereinigt die Eingabegeometrien und führt die Geometrien zusammen, um eine überschneidungsfreie Ergebnisgeometrie zu erzeugen. Die Ausgabe kann eine atomare Geometrie, eine MultiGeometry oder eine Geometriesammlung sein. Gibt es in mehreren Varianten:

**Variante mit zwei Eingaben:** gibt eine Geometrie zurück, die die Vereinigung von zwei Eingabegeometrien ist. Wenn eine der beiden Eingaben NULL ist, wird NULL zurückgegeben.

**Array-Variante:** gibt eine Geometrie zurück, die die Vereinigung eines Arrays von Geometrien ist.

**Aggregat-Variante:** gibt eine Geometrie zurück, die die Vereinigung eines Rowsets von Geometrien ist. Die Funktion `ST_Union()` ist eine "Aggregat"-Funktion in der Terminologie von PostgreSQL. Das bedeutet, dass sie mit Datenzeilen arbeitet, so wie es auch die Funktionen `SUM()` und `AVG()` tun, und wie die meisten Aggregate ignoriert sie auch NULL-Geometrien.

Siehe [ST\\_UnaryUnion](#) für eine nicht aggregierte Variante mit einem Eingang.

Die `ST_Union` Array- und Set-Varianten verwenden den schnellen Cascaded Union-Algorithmus, der in <http://blog.cleverelephant.ca/2009/01/must-faster-unions-in-postgis-14.html> beschrieben ist.

Ein `gridSize` kann angegeben werden, um im Raum mit fester Genauigkeit zu arbeiten. Die Eingaben werden auf ein Gitter der angegebenen Größe gerastert, und die Ergebnispunkte werden auf demselben Gitter berechnet. (Benötigt GEOS-3.9.0 oder höher)



### Note

`ST_Collect` kann manchmal anstelle von `ST_Union` verwendet werden, wenn das Ergebnis nicht überlappungsfrei sein muss. `ST_Collect` ist in der Regel schneller als `ST_Union`, da es keine Verarbeitung der gesammelten Geometrien vornimmt.

Wird vom GEOS Modul ausgeführt

`ST_Union` erzeugt `MultiLineString` und fügt `LineStrings` nicht zu einem einzigen `LineString` zusammen. Verwenden Sie [ST\\_LineMerge](#), um `LineStrings` zusammenzufügen.

HINWEIS: Diese Funktion hieß früher `GeomUnion()` und wurde von "Union" umbenannt, da `UNION` ein reserviertes SQL-Wort ist.

Verbessert: 3.1.0 akzeptiert einen `gridSize`-Parameter.

Erfordert GEOS >= 3.9.0 zur Verwendung des Parameters `gridSize`

Geändert: 3.0.0 ist nicht von SFCGAL abhängig.

Verfügbarkeit: 1.4.0 - `ST_Union` wurde verbessert. `ST_Union(geometry array)` wurde eingeführt und auch schnellere Aggregat-Sammlung in PostgreSQL.



Diese Methode implementiert die [OGC Simple Features Implementation Specification for SQL 1.1](#). s2.1.1.3



### Note

Die Aggregatversion ist in der OGC SPEC nicht explizit definiert.



Diese Methode setzt die SQL/MM-Spezifikation um. SQL-MM 3: 5.1.19 der z-index (Elevation), wenn Polygone beteiligt sind.



Diese Funktion unterstützt 3d und lässt den Z-Index nicht fallen. Das Ergebnis wird jedoch nur mit XY berechnet. Die Z-Werte des Ergebnisses werden kopiert, gemittelt oder interpoliert.

## Beispiele

### Beispiel für ein Aggregat

```
SELECT id,
       ST_Union(geom) as singlegeom
FROM sometable f
GROUP BY id;
```

### Nicht-Aggregat-Beispiel

```
select ST_AsText(ST_Union('POINT(1 2)' :: geometry, 'POINT(-2 3)' :: geometry))

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

select ST_AsText(ST_Union('POINT(1 2)' :: geometry, 'POINT(1 2)' :: geometry))

st_astext
-----
POINT(1 2)
```

### 3D-Beispiel - unterstützt gewissermaßen 3D (und mit gemischten Dimensionen!)

```
select ST_AsEWKT(ST_Union(geom))
from (
  select 'POLYGON((-7 4.2,-7.1 4.2,-7.1 4.3, -7 4.2))'::geometry geom
  union all
  select 'POINT(5 5 5)'::geometry geom
  union all
  select 'POINT(-2 3 1)'::geometry geom
  union all
  select 'LINESTRING(5 5 5, 10 10 10)'::geometry geom
) as foo;

st_asewkt
-----
GEOMETRYCOLLECTION(POINT(-2 3 1),LINESTRING(5 5 5,10 10 10),POLYGON((-7 4.2 5,-7.1 4.2 5,-7.1 4.3 5,-7 4.2 5)));
```

### 3d-Beispiel ohne Vermischung der Dimensionen

```
select ST_AsEWKT(ST_Union(geom))
from (
  select 'POLYGON((-7 4.2 2,-7.1 4.2 3,-7.1 4.3 2, -7 4.2 2))'::geometry geom
  union all
  select 'POINT(5 5 5)'::geometry geom
  union all
  select 'POINT(-2 3 1)'::geometry geom
  union all
  select 'LINESTRING(5 5 5, 10 10 10)'::geometry geom
) as foo;
```



```

st_asewkt
-----
GEOMETRYCOLLECTION(POINT(-2 3 1),LINESTRING(5 5 5,10 10 10),POLYGON((-7 4.2 2,-7.1 4.2 2,
3,-7.1 4.3 2,-7 4.2 2)))

--Examples using new Array construct
SELECT ST_Union(ARRAY(SELECT geom FROM sometable));

SELECT ST_AsText(ST_Union(ARRAY[ST_GeomFromText('LINESTRING(1 2, 3 4)'),
ST_GeomFromText('LINESTRING(3 4, 4 5)')])) As wktunion;

--wktunion---
MULTILINESTRING((3 4,4 5),(1 2,3 4))

```

## Siehe auch

[ST\\_Collect](#), [ST\\_UnaryUnion](#), [ST\\_MemUnion](#), [ST\\_Intersection](#), [ST\\_Difference](#), [ST\\_SymDifference](#)

## 7.14 Geometrieverarbeitung

### 7.14.1 ST\_Buffer

**ST\_Buffer** — Berechnet eine Geometrie, die alle Punkte innerhalb eines bestimmten Abstands zu einer Geometrie umfasst.

#### Synopsis

```

geometry ST_Buffer(geometry g1, float radius_of_buffer, text buffer_style_parameters = '');
geometry ST_Buffer(geometry g1, float radius_of_buffer, integer num_seg_quarter_circle);
geography ST_Buffer(geography g1, float radius_of_buffer, text buffer_style_parameters);
geography ST_Buffer(geography g1, float radius_of_buffer, integer num_seg_quarter_circle);

```

#### Beschreibung

Berechnet ein POLYGON oder MULTIPOLYGON, das alle Punkte darstellt, deren Abstand zu einer Geometrie/Geografie kleiner oder gleich einem bestimmten Abstand ist. Ein negativer Abstand verkleinert die Geometrie, anstatt sie zu vergrößern. Ein negativer Abstand kann ein Polygon vollständig schrumpfen lassen; in diesem Fall wird POLYGON EMPTY zurückgegeben. Für Punkte und Linien geben negative Abstände immer leere Ergebnisse zurück.

Bei Geometrie wird die Entfernung in den Einheiten des räumlichen Bezugssystems der Geometrie angegeben. Für die Geografie wird die Entfernung in Metern angegeben.

Der optionale dritte Parameter steuert die Genauigkeit und den Stil des Puffers. Die Genauigkeit der Kreisbögen im Puffer wird als die Anzahl der Liniensegmente angegeben, die zur Annäherung an einen Viertelkreis verwendet werden (Standardwert ist 8). Der Pufferstil kann durch Angabe einer Liste von durch Leerzeichen getrennten Schlüssel=Wert-Paaren wie folgt spezifiziert werden:

- `quad_segs=#` : Anzahl der Liniensegmente, die zur Annäherung an einen Viertelkreis verwendet werden (Standard ist 8).
- `endcap=round|flatsquare` : Stil der Endkappe (Standardwert ist "round"); "butt" wird als Synonym für "flat" akzeptiert.
- `join=round|mitre|bevel` : Art der Verbindung (Standardeinstellung ist "round"). 'miter' wird als Synonym für 'mitre' akzeptiert.
- `'mitre_limit=#.#'` : Begrenzung des Gehrungsverhältnisses (wirkt sich nur auf Gehrungsverbindungen aus). 'miter\_limit' wird als Synonym für 'mitre\_limit' akzeptiert.

- `side=both|left|right` : 'left' oder 'right' führt eine einseitige Pufferung der Geometrie durch, wobei sich die gepufferte Seite auf die Richtung der Linie bezieht. Dies gilt nur für LINESTRING-Geometrien und wirkt sich nicht auf POINT- oder POLYGON-Geometrien aus. Standardmäßig sind die Endkappen quadratisch.

**Note****Note!**

Für die Geografie ist dies eine dünne Hülle um die Geometrieimplementierung. Es wird ein planares räumliches Bezugssystem bestimmt, das am besten zum Begrenzungsrahmen des geografischen Objekts passt (versucht werden UTM, Lambert Azimuthal Equal Area (LAEA) Nord-/Südpol und schließlich Mercator). Der Puffer wird im planaren Raum berechnet und dann in WGS84 zurücktransformiert. Dies führt möglicherweise nicht zu dem gewünschten Verhalten, wenn das Eingabeobjekt viel größer als eine UTM-Zone ist oder die Datumsgrenze überschreitet.

**Note****Note!**

Die Ausgabe von Buffer ist immer eine gültige polygonale Geometrie. Buffer kann mit ungültigen Eingaben umgehen, daher wird manchmal die Pufferung um den Abstand 0 verwendet, um ungültige Polygone zu reparieren. `ST_MakeValid` kann auch für diesen Zweck verwendet werden.

**Note****Note!**

Die Pufferung wird manchmal verwendet, um eine Suche innerhalb der Entfernung durchzuführen. Für diesen Anwendungsfall ist es effizienter, `ST_DWithin` zu verwenden.

**Note****Note!**

Diese Funktion ignoriert die Z-Dimension. Sie liefert immer ein 2D-Ergebnis, auch wenn sie auf eine 3D-Geometrie angewendet wird.

Erweiterung: 2.5.0 - `ST_Buffer` ermöglicht jetzt auch eine seitliche Pufferzonenberechnung über `side=both|left|right`.

Verfügbarkeit: 1.5 - `ST_Buffer` wurde um die Unterstützung von Abschlusstücken/endcaps und Join-Typen erweitert. Diese können zum Beispiel dazu verwendet werden, um Linienzüge von Straßen in Straßenpolygone mit flachen oder rechtwinkligen Abschlüssen anstatt mit runden Enden umzuwandeln. Ein schlanker Adapter für den geographischen Datentyp wurde hinzugefügt.

Wird vom GEOS Modul ausgeführt

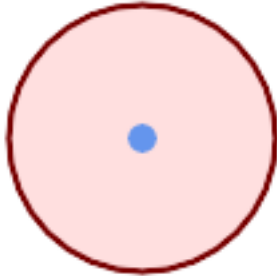


Diese Methode implementiert die [OGC Simple Features Implementation Specification for SQL 1.1](#). s2.1.1.3



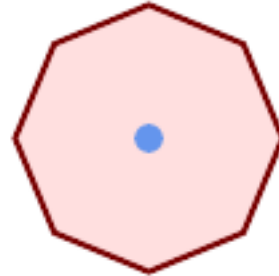
Diese Methode setzt die SQL/MM-Spezifikation um. SQL-MM IEC 13249-3: 5.1.30

**Beispiele**



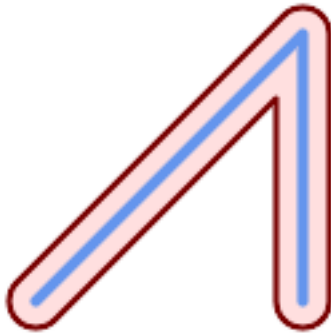
*quad\_segs=8 (Standardwert)*

```
SELECT ST_Buffer(
  ST_GeomFromText('POINT(100 90)'),
  50, 'quad_segs=8');
```



*quad\_segs=2 (lahme Ente)*

```
SELECT ST_Buffer(
  ST_GeomFromText('POINT(100 90)'),
  50, 'quad_segs=2');
```



*endcap=round join=round (Standardwert)*

```
SELECT ST_Buffer(
  ST_GeomFromText(
    'LINESTRING(50 50,150 150,150 50)'
  ), 10, 'endcap=round join=round');
```



*endcap=square*

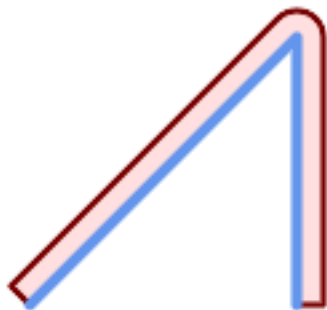
```
SELECT ST_Buffer(
  ST_GeomFromText(
    'LINESTRING(50 50,150 150,150 50)'
  ), 10, 'endcap=square join=round');
```

*join=bevel*

```
SELECT ST_Buffer(
  ST_GeomFromText(
    'LINESTRING(50 50,150 150,150 50)'
  ), 10, 'join=bevel');
```

*join=mitre mitre\_limit=5.0 (default mitre limit)*

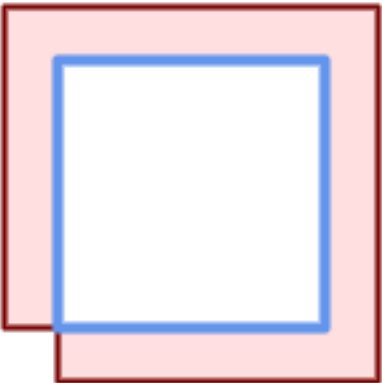
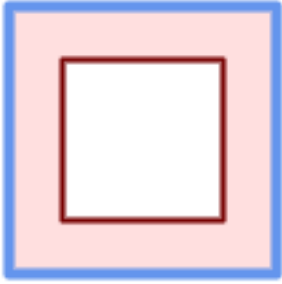
```
SELECT ST_Buffer(
  ST_GeomFromText(
    'LINESTRING(50 50,150 150,150 50)'
  ), 10, 'join=mitre mitre_limit=5.0');
```

*side=left*

```
SELECT ST_Buffer(
  ST_GeomFromText(
    'LINESTRING(50 50,150 150,150 50)'
  ), 10, 'side=left');
```

*side=right*

```
SELECT ST_Buffer(
  ST_GeomFromText(
    'LINESTRING(50 50,150 150,150 50)'
  ), 10, 'side=right');
```

	
<p><i>right-hand-winding, polygon boundary side=left</i></p>	<p><i>right-hand-winding, polygon boundary side=right</i></p>
<pre>SELECT ST_Buffer( ST_ForceRHR( ST_Boundary(   ST_GeomFromText( 'POLYGON ((50 50, 50 150, 150 150, 150 50, 50 50))'),   ), 20, 'side=left');</pre>	<pre>SELECT ST_Buffer( ST_ForceRHR( ST_Boundary(   ST_GeomFromText( 'POLYGON ((50 50, 50 150, 150 150, 150 50, 50 50))'),   ), 20, 'side=right')</pre>

```
--A buffered point approximates a circle
-- A buffered point forcing approximation of (see diagram)
-- 2 points per quarter circle is poly with 8 sides (see diagram)
SELECT ST_NPoints(ST_Buffer(ST_GeomFromText('POINT(100 90)'), 50)) As promisingcircle_pcount,
ST_NPoints(ST_Buffer(ST_GeomFromText('POINT(100 90)'), 50, 2)) As lamecircle_pcount;

promisingcircle_pcount | lamecircle_pcount
-----+-----
                33 |                9

--A lighter but lamer circle
-- only 2 points per quarter circle is an octagon
--Below is a 100 meter octagon
-- Note coordinates are in NAD 83 long lat which we transform
to Mass state plane meter and then buffer to get measurements in meters;
SELECT ST_AsText(ST_Buffer(
ST_Transform(
ST_SetSRID(ST_Point(-71.063526, 42.35785), 4269), 26986)
, 100, 2)) As octagon;
-----
POLYGON((236057.59057465 900908.759918696,236028.301252769 900838.049240578,235
957.59057465 900808.759918696,235886.879896532 900838.049240578,235857.59057465
900908.759918696,235886.879896532 900979.470596815,235957.59057465 901008.759918
696,236028.301252769 900979.470596815,236057.59057465 900908.759918696))
```

**Siehe auch**

[ST\\_Collect](#), [ST\\_DWithin](#), [ST\\_SetSRID](#), [ST\\_Transform](#), [ST\\_Union](#), [ST\\_MakeValid](#)

## 7.14.2 ST\_BuildArea

ST\_BuildArea — Erzeugt eine polygonale Geometrie, die aus dem Linienwerk einer Geometrie gebildet wird.

### Synopsis

```
geometry ST_BuildArea(geometry geom);
```

### Beschreibung

Erzeugt eine flächige Geometrie, die aus den konstituierenden Linienzügen der Eingabegeometrie gebildet wird. Die Eingabe kann ein LineString, MultiLineString, Polygon, MultiPolygon oder eine GeometryCollection sein. Das Ergebnis ist ein Polygon oder MultiPolygon, je nach Eingabe. Wenn das eingegebene Liniengerüst keine Polygone bildet, wird NULL zurückgegeben.

Im Gegensatz zu [ST\\_MakePolygon](#) akzeptiert diese Funktion Ringe, die aus mehreren Linien bestehen, und kann eine beliebige Anzahl von Polygonen bilden.

Diese Funktion wandelt innere Ringe in Löcher um. Um auch innere Ringe in Polygone zu verwandeln, verwenden Sie [ST\\_Polygonize](#).



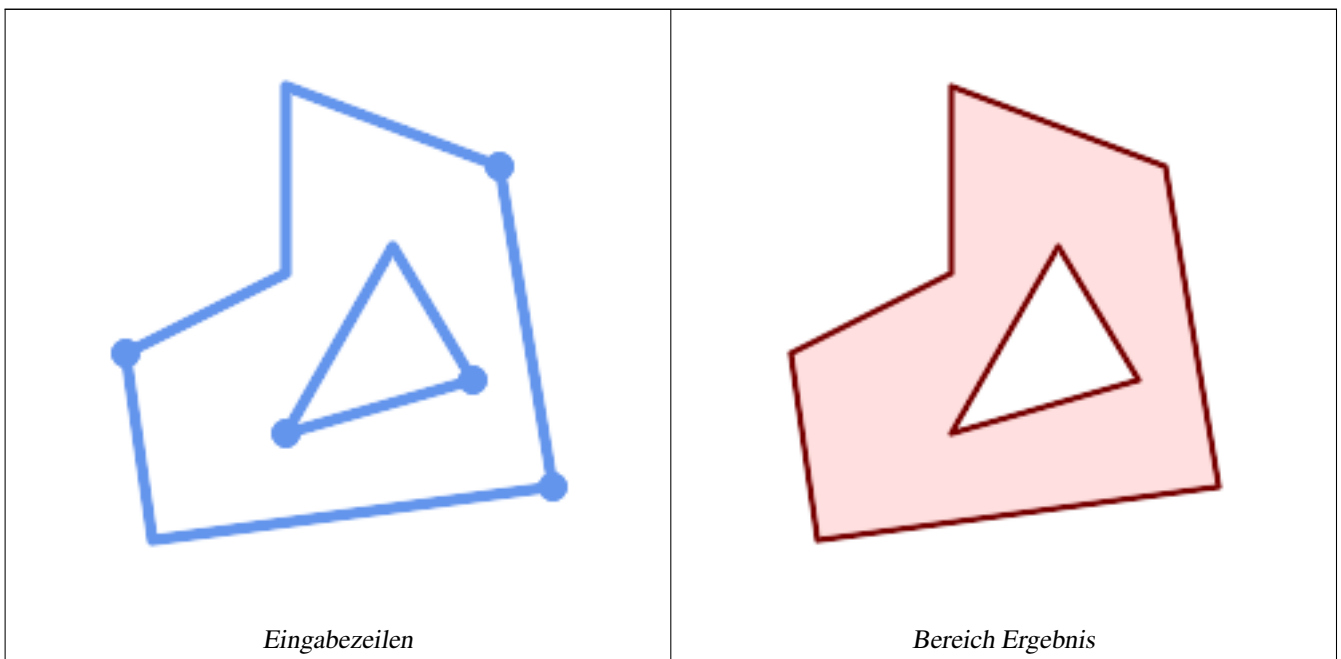
#### Note

Damit diese Funktion ordnungsgemäß funktioniert, muss das eingegebene Linienwerk korrekt genodet sein. [ST\\_Node](#) kann zum Knoten von Linien verwendet werden.

Wenn sich das eingegebene Linienwerk kreuzt, erzeugt diese Funktion ungültige Polygone. [ST\\_MakeValid](#) kann verwendet werden, um sicherzustellen, dass die Ausgabe gültig ist.

Verfügbarkeit: 1.1.0

### Beispiele



```
WITH data(geom) AS (VALUES
  ('LINESTRING (180 40, 30 20, 20 90)'::geometry)
```

```

, ('LINESTRING (180 40, 160 160)::geometry)
, ('LINESTRING (160 160, 80 190, 80 120, 20 90)::geometry)
, ('LINESTRING (80 60, 120 130, 150 80)::geometry)
, ('LINESTRING (80 60, 150 80)::geometry)
)
SELECT ST_AsText( ST_BuildArea( ST_Collect( geom )))
FROM data;

```

```

-----
POLYGON((180 40,30 20,20 90,80 120,80 190,160 160,180 40),(150 80,120 130,80 60,150 80))

```



### *Erstellen eines Donuts aus zwei kreisförmigen Polygonen*

```

SELECT ST_BuildArea(ST_Collect(inring,outring))
FROM (SELECT
  ST_Buffer('POINT(100 90)', 25) As inring,
  ST_Buffer('POINT(100 90)', 50) As outring) As t;

```

#### **Siehe auch**

[ST\\_Collect](#), [ST\\_MakePolygon](#), [ST\\_MakeValid](#), [ST\\_Node](#), [ST\\_Polygonize](#), [ST\\_BdPolyFromText](#), [ST\\_BdMPolyFromText](#) (Wrapper zu dieser Funktion mit Standard OGC Schnittstelle)

### **7.14.3 ST\_Centroid**

`ST_Centroid` — Gibt den geometrischen Schwerpunkt einer Geometrie zurück.

#### **Synopsis**

```

geometry ST_Centroid(geometry g1);
geography ST_Centroid(geography g1, boolean use_spheroid = true);

```

#### **Beschreibung**

Berechnet einen Punkt, der den geometrischen Schwerpunkt einer Geometrie darstellt. Für `[MULTI]POINTS` ist der Schwerpunkt das arithmetische Mittel der eingegebenen Koordinaten. Für `[MULTI]LINESTRINGS` wird der Schwerpunkt anhand der

gewichteten Länge der einzelnen Liniensegmente berechnet. Für [MULTI]POLYGONS wird der Schwerpunkt anhand der Fläche berechnet. Wenn eine leere Geometrie übergeben wird, wird eine leere `GEOMETRYCOLLECTION` zurückgegeben. Wenn `NULL` übergeben wird, wird `NULL` zurückgegeben. Wenn `CIRCULARSTRING` oder `COMPOUNDCURVE` übergeben werden, werden sie zuerst mit `CurveToLine` in einen `Linestring` umgewandelt, dann genauso wie bei `LINestring`.

Bei gemischtdimensionalen Eingaben entspricht das Ergebnis dem Schwerpunkt der Komponente Geometrien mit der höchsten Dimension (da die Geometrien mit geringerer Dimension kein "Gewicht" zum Schwerpunkt beitragen).

Beachten Sie, dass bei polygonalen Geometrien der Schwerpunkt nicht unbedingt im Inneren des Polygons liegt. Siehe zum Beispiel das folgende Diagramm des Schwerpunkts eines C-förmigen Polygons. Um einen Punkt zu konstruieren, der garantiert im Inneren eines Polygons liegt, verwenden Sie `ST_PointOnSurface`.

Neu in 2.3.0 : unterstützt `CIRCULARSTRING` und `COMPOUNDCURVE` (mit `CurveToLine`)

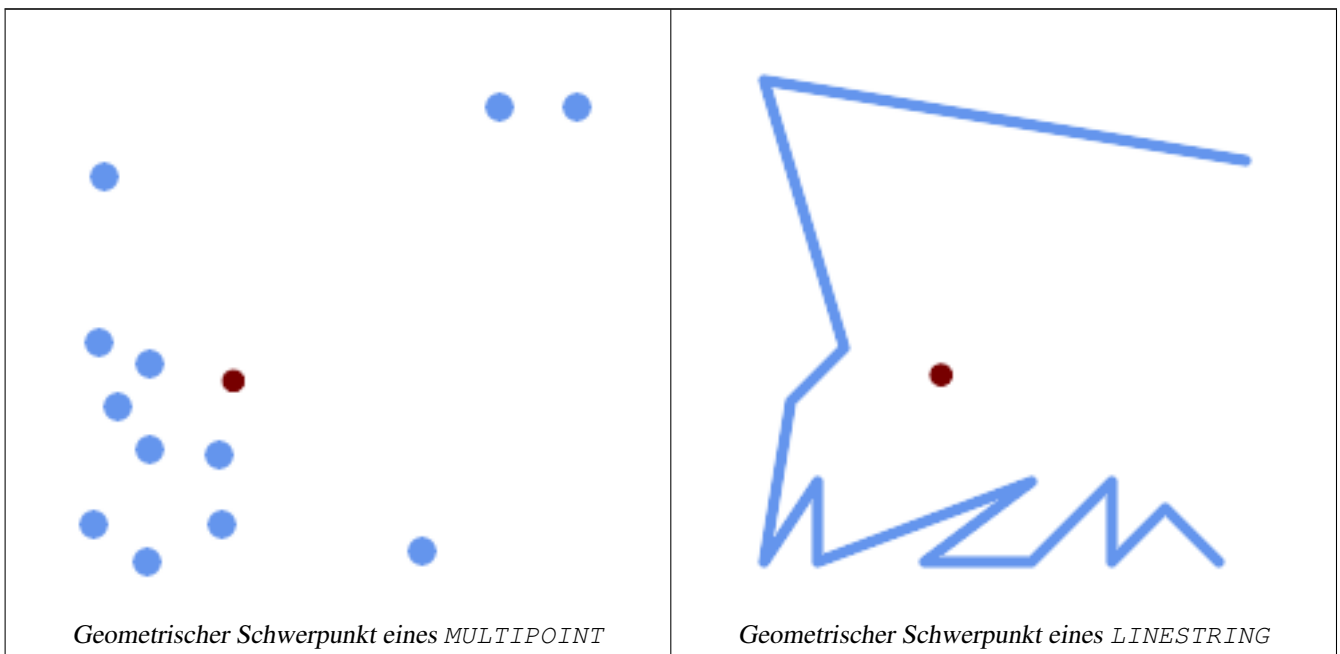
Verfügbarkeit: Mit 2.4.0 wurde die Unterstützung für den geografischen Datentyp eingeführt.

✔ Diese Methode implementiert die [OGC Simple Features Implementation Specification for SQL 1.1](#).

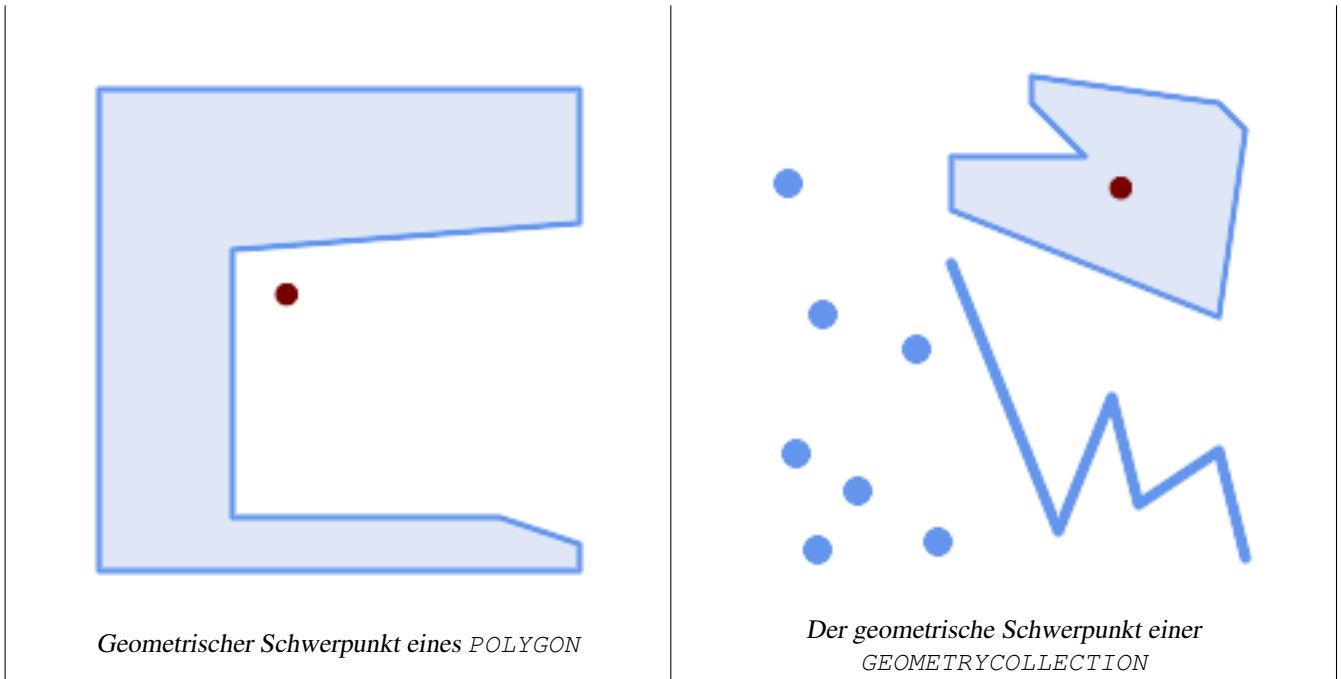
✔ Diese Methode setzt die SQL/MM-Spezifikation um. SQL-MM 3: 8.1.4, 9.5.5

### Beispiele

In den folgenden Abbildungen ist der rote Punkt der Schwerpunkt der Quellengeometrie.







```
SELECT ST_AsText(ST_Centroid('MULTIPOINT ( -1 0, -1 2, -1 3, -1 4, -1 7, 0 1, 0 3, 1 1, 2 0, 6 0, 7 8, 9 8, 10 6 )'));
          st_astext
-----
POINT(2.30769230769231 3.30769230769231)
(1 row)

SELECT ST_AsText(ST_centroid(g))
FROM   ST_GeomFromText ('CIRCULARSTRING(0 2, -1 1,0 0, 0.5 0, 1 0, 2 1, 1 2, 0.5 2, 0 2)') AS g ;
-----
POINT(0.5 1)

SELECT ST_AsText(ST_centroid(g))
FROM   ST_GeomFromText ('COMPOUNDCURVE(CIRCULARSTRING(0 2, -1 1,0 0),(0 0, 0.5 0, 1 0), CIRCULARSTRING( 1 0, 2 1, 1 2),(1 2, 0.5 2, 0 2))' ) AS g;
-----
POINT(0.5 1)
```

**Siehe auch**

[ST\\_PointOnSurface](#), [ST\\_GeometricMedian](#)

**7.14.4 ST\_ChaikinSmoothing**

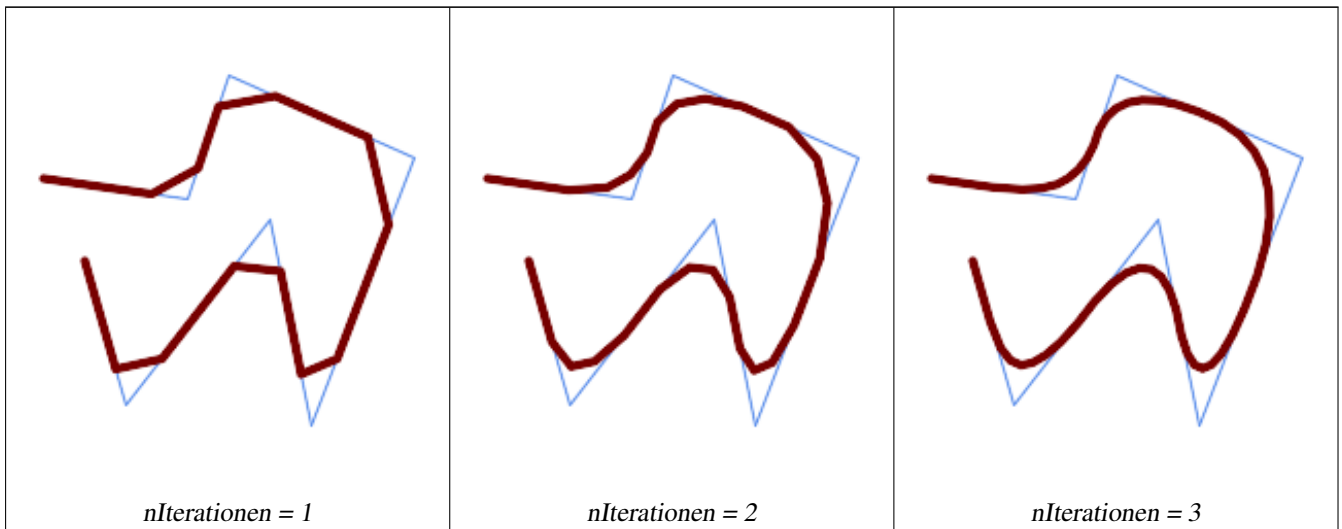
`ST_ChaikinSmoothing` — Gibt eine geglättete Version einer Geometrie zurück, die den Chaikin-Algorithmus verwendet

**Synopsis**

geometry `ST_ChaikinSmoothing`(geometry geom, integer nIterations = 1, boolean preserveEndpoints = false);



Glätten eines LineStrings mit 1, 2 und 3 Iterationen:



```
SELECT ST_ChaikinSmoothing(
  'LINESTRING (10 140, 80 130, 100 190, 190 150, 140 20, 120 120, 50 30, 30 100) ←
  ',
  generate_series(1, 3) );
```

#### Siehe auch

[ST\\_Simplify](#), [ST\\_SimplifyPreserveTopology](#), [ST\\_SimplifyVW](#)

### 7.14.5 ST\_ConcaveHull

`ST_ConcaveHull` — Berechnet eine möglicherweise konkave Geometrie, die alle Eckpunkte der Eingabegeometrie enthält

#### Synopsis

geometry **ST\_ConcaveHull**(geometry param\_geom, float param\_pctconvex, boolean param\_allow\_holes = false);

#### Beschreibung

Eine konkave Hülle ist eine (normalerweise) konkave Geometrie, die die Eingabe enthält und deren Scheitelpunkte eine Teilmenge der Eingabepunkte sind. Im allgemeinen Fall ist die konkave Hülle ein Polygon. Die konkave Hülle von zwei oder mehr kollinearen Punkten ist ein Zwei-Punkt-LineString. Die konkave Hülle von einem oder mehreren identischen Punkten ist ein Punkt. Das Polygon enthält keine Löcher, es sei denn, das optionale Argument `param_allow_holes` wird als `true` angegeben.

Man kann sich eine konkave Hülle wie eine "Schrumpfverpackung" für eine Reihe von Punkten vorstellen. Dies unterscheidet sich von der **konvexen Hülle**, die eher wie ein Gummiband um die Punkte gewickelt ist. Eine konkave Hülle hat im Allgemeinen eine kleinere Fläche und stellt eine natürlichere Begrenzung für die Eingabepunkte dar.

Die `param_pctconvex` steuert die Konkavität der berechneten Hülle. Ein Wert von 1 ergibt die konvexe Hülle. Werte zwischen 1 und 0 erzeugen Hüllen mit zunehmender Konkavität. Ein Wert von 0 ergibt eine Hülle mit maximaler Konkavität (aber immer noch ein einzelnes Polygon). Die Wahl eines geeigneten Wertes hängt von der Art der Eingabedaten ab, aber häufig führen Werte zwischen 0,3 und 0,1 zu vernünftigen Ergebnissen.

**Note**

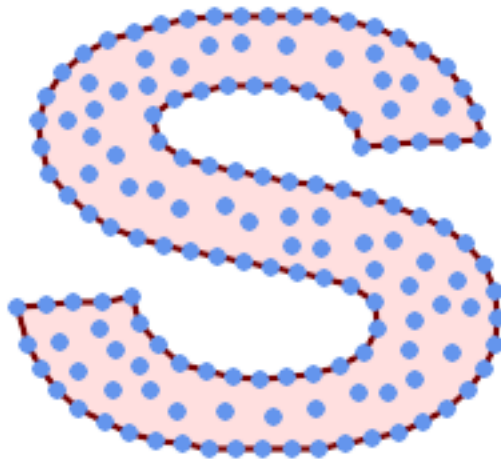
Technisch gesehen bestimmt die `param_pctconvex` eine Länge als Bruchteil der Differenz zwischen der längsten und der kürzesten Kante in der Delaunay-Triangulation der Eingabepunkte. Kanten, die länger als diese Länge sind, werden aus der Triangulation "abgetragen". Die verbleibenden Dreiecke bilden die konkave Hülle.

Bei punkt- und linienförmigen Eingaben umschließt die Hülle alle Punkte der Eingaben. Bei polygonalen Eingaben umschließt die Hülle alle Punkte der Eingabe *und auch* alle von der Eingabe abgedeckten Flächen. Wenn Sie eine punktweise Hülle einer polygonalen Eingabe wünschen, konvertieren Sie sie zunächst mit `ST_Points` in Punkte.

Es handelt sich nicht um eine Aggregatfunktion. Um die konkave Hülle einer Reihe von Geometrien zu berechnen, verwenden Sie `ST_Collect` (z. B. `ST_ConcaveHull( ST_Collect( geom ), 0.80)`).

Verfügbarkeit: 2.0.0

Verbessert: 3.3.0, native GEOS-Implementierung aktiviert für GEOS 3.11+

**Beispiele**

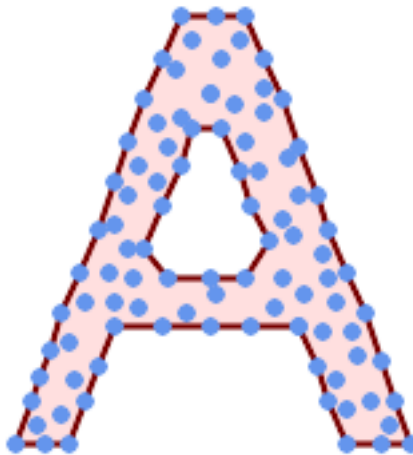
*Konvexe Hülle eines MultiLineString und eines MultiPoint*

```
SELECT ST_AsText( ST_ConcaveHull(
  'MULTIPOINT ((10 72), (53 76), (56 66), (63 58), (71 51), (81 48), (91 46), (101 ←
    45), (111 46), (121 47), (131 50), (140 55), (145 64), (144 74), (135 80), (125 ←
    83), (115 85), (105 87), (95 89), (85 91), (75 93), (65 95), (55 98), (45 102), ←
    (37 107), (29 114), (22 122), (19 132), (18 142), (21 151), (27 160), (35 167), ←
    (44 172), (54 175), (64 178), (74 180), (84 181), (94 181), (104 181), (114 181) ←
    , (124 181), (134 179), (144 177), (153 173), (162 168), (171 162), (177 154), ←
    (182 145), (184 135), (139 132), (136 142), (128 149), (119 153), (109 155), (99 ←
    155), (89 155), (79 153), (69 150), (61 144), (63 134), (72 128), (82 125), (92 ←
    123), (102 121), (112 119), (122 118), (132 116), (142 113), (151 110), (161 ←
    106), (170 102), (178 96), (185 88), (189 78), (190 68), (189 58), (185 49), ←
    (179 41), (171 34), (162 29), (153 25), (143 23), (133 21), (123 19), (113 19), ←
    (102 19), (92 19), (82 19), (72 21), (62 22), (52 25), (43 29), (33 34), (25 41) ←
    , (19 49), (14 58), (21 73), (31 74), (42 74), (173 134), (161 134), (150 133), ←
    (97 104), (52 117), (157 156), (94 171), (112 106), (169 73), (58 165), (149 40) ←
    , (70 33), (147 157), (48 153), (140 96), (47 129), (173 55), (144 86), (159 67) ←
    , (150 146), (38 136), (111 170), (124 94), (26 59), (60 41), (71 162), (41 64), ←
    (88 110), (122 34), (151 97), (157 56), (39 146), (88 33), (159 45), (47 56), ←
    (138 40), (129 165), (33 48), (106 31), (169 147), (37 122), (71 109), (163 89), ←
    (37 156), (82 170), (180 72), (29 142), (46 41), (59 155), (124 106), (157 80), ←
    (175 82), (56 50), (62 116), (113 95), (144 167))',
```

```

    0.1 ) );
---st_astext---
POLYGON ((18 142, 21 151, 27 160, 35 167, 44 172, 54 175, 64 178, 74 180, 84 181, 94 181, ←
  104 181, 114 181, 124 181, 134 179, 144 177, 153 173, 162 168, 171 162, 177 154, 182 ←
  145, 184 135, 173 134, 161 134, 150 133, 139 132, 136 142, 128 149, 119 153, 109 155, 99 ←
  155, 89 155, 79 153, 69 150, 61 144, 63 134, 72 128, 82 125, 92 123, 102 121, 112 119, ←
  122 118, 132 116, 142 113, 151 110, 161 106, 170 102, 178 96, 185 88, 189 78, 190 68, ←
  189 58, 185 49, 179 41, 171 34, 162 29, 153 25, 143 23, 133 21, 123 19, 113 19, 102 19, ←
  92 19, 82 19, 72 21, 62 22, 52 25, 43 29, 33 34, 25 41, 19 49, 14 58, 10 72, 21 73, 31 ←
  74, 42 74, 53 76, 56 66, 63 58, 71 51, 81 48, 91 46, 101 45, 111 46, 121 47, 131 50, 140 ←
  55, 145 64, 144 74, 135 80, 125 83, 115 85, 105 87, 95 89, 85 91, 75 93, 65 95, 55 98, ←
  45 102, 37 107, 29 114, 22 122, 19 132, 18 142))

```

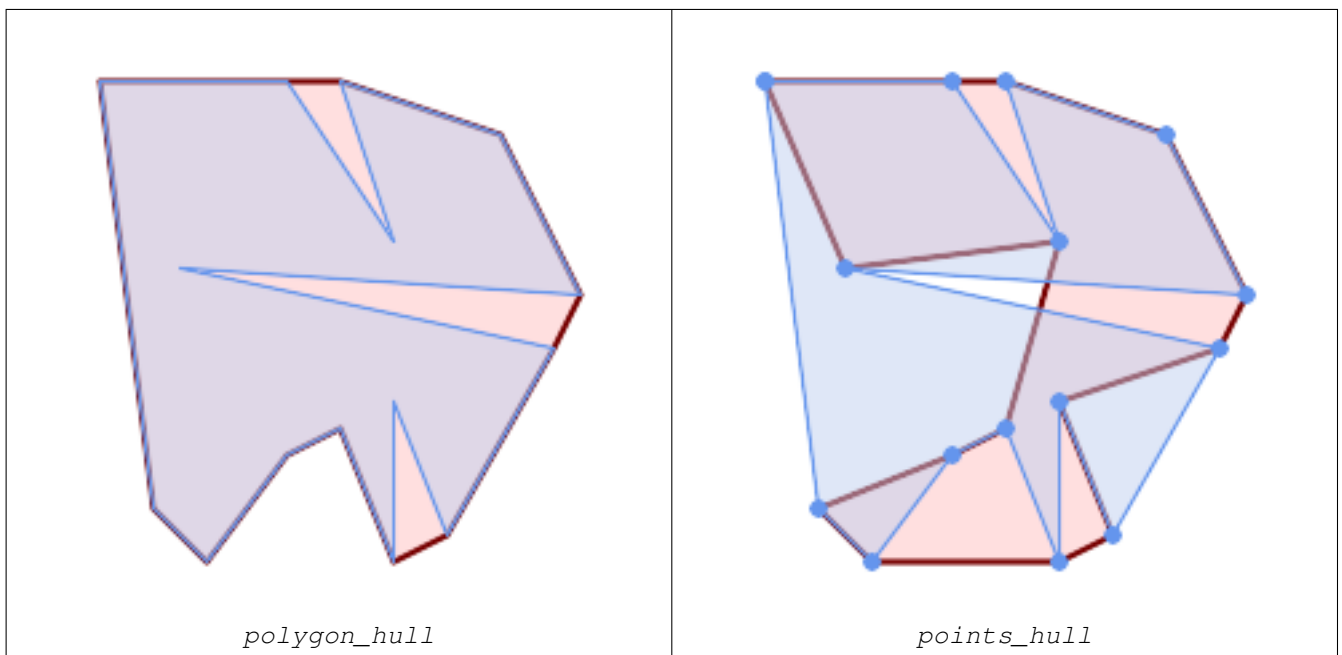


*Konvexe Hülle eines MultiLineString und eines MultiPoint*

```

SELECT ST_AsText( ST_ConcaveHull(
  'MULTIPOINT ((132 64), (114 64), (99 64), (81 64), (63 64), (57 49), (52 36), (46 ←
    20), (37 20), (26 20), (32 36), (39 55), (43 69), (50 84), (57 100), (63 118), ←
    (68 133), (74 149), (81 164), (88 180), (101 180), (112 180), (119 164), (126 ←
    149), (132 131), (139 113), (143 100), (150 84), (157 69), (163 51), (168 36), ←
    (174 20), (163 20), (150 20), (143 36), (139 49), (132 64), (99 151), (92 138), ←
    (88 124), (81 109), (74 93), (70 82), (83 82), (99 82), (112 82), (126 82), (121 ←
    96), (114 109), (110 122), (103 138), (99 151), (34 27), (43 31), (48 44), (46 ←
    58), (52 73), (63 73), (61 84), (72 71), (90 69), (101 76), (123 71), (141 62), ←
    (166 27), (150 33), (159 36), (146 44), (154 53), (152 62), (146 73), (134 76), ←
    (143 82), (141 91), (130 98), (126 104), (132 113), (128 127), (117 122), (112 ←
    133), (119 144), (108 147), (119 153), (110 171), (103 164), (92 171), (86 160), ←
    (88 142), (79 140), (72 124), (83 131), (79 118), (68 113), (63 102), (68 93), ←
    (35 45))',
  0.15, true ) );
---st_astext---
POLYGON ((43 69, 50 84, 57 100, 63 118, 68 133, 74 149, 81 164, 88 180, 101 180, 112 180, ←
  119 164, 126 149, 132 131, 139 113, 143 100, 150 84, 157 69, 163 51, 168 36, 174 20, 163 ←
  20, 150 20, 143 36, 139 49, 132 64, 114 64, 99 64, 81 64, 63 64, 57 49, 52 36, 46 20, ←
  37 20, 26 20, 32 36, 35 45, 39 55, 43 69), (88 124, 81 109, 74 93, 83 82, 99 82, 112 82, ←
  121 96, 114 109, 110 122, 103 138, 92 138, 88 124))

```



Vergleich einer konkaven Hülle eines Polygons mit der konkaven Hülle der Punkte, aus denen es besteht. Die Hülle respektiert die Begrenzung des Polygons, während die punktbasierte Hülle dies nicht tut.

```
WITH data(geom) AS (VALUES
  ('POLYGON ((10 90, 39 85, 61 79, 50 90, 80 80, 95 55, 25 60, 90 45, 70 16, 63 38, 60 10, ←
    50 30, 43 27, 30 10, 20 20, 10 90))'::geometry)
)
SELECT  ST_ConcaveHull( geom, 0.1) AS polygon_hull,
        ST_ConcaveHull( ST_Points(geom), 0.1) AS points_hull
FROM data;
```

Verwendung mit `ST_Collect` zur Berechnung der konvexen Hüllen einer Geometrie.

```
-- Compute estimate of infected area based on point observations
SELECT disease_type,
       ST_ConcaveHull( ST_Collect(obs_pnt), 0.3 ) AS geom
FROM disease_obs
GROUP BY disease_type;
```

### Siehe auch

[ST\\_ConvexHull](#), [ST\\_Collect](#), [ST\\_AlphaShape](#), [ST\\_OptimalAlphaShape](#)

## 7.14.6 ST\_ConvexHull

`ST_ConvexHull` — Berechnet die konvexe Hülle einer Geometrie.

### Synopsis

geometry `ST_ConvexHull`(geometry geomA);

## Beschreibung

Berechnet die konvexe Hülle einer Geometrie. Die konvexe Hülle stellt die kleinste konvexe Geometrie dar, welche die gesamte Geometrie einschließt.

Man kann sich die konvexe Hülle als die Geometrie vorstellen, die man erhält, wenn man ein Gummiband um eine Reihe von Geometrien wickelt. Dies unterscheidet sich von einer **konkaven Hülle**, die dem "Einschweißen" der Geometrien entspricht. Eine konvexe Hülle wird häufig verwendet, um ein betroffenes Gebiet auf der Grundlage einer Reihe von Punktbeobachtungen zu bestimmen.

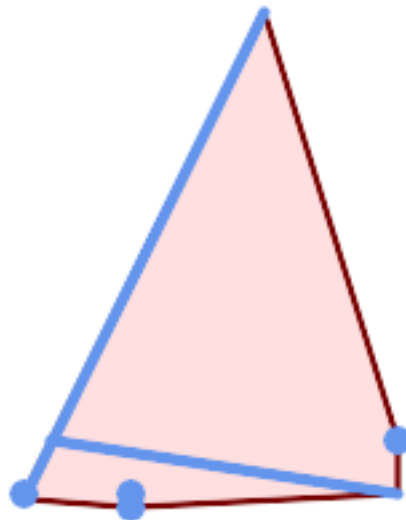
Im allgemeinen Fall ist die konvexe Hülle ein Polygon. Die konvexe Hülle von zwei oder mehr kollinearen Punkten ist ein Zweipunkt-Linienzug (LineString). Die konvexe Hülle von einem oder mehreren identischen Punkten ist ein Punkt.

Dies ist keine Aggregatfunktion. Um die konvexe Hülle einer Menge von Geometrien zu berechnen, verwenden Sie **ST\_Collect**, um sie zu einer Geometriesammlung zu aggregieren (z. B. `ST_ConvexHull(ST_Collect(geom))`).

Wird durch das GEOS Modul ausgeführt

- ✓ Diese Methode implementiert die [OGC Simple Features Implementation Specification for SQL 1.1](#). s2.1.1.3
- ✓ Diese Methode setzt die SQL/MM-Spezifikation um. SQL-MM IEC 13249-3: 5.1.16
- ✓ Diese Funktion unterstützt 3d und lässt den Z-Index nicht fallen.

## Beispiele



*Konvexe Hülle eines MultiLineString und eines MultiPoint*

```
SELECT ST_AsText(ST_ConvexHull(
  ST_Collect(
    ST_GeomFromText('MULTILINESTRING((100 190,10 8),(150 10, 20 30)'),
    ST_GeomFromText('MULTIPOINT(50 5, 150 30, 50 10, 10 10)')
  )));
---st_astext---
POLYGON((50 5,10 8,10 10,100 190,150 30,150 10,50 5))
```

Verwendung mit **ST\_Collect** zur Berechnung der konvexen Hüllen einer Geometrie.

```
--Get estimate of infected area based on point observations
SELECT d.disease_type,
  ST_ConvexHull(ST_Collect(d.geom)) As geom
FROM disease_obs As d
GROUP BY d.disease_type;
```

**Siehe auch**

[ST\\_Collect](#), [ST\\_ConcaveHull](#), [ST\\_MinimumBoundingCircle](#)

**7.14.7 ST\_DelaunayTriangles**

ST\_DelaunayTriangles — Gibt die Delaunay-Triangulation der Scheitelpunkte einer Geometrie zurück.

**Synopsis**

```
geometry ST_DelaunayTriangles(geometry g1, float tolerance = 0.0, int4 flags = 0);
```

**Beschreibung**

Berechnet die **Delaunay-Triangulation** der Scheitelpunkte der Eingabegeometrie. Die optionale `Toleranz` kann verwendet werden, um nahegelegene Eingabepunkte zusammenzufassen, was in einigen Situationen die Robustheit verbessert. Die Ergebnisgeometrie wird durch die konvexe Hülle der Eingabescheitelpunkte begrenzt. Die Darstellung der Ergebnisgeometrie wird durch den Code `flags` bestimmt:

- 0 - eine GEOMETRIESAMMLUNG von dreieckigen POLYGONEN (Standard)
- 1 - ein MULTILINESTRING der Kanten der Triangulation
- 2 - Eine TIN der Triangulation

Wird vom GEOS Modul ausgeführt

Verfügbarkeit: 2.1.0



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

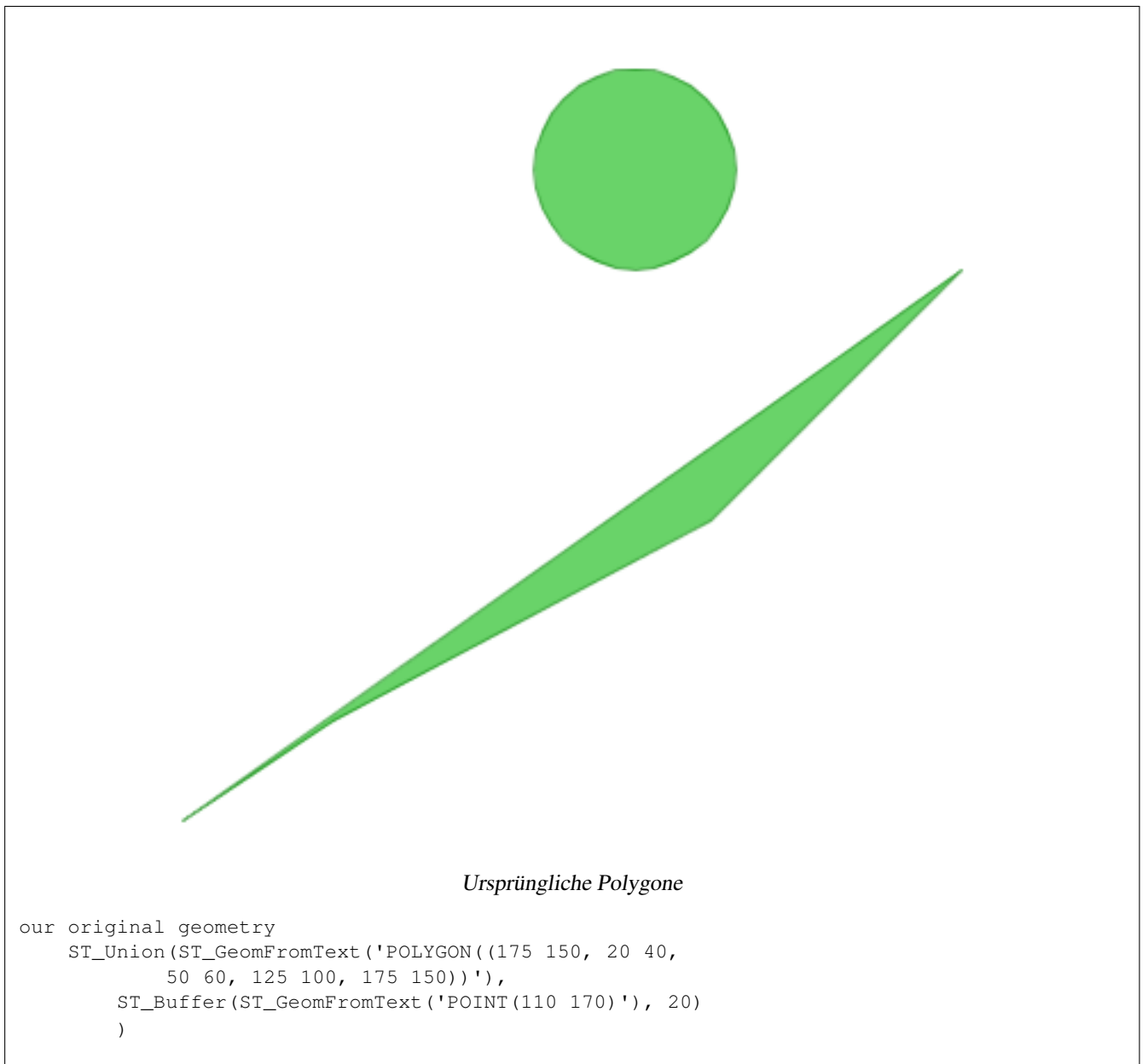


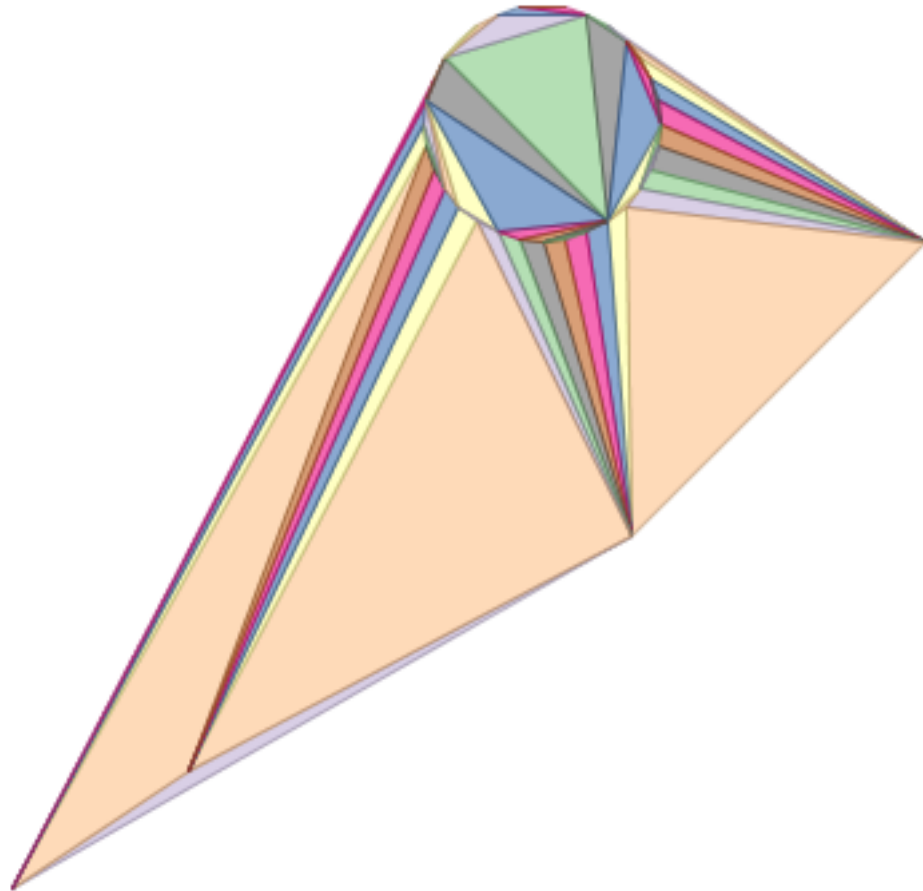
Diese Funktion unterstützt Dreiecke und dreieckige unregelmäßige Netzoberflächen (TIN).

**Beispiele**

---



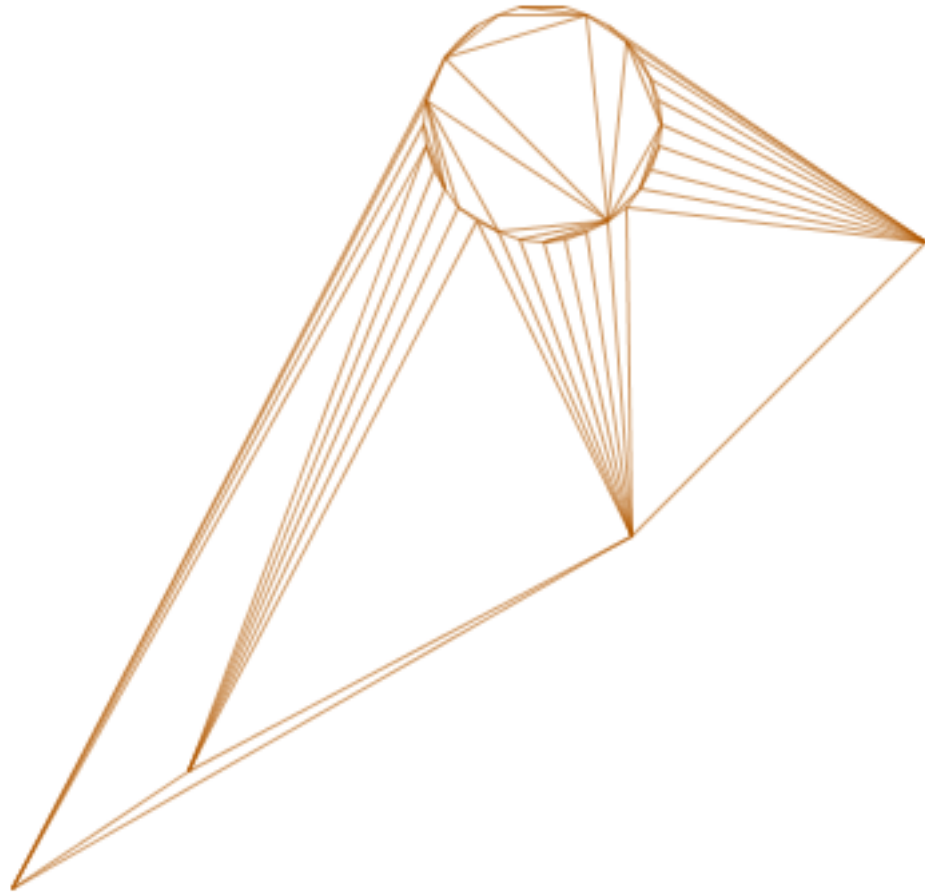




*ST\_DelaunayTriangles von 2 Polygonen: delaunay triangulierte Polygone, jedes der Dreiecke ist in einer eigenen Farbe dargestellt*

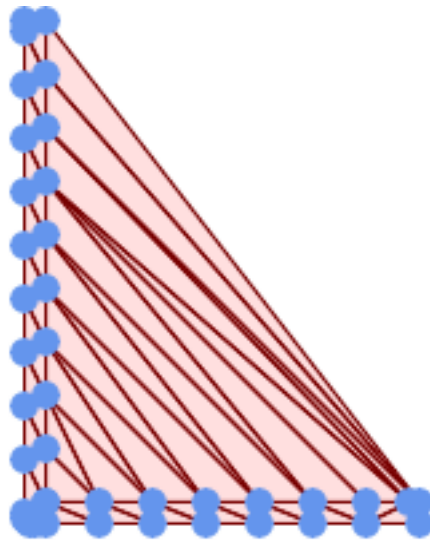
geometries overlaid multilinestring triangles

```
SELECT
  ST_DelaunayTriangles(
    ST_Union(ST_GeomFromText('POLYGON((175 150, 20 40,
      50 60, 125 100, 175 150))'),
    ST_Buffer(ST_GeomFromText('POINT(110 170)'), 20)
  ))
As dtriag;
```



*-- Delaunay-Dreiecke als MultiLinestring*

```
SELECT
  ST_DelaunayTriangles(
    ST_Union(ST_GeomFromText('POLYGON((175 150, 20 40,
      50 60, 125 100, 175 150))'),
    ST_Buffer(ST_GeomFromText('POINT(110 170)'), 20)
  ),0.001,1)
As dtriag;
```



-- Delaunay Dreiecke von 45 Punkten als 55 Dreieckspolygone

this produces a table of 42 points that form an L shape

```
SELECT (ST_DumpPoints(ST_GeomFromText (
'MULTIPOINT(14 14,34 14,54 14,74 14,94 14,114 14,134 14,
150 14,154 14,154 6,134 6,114 6,94 6,74 6,54 6,34 6,
14 6,10 6,8 6,7 7,6 8,6 10,6 30,6 50,6 70,6 90,6 110,6 130,
6 150,6 170,6 190,6 194,14 194,14 174,14 154,14 134,14 114,
14 94,14 74,14 54,14 34,14 14)'))).geom
    INTO TABLE l_shape;
```

output as individual polygon triangles

```
SELECT ST_AsText((ST_Dump(geom)).geom) As wkt
FROM ( SELECT ST_DelaunayTriangles(ST_Collect(geom)) As geom
FROM l_shape) As foo;
```

wkt

```
POLYGON((6 194,6 190,14 194,6 194))
POLYGON((14 194,6 190,14 174,14 194))
POLYGON((14 194,14 174,154 14,14 194))
POLYGON((154 14,14 174,14 154,154 14))
POLYGON((154 14,14 154,150 14,154 14))
POLYGON((154 14,150 14,154 6,154 14))
```

### Beispiel mit Scheitelpunkten mit Z-Werten.

3D multipoint

```
SELECT ST_AsText(ST_DelaunayTriangles(ST_GeomFromText (
'MULTIPOINT Z(14 14 10, 150 14 100,34 6 25, 20 10 150)')) As wkt;
```

wkt

```
GEOMETRYCOLLECTION Z (POLYGON Z ((14 14 10,20 10 150,34 6 25,14 14 10))
,POLYGON Z ((14 14 10,34 6 25,150 14 100,14 14 10))
```

**Siehe auch**

[ST\\_VoronoiPolygons](#), [ST\\_TriangulatePolygon](#), [ST\\_ConstrainedDelaunayTriangles](#), [ST\\_VoronoiLines](#), [ST\\_ConvexHull](#)

**7.14.8 ST\_FilterByM**

ST\_FilterByM — Entfernt Scheitelpunkte basierend auf ihrem M-Wert

**Synopsis**

geometry **ST\_FilterByM**(geometry geom, double precision min, double precision max = null, boolean returnM = false);

**Beschreibung**

Filtert Scheitelpunkte auf der Grundlage ihres M-Werts heraus. Gibt eine Geometrie zurück, die nur Scheitelpunkte enthält, deren M-Wert größer oder gleich dem Min-Wert und kleiner oder gleich dem Max-Wert ist. Wenn das Argument max-value weggelassen wird, wird nur der min-Wert berücksichtigt. Wenn das vierte Argument weggelassen wird, wird der M-Wert nicht in der resultierenden Geometrie enthalten sein. Wenn die resultierende Geometrie zu wenig Scheitelpunkte für ihren Geometrietyp hat, wird eine leere Geometrie zurückgegeben. In einer Geometriesammlung werden Geometrien, die nicht genügend Punkte haben, einfach stillschweigend ausgelassen.

Diese Funktion ist hauptsächlich für die Verwendung in Verbindung mit ST\_SetEffectiveArea gedacht. ST\_EffectiveArea legt die effektive Fläche eines Scheitelpunkts in seinem m-Wert fest. Mit ST\_FilterByM ist es dann möglich, eine vereinfachte Version der Geometrie ohne jegliche Berechnungen zu erhalten, einfach durch Filtern

**Note**

Es gibt einen Unterschied zwischen ST\_SimplifyVW und ST\_FilterByM, was ST\_SimplifyVW zurückgibt, wenn nicht genügend Punkte die Kriterien erfüllen. ST\_SimplifyVW gibt die Geometrie mit genügend Punkten zurück, während ST\_FilterByM eine leere Geometrie zurückgibt.

**Note**

Beachten Sie, dass die zurückgegebene Geometrie ungültig sein kann

**Note**

Diese Funktion liefert alle Dimensionen, einschließlich der Z- und M-Werte

Verfügbarkeit: 2.5.0

**Beispiele****Eine gefilterte Linie**

```
SELECT ST_AsText(ST_FilterByM(geom,30)) simplified
FROM (SELECT ST_SetEffectiveArea('LINESTRING(5 2, 3 8, 6 20, 7 25, 10 10)::geometry) geom ←
) As foo;
```

result

```
simplified
```

```
-----
LINESTRING(5 2,7 25,10 10)
```

**Siehe auch**

[ST\\_SetEffectiveArea](#), [ST\\_SimplifyVW](#)

**7.14.9 ST\_GeneratePoints**

`ST_GeneratePoints` — Erzeugt einen Multipunkt aus zufälligen Punkten, die in einem Polygon oder MultiPolygon enthalten sind.

**Synopsis**

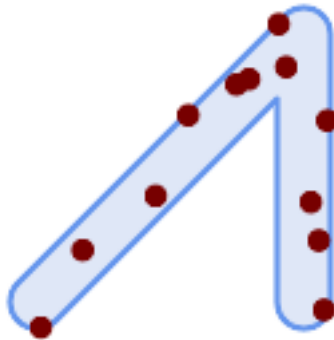
```
geometry ST_GeneratePoints(geometry g, integer npoints, integer seed = 0);
```

**Beschreibung**

`ST_GeneratePoints` erzeugt einen Multipunkt, der aus einer bestimmten Anzahl von Pseudo-Zufallspunkten besteht, die innerhalb des Eingabebereichs liegen. Der optionale Wert `seed` wird verwendet, um eine deterministische Folge von Punkten zu erzeugen, und muss größer als Null sein.

Verfügbarkeit: 2.3.0

Erweiterung: mit 3.0.0 wurde das Argument "seed" hinzugefügt

**Beispiele**

*Erzeugt einen Multipunkt, der aus 12 Punkten besteht, die über das ursprüngliche Polygon gelegt werden, unter Verwendung eines zufälligen Startwertes 1996*

```
SELECT ST_GeneratePoints(geom, 12, 1996)
FROM (
  SELECT ST_Buffer(
    ST_GeomFromText(
      'LINESTRING(50 50,150 150,150 50)'),
    10, 'endcap=round join=round') AS geom
) AS s;
```

Bei einer Tabelle mit Polygonen `s` sind 12 einzelne Punkte pro Polygon zu ermitteln. Die Ergebnisse sind bei jedem Durchlauf anders.

```
SELECT s.id, dp.path[1] AS pt_id, dp.geom
FROM s, ST_DumpPoints(ST_GeneratePoints(s.geom,12)) AS dp;
```

## Siehe auch

[ST\\_DumpPoints](#)

### 7.14.10 ST\_GeometricMedian

`ST_GeometricMedian` — Gibt den geometrischen Median eines Mehrfachpunktes zurück.

#### Synopsis

geometry **ST\_GeometricMedian** ( geometry geom, float8 tolerance = NULL, int max\_iter = 10000, boolean fail\_if\_not\_converged = false);

#### Beschreibung

Berechnet den ungefähren geometrischen Median einer MultiPoint-Geometrie unter Verwendung des Weiszfeld-Algorithmus. Der geometrische Median ist der Punkt, der die Summe der Abstände zu den Eingabepunkten minimiert. Er liefert ein Zentralitätsmaß, das weniger empfindlich auf Ausreißerpunkte reagiert als der Schwerpunkt (Center of Mass).

Der Algorithmus iteriert, bis die Abstandsänderung zwischen aufeinanderfolgenden Iterationen kleiner ist als der angegebene `tolerance` Parameter. Wenn diese Bedingung nach `max_iterations` Iterationen nicht erfüllt ist, erzeugt die Funktion einen Fehler und wird beendet, es sei denn, `fail_if_not_converged` ist auf `false` (die Voreinstellung) gesetzt.

Wenn das Argument `tolerance` nicht angegeben wird, wird der Toleranzwert auf der Grundlage der Ausdehnung der Eingabegeometrie berechnet.

Falls vorhanden, werden die Werte der Eingabepunkte `M` als deren relative Gewichte interpretiert.

Verfügbarkeit: 2.3.0

Erweiterung: 2.5.0 Unterstützung für `M` zur Gewichtung nach Punkten.

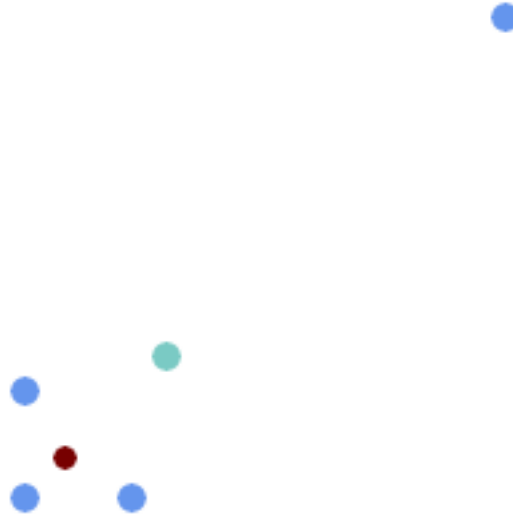


Diese Funktion unterstützt 3d und lässt den Z-Index nicht fallen.



Diese Funktion unterstützt M-Koordinaten.

## Beispiele



Vergleich des geometrischen Medians (rot) und des Schwerpunkts (türkis) eines MultiPoint.

```
WITH test AS (
SELECT 'MULTIPOINT((10 10), (10 40), (40 10), (190 190))'::geometry geom)
SELECT
  ST_AsText(ST_Centroid(geom)) centroid,
  ST_AsText(ST_GeometricMedian(geom)) median
FROM test;
```

centroid	median
POINT(62.5 62.5)	POINT(25.01778421249728 25.01778421249728)

(1 row)

## Siehe auch

[ST\\_Centroid](#)

### 7.14.11 ST\_LineMerge

ST\_LineMerge — Gibt die Linien zurück, die durch das Zusammenfügen eines MultiLineString gebildet werden.

#### Synopsis

```
geometry ST_LineMerge(geometry amultilinestring);
geometry ST_LineMerge(geometry amultilinestring, boolean directed);
```

#### Beschreibung

Gibt einen LineString oder MultiLineString zurück, der durch Zusammenfügen der Linienelemente eines MultiLineString gebildet wird. Linien werden an ihren Endpunkten an 2-Wege-Kreuzungen verbunden. Linien werden nicht über Schnittpunkte mit 3 oder mehr Richtungen verbunden.

Wenn **directed** TRUE ist, dann ändert ST\_LineMerge nicht die Reihenfolge der Punkte innerhalb von LineStrings, so dass Linien mit entgegengesetzten Richtungen nicht zusammengeführt werden



**Note**

Nur mit MultiLineString/LineStrings verwenden. Andere Geometrietypen geben eine leere GeometryCollection zurück

Wird vom GEOS Modul ausgeführt

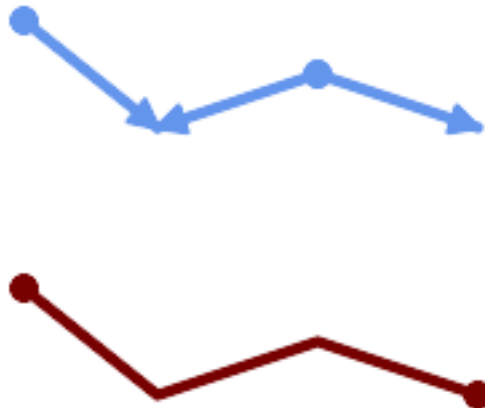
Verbessert: 3.3.0 akzeptiert einen gerichteten Parameter.

Erfordert GEOS >= 3.11.0 zur Verwendung des gerichteten Parameters.

Verfügbarkeit: 1.1.0

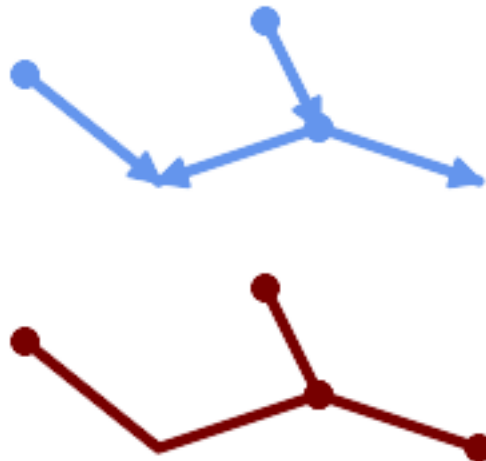
**Warning**

Diese Funktion entfernt die Dimension M.

**Beispiele**

*Zusammenführen von Linien mit unterschiedlicher Ausrichtung.*

```
SELECT ST_AsText(ST_LineMerge(  
'MULTILINESTRING((10 160, 60 120), (120 140, 60 120), (120 140, 180 120))'  
));  
-----  
LINESTRING(10 160,60 120,120 140,180 120)
```

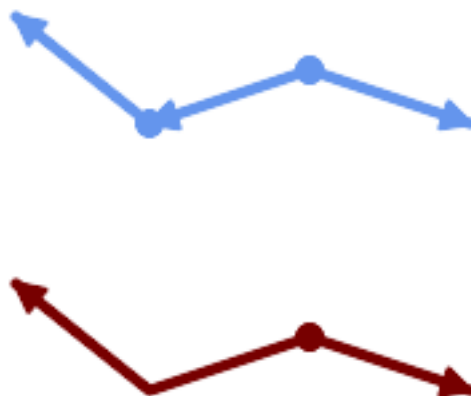


*Linien werden nicht über Schnittpunkte mit dem Grad > 2 zusammengeführt.*

```
SELECT ST_AsText(ST_LineMerge(
'MULTILINESTRING((10 160, 60 120), (120 140, 60 120), (120 140, 180 120), (100 180, 120 140))'
));
-----
MULTILINESTRING((10 160,60 120,120 140),(100 180,120 140),(120 140,180 120))
```

Ist das Zusammenführen aufgrund von sich nicht berührenden Zeilen nicht möglich, wird der ursprüngliche MultiLineString zurückgegeben.

```
SELECT ST_AsText(ST_LineMerge(
'MULTILINESTRING((-29 -27,-30 -29.7,-36 -31,-45 -33),(-45.2 -33.2,-46 -32))'
));
-----
MULTILINESTRING((-45.2 -33.2,-46 -32),(-29 -27,-30 -29.7,-36 -31,-45 -33))
```



*Linien mit entgegengesetzten Richtungen werden nicht zusammengeführt, wenn directed = TRUE.*

```
SELECT ST_AsText(ST_LineMerge(
'MULTILINESTRING((60 30, 10 70), (120 50, 60 30), (120 50, 180 30))',
```

```
TRUE));
-----
MULTILINESTRING((120 50,60 30,10 70),(120 50,180 30))
```

Beispiel für die Handhabung der Z-Dimension.

```
SELECT ST_AsText(ST_LineMerge(
  'MULTILINESTRING((-29 -27 11,-30 -29.7 10,-36 -31 5,-45 -33 6), (-29 -27 12,-30 -29.7 ←
    5), (-45 -33 1,-46 -32 11))'
));
-----
LINESTRING Z (-30 -29.7 5,-29 -27 11,-30 -29.7 10,-36 -31 5,-45 -33 1,-46 -32 11)
```

### Siehe auch

[ST\\_Segmentize](#), [ST\\_LineSubstring](#)

## 7.14.12 ST\_MaximumInscribedCircle

`ST_MaximumInscribedCircle` — Berechnet die konvexe Hülle einer Geometrie.

### Synopsis

(geometry, geometry, double precision) `ST_MaximumInscribedCircle`(geometry geom);

### Beschreibung

Findet den größten Kreis, der in einem (Mehr-)Polygon enthalten ist oder der keine Linien und Punkte überschneidet. Gibt einen Datensatz mit Feldern zurück:

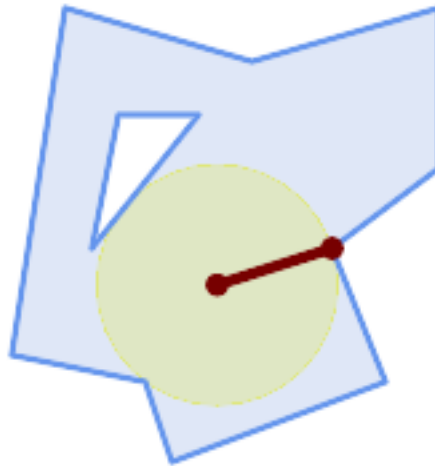
- `center` - Mittelpunkt des Kreises
- `nächstgelegener` - ein Punkt auf der Geometrie, der dem Zentrum am nächsten liegt
- `radius` - Radius des Kreises

Bei polygonalen Eingaben wird der Kreis innerhalb der Begrenzungsringe eingeschrieben, wobei die inneren Ringe als Begrenzungen verwendet werden. Bei linearen und punktförmigen Eingaben wird der Kreis in die konvexe Hülle der Eingabe eingeschrieben, wobei die eingegebenen Linien und Punkte als weitere Begrenzungen verwendet werden.

Verfügbarkeit: 3.1.0.

Erfordert GEOS >= 3.9.0.

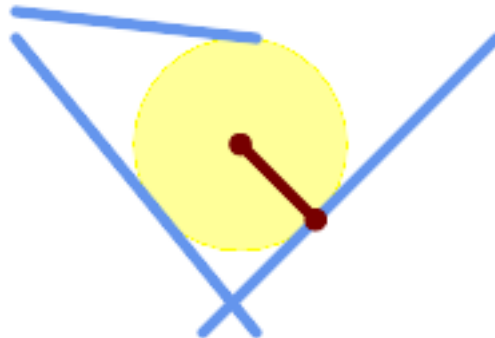
## Beispiele



*Maximaler Inkreis eines Polygons. Mittelpunkt, nächstgelegener Punkt und Radius werden zurückgegeben.*

```
SELECT radius, ST_AsText(center) AS center, ST_AsText(nearest) AS nearest
FROM ST_MaximumInscribedCircle(
  'POLYGON ((40 180, 110 160, 180 180, 180 120, 140 90, 160 40, 80 10, 70 40, 20 50, 40 180),
    (60 140, 50 90, 90 140, 60 140))');
```

radius	center	nearest
45.165845650018	POINT(96.953125 76.328125)	POINT(140 90)



*Maximaler Inkreis eines Multi-Linien-Strangs. Mittelpunkt, nächstgelegener Punkt und Radius werden zurückgegeben.*

## Siehe auch

[ST\\_MinimumBoundingRadius](#), [ST\\_LargestEmptyCircle](#)

### 7.14.13 ST\_LargestEmptyCircle

ST\_LargestEmptyCircle — Berechnet den größten Kreis, der eine Geometrie nicht überschneidet.

#### Synopsis

(geometry, geometry, double precision) **ST\_LargestEmptyCircle**(geometry geom, double precision tolerance=0.0, geometry boundary=POINT EMPTY);

#### Beschreibung

Findet den größten Kreis, der einen Satz von Punkt- und Linienhindernissen nicht überschneidet. (Polygonale Geometrien können als Hindernisse einbezogen werden, aber nur ihre Begrenzungslinien werden verwendet). Der Mittelpunkt des Kreises muss innerhalb einer polygonalen Begrenzung liegen, die standardmäßig die konvexe Hülle der Eingabegeometrie ist. Der Kreismittelpunkt ist der Punkt im Inneren der Begrenzung, der den weitesten Abstand zu den Hindernissen hat. Der Kreis selbst wird durch den Mittelpunkt und einen nächstgelegenen Punkt auf einem Hindernis gebildet, der den Radius des Kreises bestimmt.

Der Kreismittelpunkt wird mit einem iterativen Algorithmus mit einer bestimmten Genauigkeit bestimmt, die durch eine Abstandstoleranz festgelegt wird. Wenn der Genauigkeitsabstand nicht angegeben ist, wird ein angemessener Standardwert verwendet.

Gibt einen Datensatz mit Feldern zurück:

- `center` - Mittelpunkt des Kreises
- `nächstgelegener` - ein Punkt auf der Geometrie, der dem Zentrum am nächsten liegt
- `radius` - Radius des Kreises

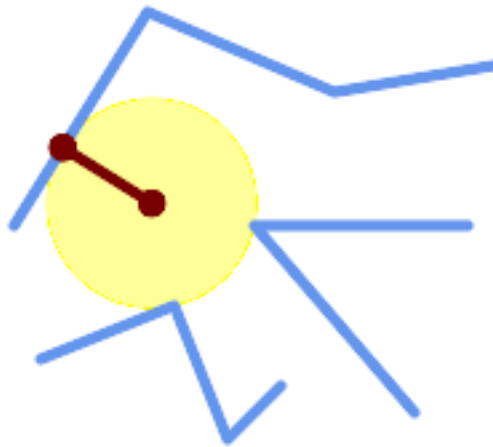
Um den größten leeren Kreis im Inneren eines Polygons zu finden, siehe [ST\\_MaximumInscribedCircle](#).

Verfügbarkeit: 3.4.0.

Erfordert GEOS >= 3.9.0.

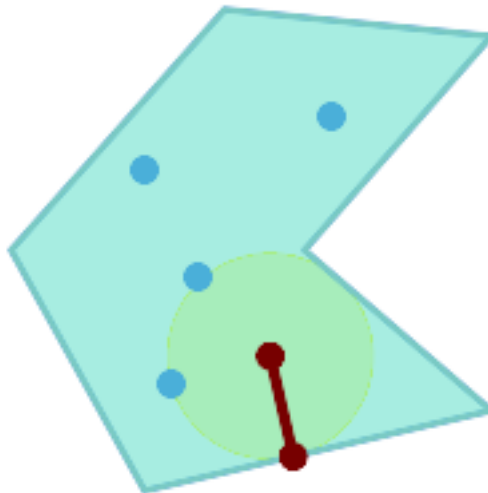
#### Beispiele

```
SELECT radius,
       center,
       nearest
FROM ST_LargestEmptyCircle(
    'MULTILINESTRING (
      (10 100, 60 180, 130 150, 190 160),
      (20 50, 70 70, 90 20, 110 40),
      (160 30, 100 100, 180 100))');
```



Größter leerer Kreis innerhalb einer Reihe von Linien.

```
SELECT radius,
       center,
       nearest
FROM ST_LargestEmptyCircle(
  ST_Collect(
    'MULTIPOINT ((70 50), (60 130), (130 150), (80 90))'::geometry,
    'POLYGON ((90 190, 10 100, 60 10, 190 40, 120 100, 190 180, 90 190))'::geometry) ←
    ,
    0,
    'POLYGON ((90 190, 10 100, 60 10, 190 40, 120 100, 190 180, 90 190))'::geometry
  );
```



Größter leerer Kreis innerhalb einer Gruppe von Punkten, der in einem Polygon liegen muss. Die Begrenzung des Polygons muss sowohl als Hindernis als auch als Beschränkung für den Kreismittelpunkt angegeben werden.

#### Siehe auch

[ST\\_MinimumBoundingRadius](#)

### 7.14.14 ST\_MinimumBoundingCircle

ST\_MinimumBoundingCircle — Gibt das kleinste Kreispolygon zurück, das eine Geometrie enthält.

#### Synopsis

```
geometry ST_MinimumBoundingCircle(geometry geomA, integer num_segs_per_qt_circ=48);
```

#### Beschreibung

Gibt das kleinste Kreispolygon zurück, das eine Geometrie enthält.



#### 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.

Verwendung mit [ST\\_Collect](#), um den minimalen Begrenzungskreis einer Reihe von Geometrien zu ermitteln.

Um zwei Punkte zu berechnen, die auf dem minimalen Kreis (dem "maximalen Durchmesser") liegen, verwenden Sie [ST\\_LongestLine](#).

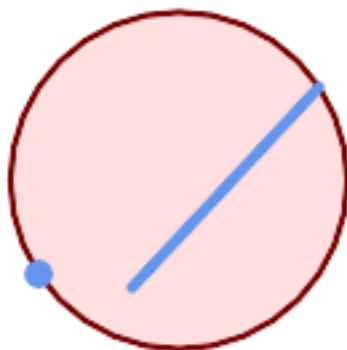
Das Verhältnis zwischen der Fläche des Polygons und der Fläche ihres kleinstmöglichen Umgebungskreises wird öfter als Roeck Test bezeichnet.

Wird vom GEOS Modul ausgeführt

Verfügbarkeit: 1.4.0

#### Beispiele

```
SELECT d.disease_type,  
       ST_MinimumBoundingCircle(ST_Collect(d.geom)) As geom  
FROM disease_obs As d  
GROUP BY d.disease_type;
```



*Minimaler Umgebungskreis eines Punktes und eines Linienzuges. Verwendet 8 Segmente um einen Viertelkreis anzunähern.*

```

SELECT ST_AsText(ST_MinimumBoundingCircle(
    ST_Collect(
        ST_GeomFromText('LINESTRING(55 75,125 150)'),
        ST_Point(20, 80)), 8
    )) As wktmbc;

wktmbc
-----
POLYGON((135.59714732062 115,134.384753327498 102.690357210921,130.79416296937 ↔
 90.8537670908995,124.963360620072 79.9451031602111,117.116420743937 ↔
 70.3835792560632,107.554896839789 62.5366393799277,96.6462329091006 ↔
 56.70583703063,84.8096427890789 53.115246672502,72.5000000000001 ↔
 51.9028526793802,60.1903572109213 53.1152466725019,48.3537670908996 ↔
 56.7058370306299,37.4451031602112 62.5366393799276,27.8835792560632 ↔
 70.383579256063,20.0366393799278 79.9451031602109,14.20583703063 ↔
 90.8537670908993,10.615246672502 102.690357210921,9.40285267938019 115,10.6152466725019 ↔
 127.309642789079,14.2058370306299 139.1462329091,20.0366393799275 ↔
 150.054896839789,27.883579256063 159.616420743937,
 37.4451031602108 167.463360620072,48.3537670908992 173.29416296937,60.190357210921 ↔
 176.884753327498,
 72.4999999999998 178.09714732062,84.8096427890786 176.884753327498,96.6462329091003 ↔
 173.29416296937,107.554896839789 167.463360620072,
 117.116420743937 159.616420743937,124.963360620072 150.054896839789,130.79416296937 ↔
 139.146232909101,134.384753327498 127.309642789079,135.59714732062 115))

```

**Siehe auch**

[ST\\_Collect](#), [ST\\_MinimumBoundingRadius](#), [ST\\_LargestEmptyCircle](#), [ST\\_LongestLine](#)

**7.14.15 ST\_MinimumBoundingRadius**

`ST_MinimumBoundingRadius` — Gibt den Mittelpunkt und den Radius des kleinsten Kreises zurück, der eine Geometrie enthält.

**Synopsis**

(geometry, double precision) `ST_MinimumBoundingRadius`(geometry geom);

**Beschreibung**

Berechnet den Mittelpunkt und den Radius des kleinsten Kreises, der eine Geometrie enthält. Gibt einen Datensatz mit Feldern zurück:

- center - Mittelpunkt des Kreises
- radius - Radius des Kreises

Verwendung mit [ST\\_Collect](#), um den minimalen Begrenzungskreis einer Reihe von Geometrien zu ermitteln.

Um zwei Punkte zu berechnen, die auf dem minimalen Kreis (dem "maximalen Durchmesser") liegen, verwenden Sie [ST\\_LongestLine](#).

Verfügbarkeit: 2.3.0



**Beispiele**

```
SELECT ST_AsText(center), radius FROM ST_MinimumBoundingRadius('POLYGON((26426 65078,26531 ←
  65242,26075 65136,26096 65427,26426 65078))');
```

st_astext	radius
POINT(26284.8418027133 65267.1145090825)	247.436045591407

**Siehe auch**

[ST\\_Collect](#), [ST\\_MinimumBoundingCircle](#), [ST\\_LongestLine](#)

**7.14.16 ST\_OrientedEnvelope**

`ST_OrientedEnvelope` — Gibt ein Rechteck mit minimalem Flächeninhalt zurück, das eine Geometrie enthält.

**Synopsis**

```
geometry ST_OrientedEnvelope( geometry geom );
```

**Beschreibung**

Gibt das gedrehte Rechteck mit dem kleinsten Flächeninhalt zurück, das eine Geometrie umschließt. Beachten Sie, dass es mehr als ein solches Rechteck geben kann. Kann im Fall von degenerierten Eingaben einen Punkt oder einen LineString zurückgeben.

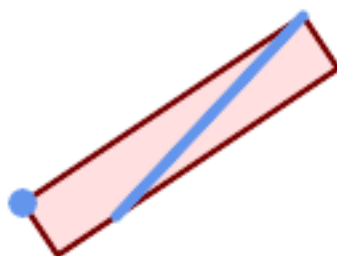
Verfügbarkeit: 2.5.0.

Erfordert GEOS >= 3.6.0.

**Beispiele**

```
SELECT ST_AsText(ST_OrientedEnvelope('MULTIPOINT ((0 0), (-1 -1), (3 2))'));)
```

st_astext
POLYGON((3 2,2.88 2.16,-1.12 -0.84,-1 -1,3 2))



*Die ausgerichtete Einhüllende eines Punktes und einer Linie.*

```
SELECT ST_AsText(ST_OrientedEnvelope(
  ST_Collect(
    ST_GeomFromText('LINESTRING(55 75,125 150)'),
    ST_Point(20, 80))
  )) As wktenv;

wktenv
-----
POLYGON((19.9999999999997 79.9999999999999,33.0769230769229 ↵
  60.3846153846152,138.076923076924 130.384615384616,125.000000000001 ↵
  150.000000000001,19.9999999999997 79.9999999999999))
```

## Siehe auch

[ST\\_Envelope](#) [ST\\_MinimumBoundingCircle](#)

### 7.14.17 ST\_OffsetCurve

**ST\_OffsetCurve** — Gibt eine versetzte Linie in einem bestimmten Abstand und einer bestimmten Seite von einer Eingabelinie zurück.

#### Synopsis

geometry **ST\_OffsetCurve**(geometry line, float signed\_distance, text style\_parameters=');

#### Beschreibung

Gibt eine versetzte Linie in einem bestimmten Abstand und einer bestimmten Seite von einer Eingabelinie zurück. Alle Punkte der zurückgegebenen Geometrien sind nicht weiter als der angegebene Abstand von der Eingabegeometrie entfernt. Nützlich für die Berechnung von parallelen Linien um eine Mittellinie.

Bei einem positiven Abstand befindet sich der Versatz auf der linken Seite der Eingabezeile und behält die gleiche Richtung bei. Bei einem negativen Abstand befindet er sich auf der rechten Seite und in der entgegengesetzten Richtung.

Die Einheiten der Entfernung werden in den Einheiten des Koordinatenreferenzsystems gemessen.

Bei einer Puzzlestück-förmigen Geometrie kann die Ausgabe manchmal ein MULTILINESTRING oder EMPTY sein.

Der optionale dritte Parameter ermöglicht es eine Liste von leerzeichengetrennten key=value Paaren anzulegen, um die Berechnungen wie folgt zu optimieren:

- 'quad\_segs=#' : Anzahl der Segmente die verwendet werden um einen Viertelkreis anzunähern (standardmäßig 8).
- 'join=round|mitre|bevel' : join style (defaults to "round"). 'miter' kann auch als Synonym für 'mitre' verwendet werden.
- 'mitre\_limit=#.#' : Gehrungsobergrenze (beeinflusst nur Gehrungsverbindungen). 'miter\_limit' kann auch als Synonym von 'mitre\_limit' verwendet werden.

Wird vom GEOS Modul ausgeführt

Behavior changed in GEOS 3.11 so offset curves now have the same direction as the input line, for both positive and negative offsets.

Verfügbarkeit: 2.0

Erweiterung: ab 2.5 wird auch GEOMETRYCOLLECTION und MULTILINESTRING unterstützt.



#### Note

Diese Funktion ignoriert die Z-Dimension. Sie liefert immer ein 2D-Ergebnis, auch wenn sie auf eine 3D-Geometrie angewendet wird.

## Beispiele

### Einen offenen Puffer um die Straßen rechnen

```
SELECT ST_Union(
  ST_OffsetCurve(f.geom, f.width/2, 'quad_segs=4 join=round'),
  ST_OffsetCurve(f.geom, -f.width/2, 'quad_segs=4 join=round')
) as track
FROM someroadstable;
```



15, 'quad\_segs=4 join=round' Ausgangslinie und die um 15 Einheiten versetzte Parallele.

```
SELECT ST_AsText(ST_OffsetCurve(↵
  ST_GeomFromText(↵
'LINESTRING(164 16,144 16,124 16,104 ↵
  16,84 16,64 16,↵
  44 16,24 16,20 16,18 16,17 17,↵
  16 18,16 20,16 40,16 60,16 80,16 100,↵
  16 120,16 140,16 160,16 180,16 195)') ↵
,↵
  15, 'quad_segs=4 join=round'));
```

output

```
LINESTRING(164 1,18 1,12.2597485145237 ↵
  2.1418070123307,↵
  7.39339828220179 5.39339828220179,↵
  5.39339828220179 7.39339828220179,↵
  2.14180701233067 12.2597485145237,1 ↵
  18,1 195)
```

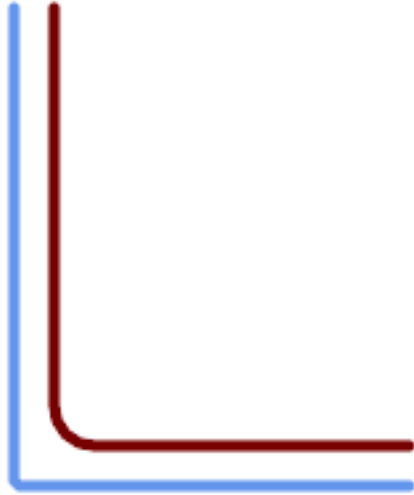


-15, 'quad\_segs=4 join=round' Ausgangslinie und die um -15 Einheiten versetzte Parallele.

```
SELECT ST_AsText(ST_OffsetCurve(geom,↵
  -15, 'quad_segs=4 join=round')) As ↵
  notsocurvy↵
FROM ST_GeomFromText(↵
'LINESTRING(164 16,144 16,124 16,104 ↵
  16,84 16,64 16,↵
  44 16,24 16,20 16,18 16,17 17,↵
  16 18,16 20,16 40,16 60,16 80,16 100,↵
  16 120,16 140,16 160,16 180,16 195)') ↵
  As geom;
```

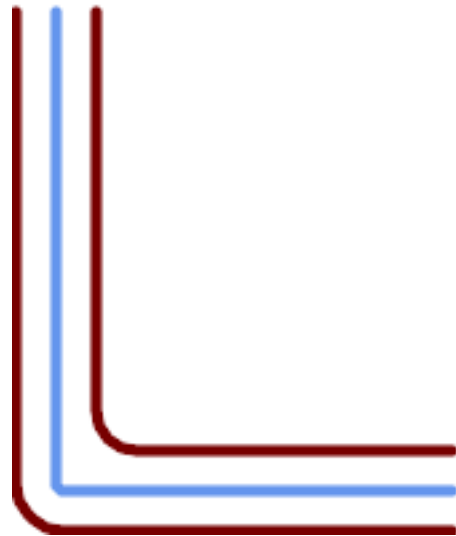
notsocurvy

```
LINESTRING(31 195,31 31,164 31)
```



*doppelter Versatz um es kurviger zu bekommen; beachte die Richtungsänderung, also  $-30 + 15 = -15$*

```
SELECT ST_AsText(ST_OffsetCurve( ←
  ST_OffsetCurve(geom, ←
    -30, 'quad_segs=4 join=round'), -15, ←
    'quad_segs=4 join=round')) As morecurvy
FROM ST_GeomFromText (
'LINESTRING(164 16,144 16,124 16,104 ←
  16,84 16,64 16, ←
  44 16,24 16,20 16,18 16,17 17, ←
  16 18,16 20,16 40,16 60,16 80,16 100, ←
  16 120,16 140,16 160,16 180,16 195)') ←
  As geom;
morecurvy
LINESTRING(164 31,46 31,40.2597485145236 ←
  32.1418070123307, ←
  35.3933982822018 35.3933982822018, ←
  32.1418070123307 40.2597485145237,31 ←
  46,31 195)
```



*Doppelter Versatz um es kurviger zu bekommen, kombiniert mit einem normalen Versatz von 15 um parallele Linien zu erhalten. Überlagert mit dem Original.*

```
SELECT ST_AsText(ST_Collect (
  ST_OffsetCurve(geom, 15, 'quad_segs=4 ←
    join=round'), ←
  ST_OffsetCurve(ST_OffsetCurve(geom, ←
    -30, 'quad_segs=4 join=round'), -15, ←
    'quad_segs=4 join=round') ←
  )
) As parallel_curves
FROM ST_GeomFromText (
'LINESTRING(164 16,144 16,124 16,104 ←
  16,84 16,64 16, ←
  44 16,24 16,20 16,18 16,17 17, ←
  16 18,16 20,16 40,16 60,16 80,16 100, ←
  16 120,16 140,16 160,16 180,16 195)') ←
  As geom;
parallel curves
MULTILINESTRING((164 1,18 ←
  1,12.2597485145237 2.1418070123307, ←
  7.39339828220179 ←
  5.39339828220179,5.39339828220179 7.393398282201 ←
  2.14180701233067 12.2597485145237,1 18,1 ←
  195), ←
(164 31,46 31,40.2597485145236 ←
  32.1418070123307,35.3933982822018 35.39339828220 ←
  32.1418070123307 40.2597485145237,31 ←
  46,31 195))
```

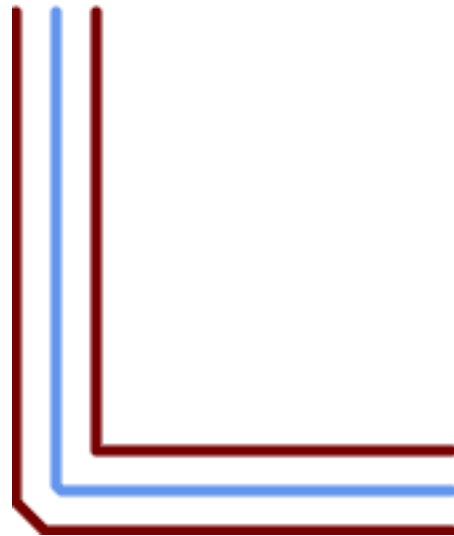


15, 'quad\_segs=4 join=bevel' gemeinsam mit der Ausgangslinie

```
SELECT ST_AsText(ST_OffsetCurve(↵
  ST_GeomFromText(↵
'LINESTRING(164 16,144 16,124 16,104 ↵
  16,84 16,64 16,↵
  44 16,24 16,20 16,18 16,17 17,↵
  16 18,16 20,16 40,16 60,16 80,16 100,↵
  16 120,16 140,16 160,16 180,16 195)')↵
  ,↵
    15, 'quad_segs=4 join=bevel'));
```

output

```
LINESTRING(164 1,18 1,7.39339828220179 ↵
  5.39339828220179,↵
  5.39339828220179 7.39339828220179,1 ↵
  18,1 195)
```



15,-15 collected, join=mitre mitre\_limit=2.1

```
SELECT ST_AsText(ST_Collect(↵
  ST_OffsetCurve(geom, 15, 'quad_segs=4 ↵
  join=mitre mitre_limit=2.2'),↵
  ST_OffsetCurve(geom, -15, 'quad_segs ↵
  =4 join=mitre mitre_limit=2.2')↵
  ) )
```

```
FROM ST_GeomFromText(↵
'LINESTRING(164 16,144 16,124 16,104 ↵
  16,84 16,64 16,↵
  44 16,24 16,20 16,18 16,17 17,↵
  16 18,16 20,16 40,16 60,16 80,16 100,↵
  16 120,16 140,16 160,16 180,16 195)')↵
  As geom;
```

output

```
MULTILINESTRING((164 1,11.7867965644036 ↵
  1,1 11.7867965644036,1 195),↵
  (31 195,31 31,164 31))
```

## Siehe auch

[ST\\_Buffer](#)

## 7.14.18 ST\_PointOnSurface

`ST_PointOnSurface` — Berechnet einen Punkt, der garantiert in einem Polygon oder auf einer Geometrie liegt.

### Synopsis

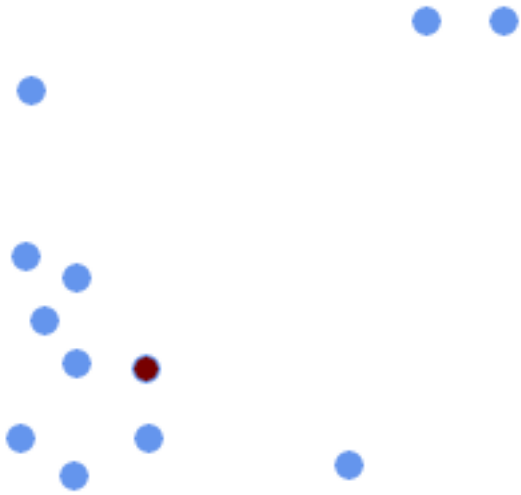
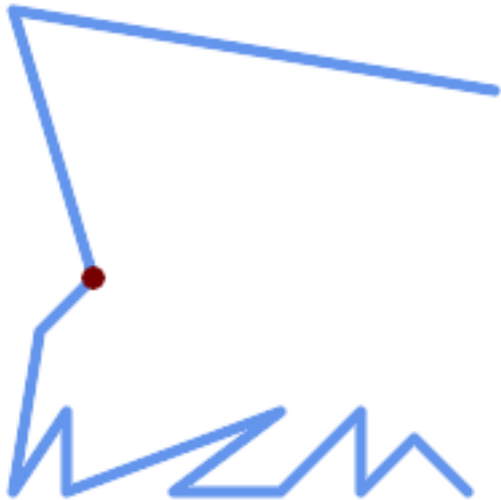
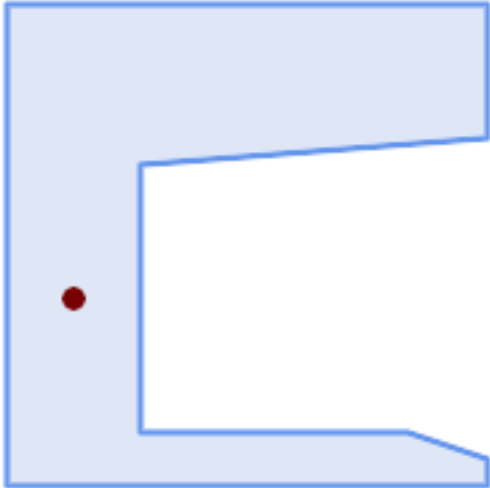
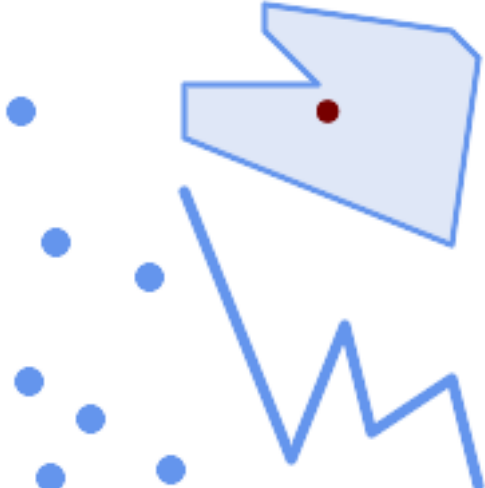
```
geometry ST_PointOnSurface(geometry g1);
```

### Beschreibung

Gibt einen POINT zurück, der garantiert im Inneren einer Fläche liegt (POLYGON, MULTIPOLYGON und CURVEPOLYGON). In PostGIS funktioniert diese Funktion auch für Linien- und Punktgeometrien.

- ✔ Diese Methode implementiert die [OGC Simple Features Implementation Specification for SQL 1.1](#). s3.2.14.2 // s3.2.18.2
- ✔ Diese Methode setzt die SQL/MM-Spezifikation um. SQL-MM 3: 8.1.5, 9.5.6. Die Spezifikationen definieren ST\_PointOnSurface nur für Oberflächengeometrien. PostGIS erweitert die Funktion, um alle gängigen Geometrietypen zu unterstützen. Andere Datenbanken (Oracle, DB2, ArcSDE) scheinen diese Funktion nur für Flächen zu unterstützen. SQL Server 2008 unterstützt alle gängigen Geometrietypen.
- ✔ Diese Funktion unterstützt 3d und lässt den Z-Index nicht fallen.

**Beispiele**

 <p>Punkt auf der Oberfläche eines <i>MULTIPOINT</i></p>	 <p>Punkt auf der Oberfläche eines <i>LINESTRING</i></p>
 <p>Punkt auf der Oberfläche eines <i>POLYGON</i></p>	 <p>Punkt auf der Oberfläche eines <i>GEOMETRYCOLLECTION</i></p>

```
SELECT ST_AsText(ST_PointOnSurface('POINT(0 5)::geometry'));
-----
```

```

POINT(0 5)

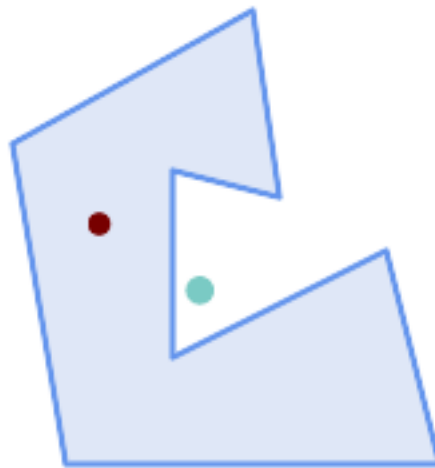
SELECT ST_AsText(ST_PointOnSurface('LINESTRING(0 5, 0 10)')::geometry);
-----
POINT(0 5)

SELECT ST_AsText(ST_PointOnSurface('POLYGON((0 0, 0 5, 5 5, 5 0, 0 0))')::geometry);
-----
POINT(2.5 2.5)

SELECT ST_AsEWKT(ST_PointOnSurface(ST_GeomFromEWKT('LINESTRING(0 5 1, 0 0 1, 0 10 2)')));
-----
POINT(0 0 1)

```

**Beispiel:** Das Ergebnis von `ST_PointOnSurface` liegt garantiert innerhalb von Polygonen, während der von `ST_Centroid` berechnete Punkt außerhalb liegen kann.



*Rot: Punkt auf der Oberfläche; Grün: Flächenschwerpunkt*

```

SELECT ST_AsText(ST_PointOnSurface(geom)) AS pt_on_surf,
       ST_AsText(ST_Centroid(geom)) AS centroid
FROM (SELECT 'POLYGON ((130 120, 120 190, 30 140, 50 20, 190 20,
                       170 100, 90 60, 90 130, 130 120))'::geometry AS geom) AS t;

```

pt_on_surf	centroid
POINT(62.5 110)	POINT(100.18264840182648 85.11415525114155)

#### Siehe auch

[ST\\_Centroid](#), [ST\\_MaximumInscribedCircle](#)

### 7.14.19 ST\_Polygonize

`ST_Polygonize` — Berechnet eine Sammlung von Polygonen, die aus dem Linienwerk einer Reihe von Geometrien gebildet werden.

## Synopsis

```
geometry ST_Polygonize(geometry set geomfield);
geometry ST_Polygonize(geometry[] geom_array);
```

## Beschreibung

Erzeugt eine GeometryCollection, die die Polygone enthält, die durch das Liniengerüst einer Menge von Geometrien gebildet werden. Wenn das eingegebene Linienwerk keine Polygone bildet, wird eine leere GeometryCollection zurückgegeben.

Diese Funktion erzeugt Polygone, die alle abgegrenzten Bereiche abdecken. Wenn das Ergebnis eine gültige polygonale Geometrie bilden soll, verwenden Sie [ST\\_BuildArea](#), um zu verhindern, dass Löcher gefüllt werden.

Note!

### Note

Die Eingabelinie muss korrekt genodet sein, damit diese Funktion ordnungsgemäß funktioniert. Um sicherzustellen, dass die Eingabe nodiert ist, verwenden Sie [ST\\_Node](#) für die Eingabegeometrie vor der Polygonisierung.

Note!

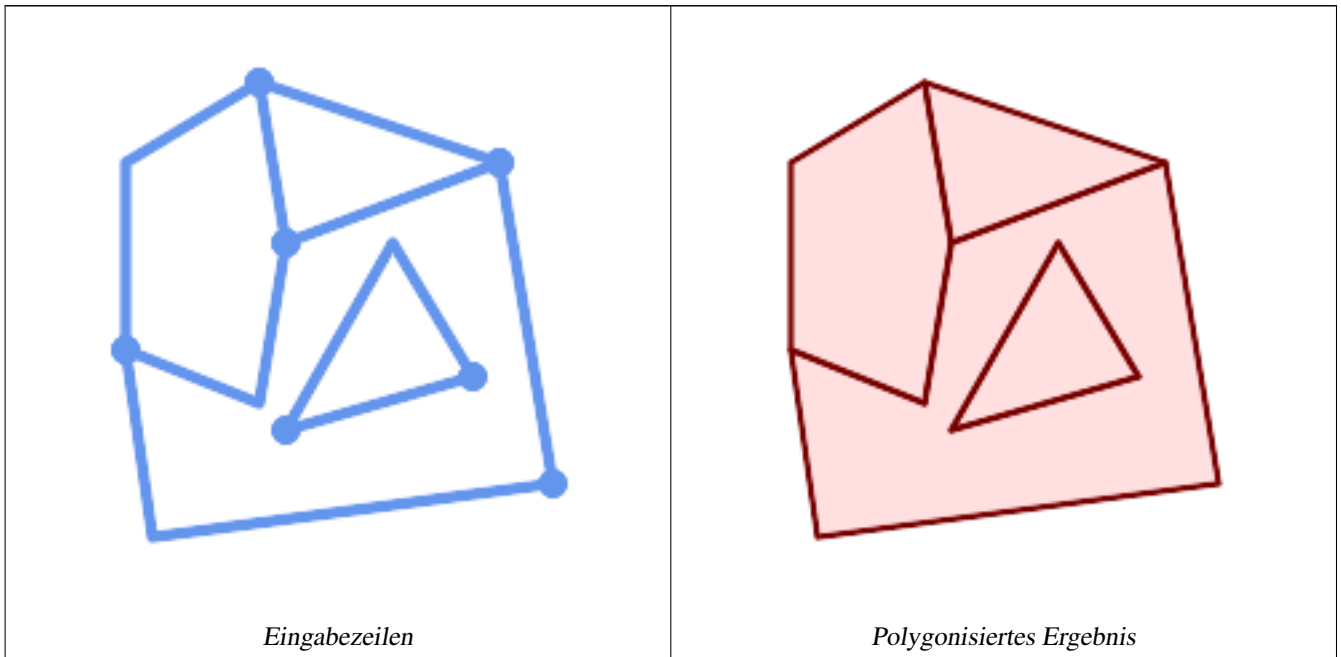
### Note

GeometryCollections können mit externen Werkzeugen schwer zu handhaben sein. Verwenden Sie [ST\\_Dump](#), um das polygonisierte Ergebnis in separate Polygone umzuwandeln.

Wird vom GEOS Modul ausgeführt

Verfügbarkeit: 1.0.0RC1

## Beispiele



```
WITH data(geom) AS (VALUES
  ('LINESTRING (180 40, 30 20, 20 90)')::geometry)
, ('LINESTRING (180 40, 160 160)')::geometry)
```



```

, ('LINESTRING (80 60, 120 130, 150 80)::geometry)
, ('LINESTRING (80 60, 150 80)::geometry)
, ('LINESTRING (20 90, 70 70, 80 130)::geometry)
, ('LINESTRING (80 130, 160 160)::geometry)
, ('LINESTRING (20 90, 20 160, 70 190)::geometry)
, ('LINESTRING (70 190, 80 130)::geometry)
, ('LINESTRING (70 190, 160 160)::geometry)
)
SELECT ST_AsText( ST_Polygonize( geom )
FROM data;
-----
GEOMETRYCOLLECTION (POLYGON ((180 40, 30 20, 20 90, 70 70, 80 130, 160 160, 180 40), (150 ←
80, 120 130, 80 60, 150 80)),
POLYGON ((20 90, 20 160, 70 190, 80 130, 70 70, 20 90)),
POLYGON ((160 160, 80 130, 70 190, 160 160)),
POLYGON ((80 60, 120 130, 150 80, 80 60)))

```

### Polygonisierung einer Tabelle mit Linienzügen:

```

SELECT ST_AsEWKT(ST_Polygonize(geom_4269)) As geomtextrep
FROM (SELECT geom_4269 FROM ma.suffolk_edges) As foo;
-----
SRID=4269;GEOMETRYCOLLECTION(POLYGON((-71.040878 42.285678,-71.040943 42.2856,-71.04096 ←
42.285752,-71.040878 42.285678)),
POLYGON((-71.17166 42.353675,-71.172026 42.354044,-71.17239 42.354358,-71.171794 ←
42.354971,-71.170511 42.354855,
-71.17112 42.354238,-71.17166 42.353675)))

--Use ST_Dump to dump out the polygonize geoms into individual polygons
SELECT ST_AsEWKT((ST_Dump(t.polycoll)).geom) AS geomtextrep
FROM (SELECT ST_Polygonize(geom_4269) AS polycoll
FROM (SELECT geom_4269 FROM ma.suffolk_edges)
As foo) AS t;
-----
SRID=4269;POLYGON((-71.040878 42.285678,-71.040943 42.2856,-71.04096 42.285752,
-71.040878 42.285678))
SRID=4269;POLYGON((-71.17166 42.353675,-71.172026 42.354044,-71.17239 42.354358
,-71.171794 42.354971,-71.170511 42.354855,-71.17112 42.354238,-71.17166 42.353675))

```

### Siehe auch

[ST\\_BuildArea](#), [ST\\_Dump](#), [ST\\_Node](#)

## 7.14.20 ST\_ReducePrecision

`ST_ReducePrecision` — Gibt eine gültige Geometrie mit auf eine Rastertoleranz gerundeten Punkten zurück.

### Synopsis

```
geometry ST_ReducePrecision(geometry g, float8 gridszize);
```

## Beschreibung

Gibt eine gültige Geometrie zurück, bei der alle Punkte auf die angegebene Rastertoleranz gerundet und Features unterhalb der Toleranz entfernt wurden.

Im Gegensatz zu [ST\\_SnapToGrid](#) ist die zurückgegebene Geometrie gültig, ohne Ringselbschnittpunkte oder kollabierte Komponenten.

Die Präzisionsreduzierung kann verwendet werden, um:

- Anpassung der Koordinatengenauigkeit an die Datengenauigkeit
- die Anzahl der für die Darstellung einer Geometrie erforderlichen Koordinaten zu verringern
- Gewährleistung einer gültigen Geometrieausgabe in Formaten, die eine geringere Genauigkeit verwenden (z. B. Textformate wie WKT, GeoJSON oder KML, wenn die Zahl der ausgegebenen Dezimalstellen begrenzt ist).
- Export gültiger Geometrie in Systeme, die eine geringere oder begrenzte Genauigkeit verwenden (z. B. SDE, Oracle-Toleranzwert)

Verfügbarkeit: 3.1.0.

Erfordert GEOS >= 3.9.0.

## Beispiele

```
SELECT ST_AsText(ST_ReducePrecision('POINT(1.412 19.323)', 0.1));
      st_astext
-----
POINT(1.4 19.3)

SELECT ST_AsText(ST_ReducePrecision('POINT(1.412 19.323)', 1.0));
      st_astext
-----
POINT(1 19)

SELECT ST_AsText(ST_ReducePrecision('POINT(1.412 19.323)', 10));
      st_astext
-----
POINT(0 20)
```

Präzisionsreduzierung kann die Anzahl der Scheitelpunkte reduzieren

```
SELECT ST_AsText(ST_ReducePrecision('LINESTRING (10 10, 19.6 30.1, 20 30, 20.3 30, 40 40)', ←
      1));
      st_astext
-----
LINESTRING (10 10, 20 30, 40 40)
```

Präzisionsreduzierung teilt Polygone bei Bedarf auf, um die Gültigkeit zu gewährleisten

```
SELECT ST_AsText(ST_ReducePrecision('POLYGON ((10 10, 60 60.1, 70 30, 40 40, 50 10, 10 10)) ←
      ', 10));
      st_astext
-----
MULTIPOLYGON (((60 60, 70 30, 40 40, 60 60)), ((40 40, 50 10, 10 10, 40 40)))
```

## Siehe auch

[ST\\_SnapToGrid](#), [ST\\_Simplify](#), [ST\\_SimplifyVW](#)

### 7.14.21 ST\_SharedPaths

ST\_SharedPaths — Gibt eine Sammelgeometrie zurück, welche die gemeinsamen Strecken der beiden eingegebenen LineStrings-/MultiLineStrings enthält.

#### Synopsis

```
geometry ST_SharedPaths(geometry lineal1, geometry lineal2);
```

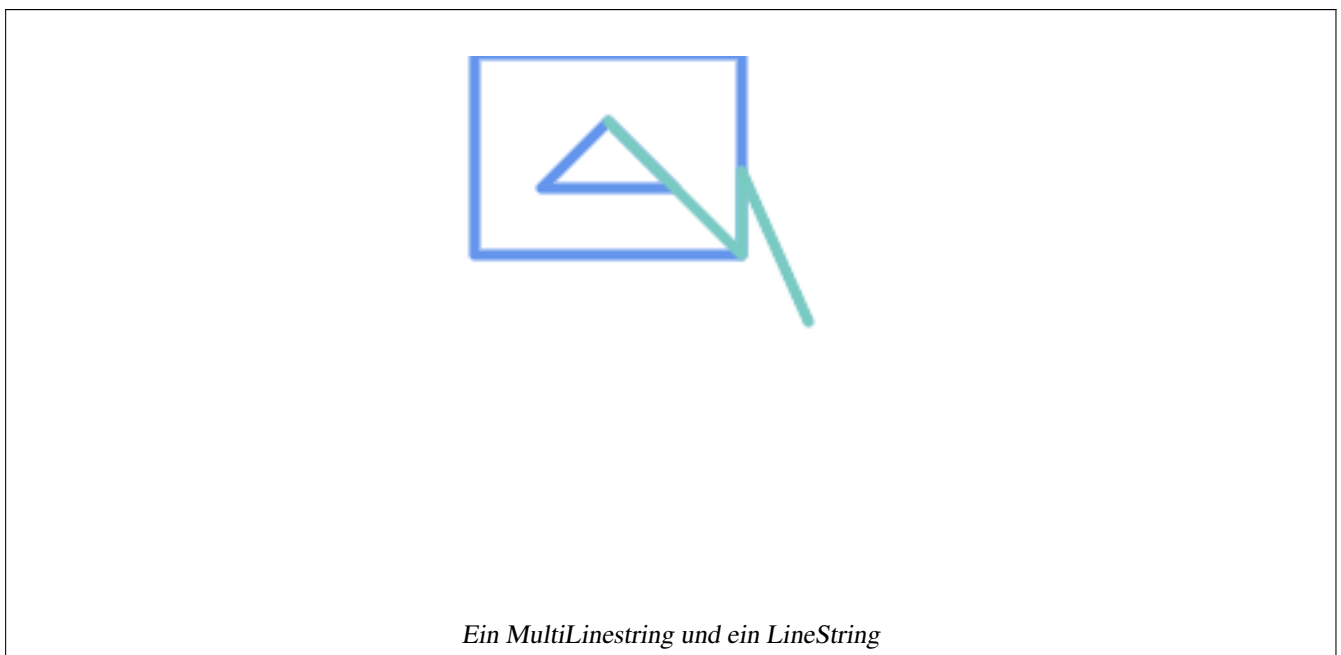
#### Beschreibung

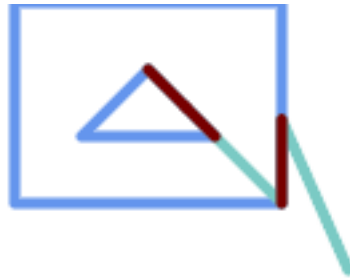
Gibt eine Sammelgeometrie zurück, die die gemeinsamen Pfade zweier Eingabegeometrie enthält. Jene, die in derselben Richtung orientiert sind, werden im ersten Element der Sammelgeometrie, jene die in die entgegengesetzte Richtung orientiert sind, werden im zweiten Element gespeichert. Die Pfade selbst befinden sich in der ersten Geometrie.

Wird vom GEOS Modul ausgeführt

Verfügbarkeit: 2.0.0

#### Beispiele: Gemeinsame Strecken finden





*Die gemeinsame Strecke eines MultiLinestring und Linestring mit überlagerter Ursprungsgeometrie.*

```
SELECT ST_AsText(
  ST_SharedPaths(
    ST_GeomFromText('MULTILINESTRING((26 125,26 200,126 200,126 125,26 125),
      (51 150,101 150,76 175,51 150))'),
    ST_GeomFromText('LINESTRING(151 100,126 156.25,126 125,90 161, 76 175)')
  )
) As wkt
```

wkt

```
-----
GEOMETRYCOLLECTION(MULTILINESTRING((126 156.25,126 125),
(101 150,90 161),(90 161,76 175)),MULTILINESTRING EMPTY)
```

same example but linestring orientation flipped

```
SELECT ST_AsText(
  ST_SharedPaths(
    ST_GeomFromText('LINESTRING(76 175,90 161,126 125,126 156.25,151 100)'),
    ST_GeomFromText('MULTILINESTRING((26 125,26 200,126 200,126 125,26 125),
      (51 150,101 150,76 175,51 150))')
  )
) As wkt
```

wkt

```
-----
GEOMETRYCOLLECTION(MULTILINESTRING EMPTY,
MULTILINESTRING((76 175,90 161),(90 161,101 150),(126 125,126 156.25)))
```

**Siehe auch**

[ST\\_Dump](#), [ST\\_GeometryN](#), [ST\\_NumGeometries](#)

## 7.14.22 ST\_Simplify

`ST_Simplify` — Gibt eine vereinfachte Darstellung einer Geometrie unter Verwendung des Douglas-Peucker-Algorithmus zurück.

### Synopsis

```
geometry ST_Simplify(geometry geom, float tolerance);
geometry ST_Simplify(geometry geom, float tolerance, boolean preserveCollapsed);
```

### Beschreibung

Berechnet eine vereinfachte Darstellung einer Geometrie unter Verwendung des **Douglas-Peucker-Algorithmus**. Die Vereinfachung `Tolerance` ist ein Abstandswert, in den Einheiten der Eingabe SRS. Bei der Vereinfachung werden Eckpunkte entfernt, die innerhalb der Toleranzdistanz des vereinfachten Liniennetzes liegen. Das Ergebnis kann ungültig sein, auch wenn die Eingabe gültig ist.

Die Funktion kann mit jeder Art von Geometrie (einschließlich `GeometryCollections`) aufgerufen werden, aber nur Linien- und Polygonelemente werden vereinfacht. Die Endpunkte der linearen Geometrie bleiben erhalten.

Das Kennzeichen `preserveCollapsed` bewahrt kleine Geometrien, die andernfalls bei der angegebenen Toleranz entfernt werden würden. Wenn zum Beispiel eine 1 m lange Linie mit einer Toleranz von 10 m vereinfacht wird, verschwindet die Linie nicht, wenn `preserveCollapsed` `true` ist. Dieses Flag ist für Rendering-Zwecke nützlich, um zu verhindern, dass sehr kleine Merkmale aus einer Karte verschwinden.



#### Note

Die zurückgegebene Geometrie kann ihre Einfachheit verlieren (siehe `ST_IsSimple`), die Topologie wird möglicherweise nicht beibehalten, und polygonale Ergebnisse können ungültig sein (siehe `ST_IsValid`). Verwenden Sie `ST_SimplifyPreserveTopology`, um die Topologie zu erhalten und die Gültigkeit sicherzustellen.



#### Note

Diese Funktion erhält keine gemeinsamen Grenzen zwischen Polygonen. Verwenden Sie `ST_CoverageSimplify`, wenn dies erforderlich ist.

Verfügbarkeit: 1.2.2

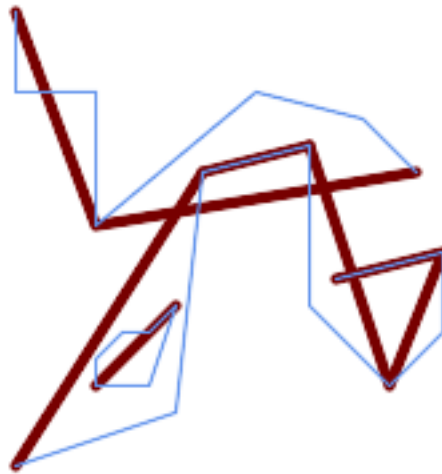
### Beispiele

Ein zu stark vereinfachter Kreis wird zu einem Dreieck, mittelmäßig vereinfacht zum Achteck:

```
SELECT ST_Npoints(geom) AS np_before,
       ST_Npoints(ST_Simplify(geom, 0.1)) AS np01_notbadcircle,
       ST_Npoints(ST_Simplify(geom, 0.5)) AS np05_notquitecircle,
       ST_Npoints(ST_Simplify(geom, 1)) AS np1_octagon,
       ST_Npoints(ST_Simplify(geom, 10)) AS np10_triangle,
       (ST_Simplify(geom, 100) is null) AS np100_geometrygoesaway
FROM (SELECT ST_Buffer('POINT(1 3)', 10,12) As geom) AS t;
```

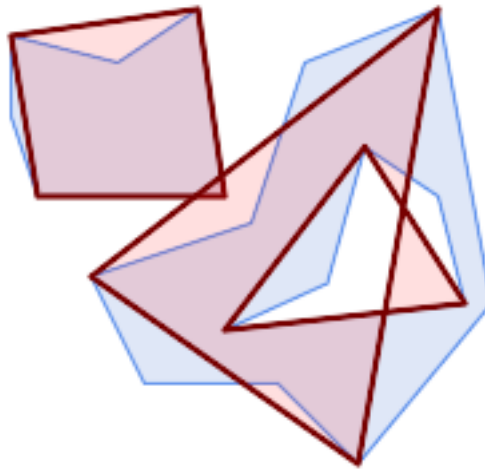
np_before	np01_notbadcircle	np05_notquitecircle	np1_octagon	np10_triangle	np100_geometrygoesaway
49	33	17	9	4	t

Vereinfachung einer Reihe von Linien. Linien können sich nach der Vereinfachung kreuzen.



```
SELECT ST_Simplify(
  'MULTILINESTRING ((20 180, 20 150, 50 150, 50 100, 110 150, 150 140, 170 120), (20 10, 80 ←
    30, 90 120), (90 120, 130 130), (130 130, 130 70, 160 40, 180 60, 180 90, 140 80), ←
    (50 40, 70 40, 80 70, 70 60, 60 60, 50 50, 50 40))',
  40);
```

Vereinfachung eines MultiPolygons. Polygonale Ergebnisse können ungültig sein.



```
SELECT ST_Simplify(
  'MULTIPOLYGON (((90 110, 80 180, 50 160, 10 170, 10 140, 20 110, 90 110)), ((40 80, 100 ←
    100, 120 160, 170 180, 190 70, 140 10, 110 40, 60 40, 40 80), (180 70, 170 110, 142.5 ←
    128.5, 128.5 77.5, 90 60, 180 70)))',
  40);
```

#### Siehe auch

[ST\\_IsSimple](#), [ST\\_SimplifyPreserveTopology](#), [ST\\_SimplifyVW](#), [ST\\_CoverageSimplify](#), [Topologie ST\\_Simplify](#)

#### 7.14.23 ST\_SimplifyPreserveTopology

`ST_SimplifyPreserveTopology` — Gibt eine vereinfachte und gültige Darstellung einer Geometrie zurück, die den Douglas-Peucker-Algorithmus verwendet.

## Synopsis

```
geometry ST_SimplifyPreserveTopology(geometry geom, float tolerance);
```

## Beschreibung

Berechnet eine vereinfachte Darstellung einer Geometrie unter Verwendung einer Variante des [Douglas-Peucker-Algorithmus](#), der die Vereinfachung einschränkt, um sicherzustellen, dass das Ergebnis die gleiche Topologie wie die Eingabe hat. Die Vereinfachung `Tolerance` ist ein Abstandswert in den Einheiten der Eingabe-SRS. Bei der Vereinfachung werden Eckpunkte entfernt, die innerhalb der Toleranzdistanz des vereinfachten Linienwerks liegen, solange die Topologie erhalten bleibt. Das Ergebnis wird gültig und einfach sein, wenn die Eingabe lautet.

Die Funktion kann mit jeder Art von Geometrie (einschließlich `GeometryCollections`) aufgerufen werden, aber nur Linien- und Polygonelemente werden vereinfacht. Bei polygonalen Eingaben hat das Ergebnis die gleiche Anzahl von Ringen (Schalen und Löcher), und die Ringe kreuzen sich nicht. Die Endpunkte der Ringe können vereinfacht werden. Bei linearen Eingaben hat das Ergebnis die gleiche Anzahl von Linien, und die Linien schneiden sich nicht, wenn sie dies in der ursprünglichen Geometrie nicht taten. Die Endpunkte der linearen Geometrie bleiben erhalten.



### Note

Diese Funktion erhält keine gemeinsamen Grenzen zwischen Polygonen. Verwenden Sie [ST\\_CoverageSimplify](#), wenn dies erforderlich ist.

Wird vom GEOS Modul ausgeführt

Verfügbarkeit: 1.3.3

## Beispiele

Für dasselbe Beispiel wie [ST\\_Simplify](#) verhindert `ST_SimplifyPreserveTopology` eine zu starke Vereinfachung. Der Kreis kann höchstens zu einem Quadrat werden.

```
SELECT  ST_Npoints(geom) AS np_before,
        ST_NPoints(ST_SimplifyPreserveTopology(geom, 0.1)) AS np01_notbadcircle,
        ST_NPoints(ST_SimplifyPreserveTopology(geom, 0.5)) AS np05_notquitecircle,
        ST_NPoints(ST_SimplifyPreserveTopology(geom, 1))   AS np1_octagon,
        ST_NPoints(ST_SimplifyPreserveTopology(geom, 10))  AS np10_square,
        ST_NPoints(ST_SimplifyPreserveTopology(geom, 100)) AS np100_stillsquare
FROM (SELECT ST_Buffer('POINT(1 3)', 10,12) AS geom) AS t;
```

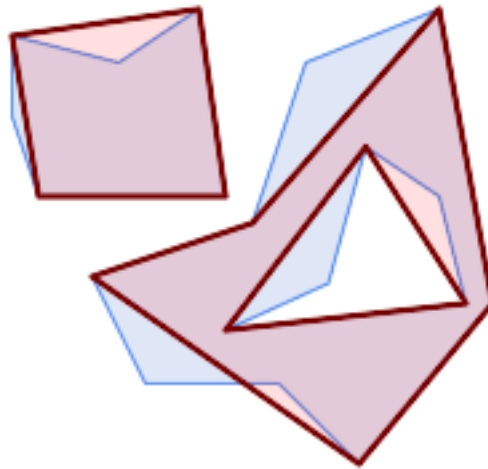
np_before	np01_notbadcircle	np05_notquitecircle	np1_octagon	np10_square	np100_stillsquare
49	33	17	9	5	5

Vereinfachung einer Reihe von Linien, wobei die Topologie der sich nicht schneidenden Linien erhalten bleibt.



```
SELECT ST_SimplifyPreserveTopology(
  'MULTILINESTRING ((20 180, 20 150, 50 150, 50 100, 110 150, 150 140, 170 120), (20 10, 80 ←
    30, 90 120), (90 120, 130 130), (130 130, 130 70, 160 40, 180 60, 180 90, 140 80), ←
    (50 40, 70 40, 80 70, 70 60, 60 60, 50 50, 50 40))',
  40);
```

Vereinfachung eines MultiPolygons unter Beibehaltung der Topologie von Schalen und Löchern.



```
SELECT ST_SimplifyPreserveTopology(
  'MULTIPOLYGON (((90 110, 80 180, 50 160, 10 170, 10 140, 20 110, 90 110)), ((40 80, 100 ←
    100, 120 160, 170 180, 190 70, 140 10, 110 40, 60 40, 40 80), (180 70, 170 110, 142.5 ←
    128.5, 128.5 77.5, 90 60, 180 70)))',
  40);
```

#### Siehe auch

[ST\\_Simplify](#), [ST\\_SimplifyVW](#), [ST\\_CoverageSimplify](#)

#### 7.14.24 ST\_SimplifyPolygonHull

`ST_SimplifyPolygonHull` — Berechnet eine vereinfachte topologieerhaltende äußere oder innere Hülle einer polygonalen Geometrie.



## Synopsis

geometry **ST\_SimplifyPolygonHull**(geometry param\_geom, float vertex\_fraction, boolean is\_outer = true);

## Beschreibung

Berechnet eine vereinfachte topologieerhaltende äußere oder innere Hülle einer polygonalen Geometrie. Eine äußere Hülle deckt die Eingabegeometrie vollständig ab. Eine innere Hülle wird vollständig von der Eingabegeometrie abgedeckt. Das Ergebnis ist eine polygonale Geometrie, die durch eine Teilmenge der Eingabepunkte gebildet wird. MultiPolygone und Löcher werden behandelt und führen zu einem Ergebnis mit derselben Struktur wie die Eingabe.

Die Verringerung der Scheitelpunktzahl wird durch den Parameter `vertex_fraction` gesteuert, der eine Zahl im Bereich von 0 bis 1 ist. Niedrigere Werte führen zu einfacheren Ergebnissen mit einer geringeren Anzahl von Scheitelpunkten und einer geringeren Konkavität. Ein Scheitelpunktanteil von 1,0 ergibt sowohl für äußere als auch für innere Hüllen die ursprüngliche Geometrie. Für äußere Hüllen ergibt ein Wert von 0,0 die konvexe Hülle (für ein einzelnes Polygon); für innere Hüllen ergibt er ein Dreieck.

Bei der Vereinfachung werden nach und nach die konkaven Ecken entfernt, die die geringste Fläche enthalten, bis das Ziel für die Anzahl der Eckpunkte erreicht ist. Er verhindert, dass sich Kanten kreuzen, so dass das Ergebnis immer eine gültige polygonale Geometrie ist.

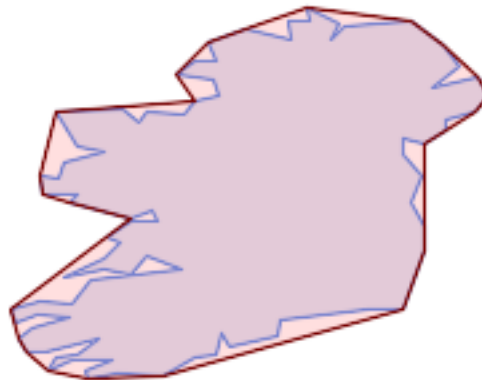
Um bessere Ergebnisse bei Geometrien zu erzielen, die relativ lange Liniensegmente enthalten, kann es notwendig sein, die Eingabe zu "segmentieren", wie unten gezeigt.

Wird vom GEOS Modul ausgeführt

Verfügbarkeit: 3.3.0.

Erfordert GEOS >= 3.11.0.

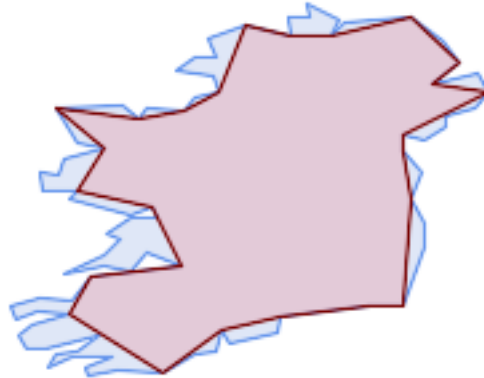
## Beispiele



*Äußere Hülle eines Polygons*

```
SELECT ST_SimplifyPolygonHull(  
  'POLYGON ((131 158, 136 163, 161 165, 173 156, 179 148, 169 140, 186 144, 190 137, 185 ↵  
    131, 174 128, 174 124, 166 119, 158 121, 158 115, 165 107, 161 97, 166 88, 166 79, 158 ↵  
    57, 145 57, 112 53, 111 47, 93 43, 90 48, 88 40, 80 39, 68 32, 51 33, 40 31, 39 34, ↵  
    49 38, 34 38, 25 34, 28 39, 36 40, 44 46, 24 41, 17 41, 14 46, 19 50, 33 54, 21 55, 13 ↵  
    52, 11 57, 22 60, 34 59, 41 68, 75 72, 62 77, 56 70, 46 72, 31 69, 46 76, 52 82, 47 ↵  
    84, 56 90, 66 90, 64 94, 56 91, 33 97, 36 100, 23 100, 22 107, 29 106, 31 112, 46 116, ↵  
    36 118, 28 131, 53 132, 59 127, 62 131, 76 130, 80 135, 89 137, 87 143, 73 145, 80 ↵  
    150, 88 150, 85 157, 99 162, 116 158, 115 165, 123 165, 122 170, 134 164, 131 158))',
```

```
0.3);
```



*Innere Hülle eines Polygons*

```
SELECT ST_SimplifyPolygonHull(
  'POLYGON ((131 158, 136 163, 161 165, 173 156, 179 148, 169 140, 186 144, 190 137, 185 ↵
    131, 174 128, 174 124, 166 119, 158 121, 158 115, 165 107, 161 97, 166 88, 166 79, 158 ↵
    57, 145 57, 112 53, 111 47, 93 43, 90 48, 88 40, 80 39, 68 32, 51 33, 40 31, 39 34, ↵
    49 38, 34 38, 25 34, 28 39, 36 40, 44 46, 24 41, 17 41, 14 46, 19 50, 33 54, 21 55, 13 ↵
    52, 11 57, 22 60, 34 59, 41 68, 75 72, 62 77, 56 70, 46 72, 31 69, 46 76, 52 82, 47 ↵
    84, 56 90, 66 90, 64 94, 56 91, 33 97, 36 100, 23 100, 22 107, 29 106, 31 112, 46 116, ↵
    36 118, 28 131, 53 132, 59 127, 62 131, 76 130, 80 135, 89 137, 87 143, 73 145, 80 ↵
    150, 88 150, 85 157, 99 162, 116 158, 115 165, 123 165, 122 170, 134 164, 131 158))',
  0.3, false);
```



*Vereinfachung der Außenhülle eines MultiPolygons, mit Segmentierung*

```
SELECT ST_SimplifyPolygonHull(
  ST_Segmentize(ST_Letters('xt'), 2.0),
  0.1);
```

**Siehe auch**

[ST\\_ConvexHull](#), [ST\\_SimplifyVW](#), [ST\\_ConcaveHull](#), [ST\\_Segmentize](#)

**7.14.25 ST\_SimplifyVW**

`ST_SimplifyVW` — Liefert eine vereinfachte Darstellung einer Geometrie unter Verwendung des Visvalingam-Whyatt-Algorithmus

**Synopsis**

```
geometry ST_SimplifyVW(geometry geom, float tolerance);
```

**Beschreibung**

Liefert eine vereinfachte Darstellung einer Geometrie unter Verwendung des [Visvalingam-Whyatt-Algorithmus](#). Die Vereinfachung `Tolerance` ist ein Flächenwert, in den Einheiten der Eingabe SRS. Bei der Vereinfachung werden Eckpunkte entfernt, die "Ecken" mit einer Fläche kleiner als die Toleranz bilden. Das Ergebnis ist möglicherweise nicht gültig, auch wenn die Eingabe stimmt.

Die Funktion kann mit jeder Art von Geometrie (einschließlich `GeometryCollections`) aufgerufen werden, aber nur Linien- und Polygonelemente werden vereinfacht. Die Endpunkte der linearen Geometrie bleiben erhalten.

**Note**

Die zurückgegebene Geometrie kann ihre Einfachheit verlieren (siehe [ST\\_IsSimple](#)), die Topologie bleibt möglicherweise nicht erhalten, und polygonale Ergebnisse können ungültig sein (siehe [ST\\_IsValid](#)). Verwenden Sie [ST\\_SimplifyPreserveTopology](#), um die Topologie zu erhalten und die Gültigkeit sicherzustellen. [ST\\_CoverageSimplify](#) erhält ebenfalls die Topologie und die Gültigkeit.

**Note**

Diese Funktion erhält keine gemeinsamen Grenzen zwischen Polygonen. Verwenden Sie [ST\\_CoverageSimplify](#), wenn dies erforderlich ist.

**Note**

Diese Funktion kann mit 3D umgehen und die dritte Dimension beeinflusst auch das Ergebnis.

Verfügbarkeit: 2.2.0

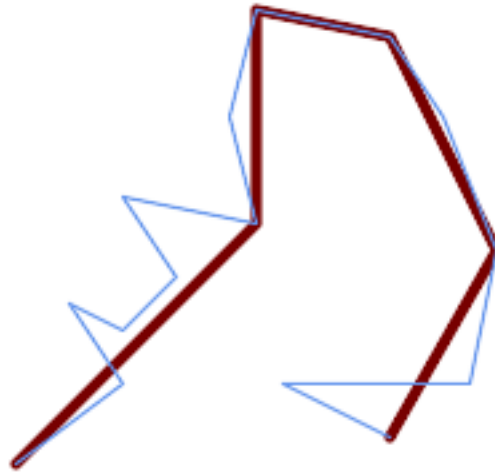
**Beispiele**

Ein `LineString` wird mit einer minimalen Flächentoleranz von 30 vereinfacht.

```
SELECT ST_AsText(ST_SimplifyVW(geom,30)) simplified
FROM (SELECT 'LINESTRING(5 2, 3 8, 6 20, 7 25, 10 10)::geometry AS geom) AS t;

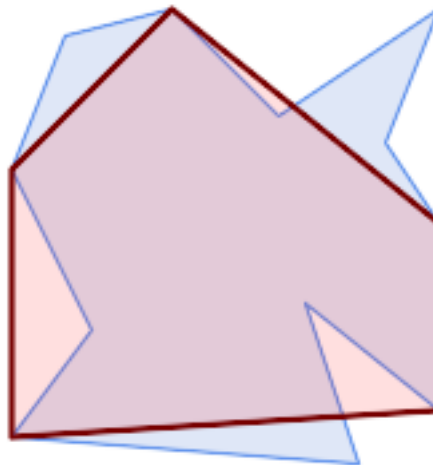
simplified
-----
LINESTRING(5 2,7 25,10 10)
```

Vereinfachung einer Linie.



```
SELECT ST_SimplifyVW(
  'LINESTRING (10 10, 50 40, 30 70, 50 60, 70 80, 50 110, 100 100, 90 140, 100 180, 150 170, 170 140, 190 90, 180 40, 110 40, 150 20)',
  1600);
```

Vereinfachung eines Polygons.



```
SELECT ST_SimplifyVW(
  'MULTIPOLYGON (((90 110, 80 180, 50 160, 10 170, 10 140, 20 110, 90 110)), ((40 80, 100 100, 120 160, 170 180, 190 70, 140 10, 110 40, 60 40, 40 80), (180 70, 170 110, 142.5 128.5, 128.5 77.5, 90 60, 180 70)))',
  40);
```

#### Siehe auch

[ST\\_SetEffectiveArea](#), [ST\\_Simplify](#), [ST\\_SimplifyPreserveTopology](#), [ST\\_CoverageSimplify](#), Topologie [ST\\_Simplify](#)

#### 7.14.26 ST\_SetEffectiveArea

**ST\_SetEffectiveArea** — Legt die effektive Fläche für jeden Scheitelpunkt unter Verwendung des Visvalingam-Whyatt-Algorithmus fest.

## Synopsis

```
geometry ST_SetEffectiveArea(geometry geom, float threshold = 0, integer set_area = 1);
```

## Beschreibung

Setzt die Nutzfläche für jeden Knoten. Verwendet den Visvalingam-Whyatt Algorithmus. Die Nutzfläche wird als M-Wert des Knoten gespeichert. Wird der optionale Parameter "threshold" verwendet, so wird eine vereinfachte Geometrie zurückgegeben, die nur jene Knoten enthält, deren Nutzfläche größer oder gleich dem Schwellenwert ist.

Diese Funktion kann für die serverseitige Vereinfachung, mittels eines Schwellenwerts verwendet werden. Eine andere Möglichkeit besteht darin, einen Schwellenwert von null anzugeben. In diesem Fall wird die gesamte Geometrie inklusive der Nutzflächen als M-Werte zurückgegeben, welche dann am Client für eine rasche Vereinfachung genutzt werden können.

Tut zurzeit nur mit (Multi)Lines und (Multi)Polygons etwas, kann aber gefahrlos mit jedem geometrischen Datentyp verwendet werden. Da die Vereinfachung auf einer Objekt zu Objekt Basis passiert, können Sie diese Funktion auch mit einer Sammelgeometrie speisen.



### 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](#)**7.14.27 ST\_TriangulatePolygon**

ST\_TriangulatePolygon — Berechnet die eingeschränkte Delaunay-Triangulation von Polygonen

**Synopsis**geometry **ST\_TriangulatePolygon**(geometry geom);**Beschreibung**

Berechnet die eingeschränkte Delaunay-Triangulation von Polygonen. Löcher und Multipolygone werden unterstützt.

Die "eingeschränkte Delaunay-Triangulation" eines Polygons ist eine Menge von Dreiecken, die aus den Scheitelpunkten des Polygons gebildet werden und es genau abdecken, mit dem maximalen Gesamtinnenwinkel aller möglichen Triangulationen. Sie liefert die "beste Qualität" der Triangulation des Polygons.

Verfügbarkeit: 3.3.0.

Erfordert GEOS &gt;= 3.11.0.

**Beispiel**

Triangulierung eines Quadrats.

```
SELECT ST_AsText (
  ST_TriangulatePolygon('POLYGON((0 0, 0 1, 1 1, 1 0, 0 0))');

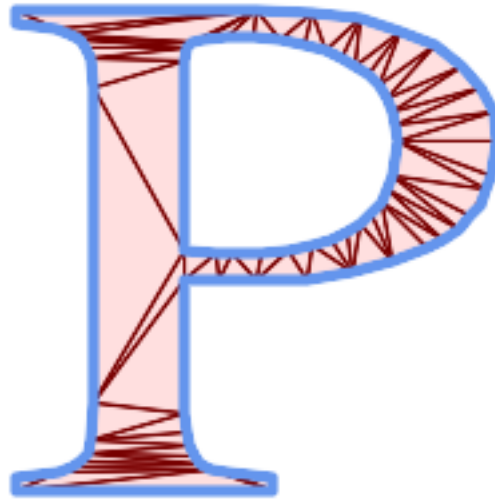
```

st_astext
GEOMETRYCOLLECTION(POLYGON((0 0,0 1,1 1,0 0)),POLYGON((1 1,1 0,0 0,1 1)))

**Beispiel**

Triangulation des Buchstabens P.

```
SELECT ST_AsText(ST_TriangulatePolygon(
  'POLYGON ((26 17, 31 19, 34 21, 37 24, 38 29, 39 43, 39 161, 38 172, 36 176, 34 179, 30 ↵
    181, 25 183, 10 185, 10 190, 100 190, 121 189, 139 187, 154 182, 167 177, 177 169, ↵
    184 161, 189 152, 190 141, 188 128, 186 123, 184 117, 180 113, 176 108, 170 104, 164 ↵
    101, 151 96, 136 92, 119 89, 100 89, 86 89, 73 89, 73 39, 74 32, 75 27, 77 23, 79 ↵
    20, 83 18, 89 17, 106 15, 106 10, 10 10, 10 15, 26 17), (152 147, 151 152, 149 157, ↵
    146 162, 142 166, 137 169, 132 172, 126 175, 118 177, 109 179, 99 180, 89 180, 80 ↵
    179, 76 178, 74 176, 73 171, 73 100, 85 99, 91 99, 102 99, 112 100, 121 102, 128 ↵
    104, 134 107, 139 110, 143 114, 147 118, 149 123, 151 128, 153 141, 152 147))'
));
```



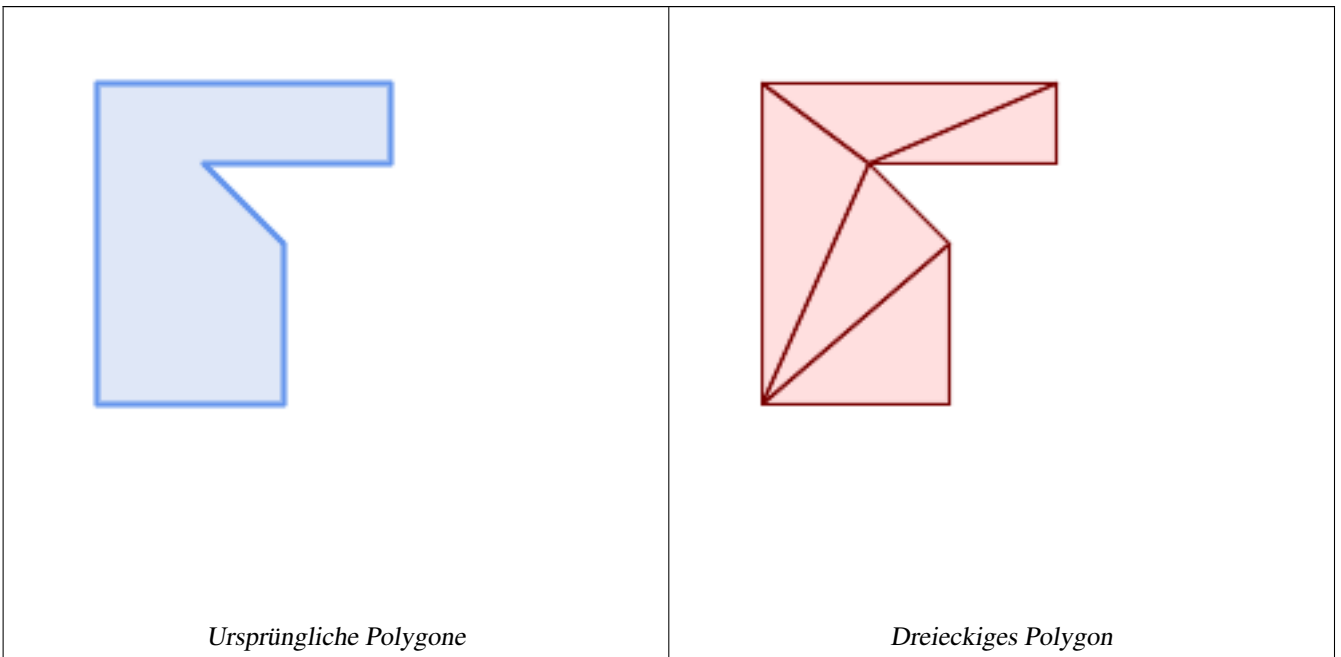
*Polygon-Triangulation*

### Gleiches Beispiel wie ST\_Tesselate

```
SELECT ST_TriangulatePolygon(
    'POLYGON (( 10 190, 10 70, 80 70, 80 130, 50 160, 120 160, 120 190, 10 190 ←
    ))'::geometry
);
```

### ST\_AsText Ausgabe

```
GEOMETRYCOLLECTION(POLYGON((50 160,120 190,120 160,50 160))
, POLYGON((10 70,80 130,80 70,10 70))
, POLYGON((50 160,10 70,10 190,50 160))
, POLYGON((120 190,50 160,10 190,120 190))
, POLYGON((80 130,10 70,50 160,80 130))
```



**Siehe auch**

[ST\\_ConstrainedDelaunayTriangles](#), [ST\\_DelaunayTriangles](#), [ST\\_Tessellate](#)

**7.14.28 ST\_VoronoiLines**

`ST_VoronoiLines` — Gibt die Grenzen des Voronoi-Diagramms der Eckpunkte einer Geometrie zurück.

**Synopsis**

```
geometry ST_VoronoiLines( geometry geom , float8 tolerance = 0.0 , geometry extend_to = NULL );
```

**Beschreibung**

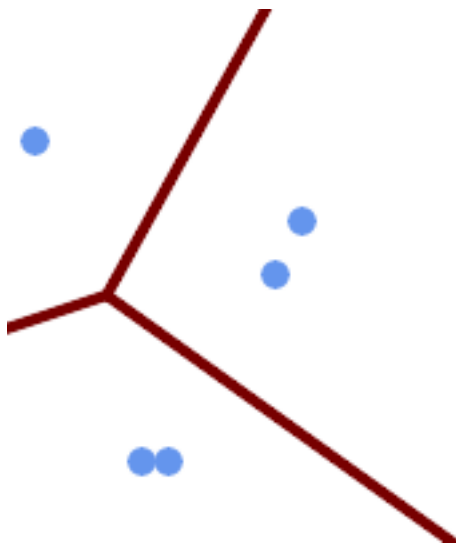
Berechnet ein zweidimensionales **Voronoi-Diagramm** aus den Scheitelpunkten der angegebenen Geometrie und gibt die Grenzen zwischen den Zellen im Diagramm als `MultiLineString` zurück. Gibt null zurück, wenn die Eingabegeometrie null ist. Gibt eine leere Geometriesammlung zurück, wenn die Eingabegeometrie nur einen Scheitelpunkt enthält. Gibt eine leere Geometriesammlung zurück, wenn die `extend_to`-Hülle eine Fläche von Null hat.

Optionale Parameter:

- **Toleranz**: Der Abstand, innerhalb dessen Scheitelpunkte als gleichwertig betrachtet werden. Die Robustheit des Algorithmus kann durch Angabe einer Toleranzdistanz ungleich Null verbessert werden. (Voreinstellung = 0.0)
- **extend\_to**: Wenn vorhanden, wird das Diagramm so erweitert, dass es die Hüllkurve der übergebenen Geometrie abdeckt, sofern diese nicht kleiner als die Standardhüllkurve ist (Standard = NULL, Standardhüllkurve ist die um etwa 50 % erweiterte Bounding Box der Eingabe).

Wird vom GEOS Modul ausgeführt

Verfügbarkeit: 2.3.0

**Beispiele**

*Linien des Voronoi-Diagramms, mit einer Toleranz von 30 Einheiten*



```
SELECT ST_VoronoiLines(  
    'MULTIPOINT (50 30, 60 30, 100 100,10 150, 110 120) '::geometry,  
    30) AS geom;
```

```
ST_AsText output  
MULTILINESTRING((135.555555555556 270,36.8181818181818 92.2727272727273), (36.8181818181818 ←  
    92.2727272727273,-110 43.3333333333333), (230 -45.7142857142858,36.8181818181818 ←  
    92.2727272727273))
```

## Siehe auch

[ST\\_DelaunayTriangles](#), [ST\\_VoronoiPolygons](#)

## 7.14.29 ST\_VoronoiPolygons

`ST_VoronoiPolygons` — Gibt die Zellen des Voronoi-Diagramms der Scheitelpunkte einer Geometrie zurück.

### Synopsis

```
geometry ST_VoronoiPolygons( geometry geom , float8 tolerance = 0.0 , geometry extend_to = NULL );
```

### Beschreibung

Berechnet ein zweidimensionales **Voronoi-Diagramm** aus den Scheitelpunkten der angegebenen Geometrie. Das Ergebnis ist eine GEOMETRIE-SAMMLUNG von POLYGONEN, die einen Bereich abdeckt, der größer ist als die Ausdehnung der Eingabescheitelpunkte. Gibt null zurück, wenn die Eingabegeometrie null ist. Gibt eine leere Geometriesammlung zurück, wenn die Eingabegeometrie nur einen Scheitelpunkt enthält. Gibt eine leere Geometriesammlung zurück, wenn die `extend_to`-Hüllkurve eine Fläche von Null hat.

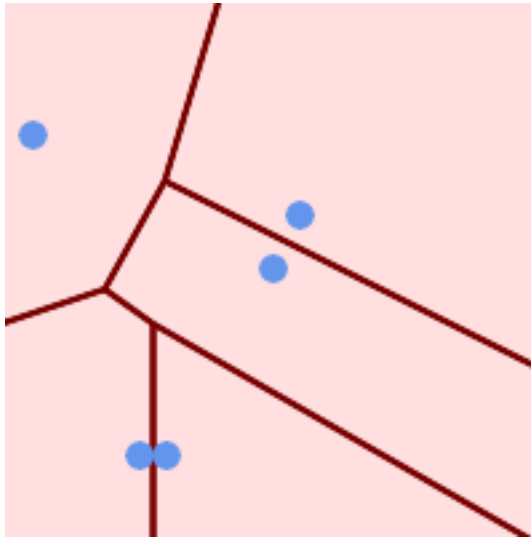
Optionale Parameter:

- **Toleranz**: Der Abstand, innerhalb dessen Scheitelpunkte als gleichwertig betrachtet werden. Die Robustheit des Algorithmus kann durch Angabe einer Toleranzdistanz ungleich Null verbessert werden. (Voreinstellung = 0.0)
- **extend\_to**: Wenn vorhanden, wird das Diagramm so erweitert, dass es die Hüllkurve der übergebenen Geometrie abdeckt, sofern diese nicht kleiner als die Standardhüllkurve ist (Standard = NULL, Standardhüllkurve ist die um etwa 50 % erweiterte Bounding Box der Eingabe).

Wird vom GEOS Modul ausgeführt

Verfügbarkeit: 2.3.0

## Beispiele

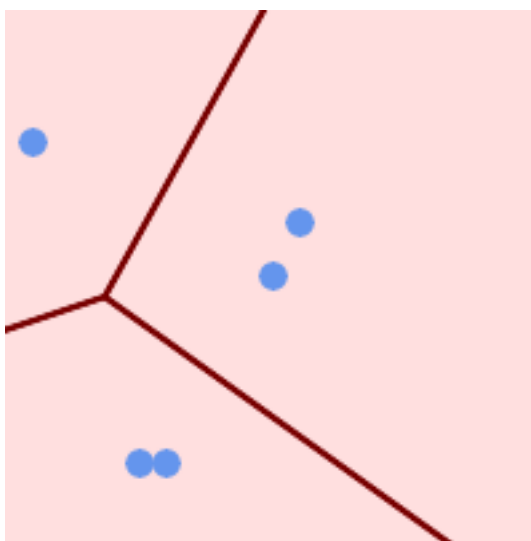


Punkte über dem Voronoi Diagramm

```
SELECT ST_VoronoiPolygons(
    'MULTIPOINT (50 30, 60 30, 100 10,10 150, 110 120)>::geometry
) AS geom;
```

ST\_AsText output

```
GEOMETRYCOLLECTION(POLYGON((-110 43.33333333333333,-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-Diagramm, mit einer Toleranz von 30 Einheiten

```
SELECT ST_VoronoiPolygons(  
    'MULTIPOINT (50 30, 60 30, 100 100,10 150, 110 120) '::geometry,  
    30) AS geom;
```

ST\_AsText output

```
GEOMETRYCOLLECTION(POLYGON((-110 43.33333333333333,-110 270,100.5 270,59.3478260869565 ↔  
    132.826086956522,36.8181818181818 92.2727272727273,-110 43.3333333333333)),  
POLYGON((230 47.5,230 -45.7142857142858,36.8181818181818 92.2727272727273,59.3478260869565 ↔  
    132.826086956522,230 47.5)),POLYGON((230 -45.7142857142858,230 -90,-110 -90,-110 ↔  
    43.3333333333333,36.8181818181818 92.2727272727273,230 -45.7142857142858)),  
POLYGON((100.5 270,230 270,230 47.5,59.3478260869565 132.826086956522,100.5 270)))
```

## Siehe auch

[ST\\_DelaunayTriangles](#), [ST\\_VoronoiLines](#)

## 7.15 Deckungen

### 7.15.1 ST\_CoverageInvalidEdges

`ST_CoverageInvalidEdges` — Fensterfunktion, die Stellen findet, an denen die Polygone keine gültige Abdeckung bilden.

#### Synopsis

```
geometry ST_CoverageInvalidEdges(geometry winset geom, float8 tolerance = 0);
```

#### Beschreibung

Eine Fensterfunktion, die prüft, ob die Polygone in der Fensterpartition eine gültige polygonale Abdeckung bilden. Sie gibt lineare Indikatoren zurück, die die Lage der ungültigen Kanten (falls vorhanden) in jedem Polygon anzeigen.

Eine Menge gültiger Polygone ist eine gültige Abdeckung, wenn die folgenden Bedingungen erfüllt sind:

- **Nicht überlappend** - Polygone überlappen sich nicht (ihre Innenräume schneiden sich nicht)
- **Edge-Matched** - Scheitelpunkte entlang gemeinsamer Kanten sind identisch

Als Fensterfunktion wird für jedes Eingabepolygon ein Wert zurückgegeben. Für Polygone, die eine oder mehrere der Gültigkeitsbedingungen verletzen, ist der Rückgabewert ein `MULTILINESTRING`, der die problematischen Kanten enthält. Flächendeckend gültige Polygone geben den Wert `NULL` zurück. Nicht-polygonale oder leere Geometrien liefern ebenfalls `NULL`-Werte.

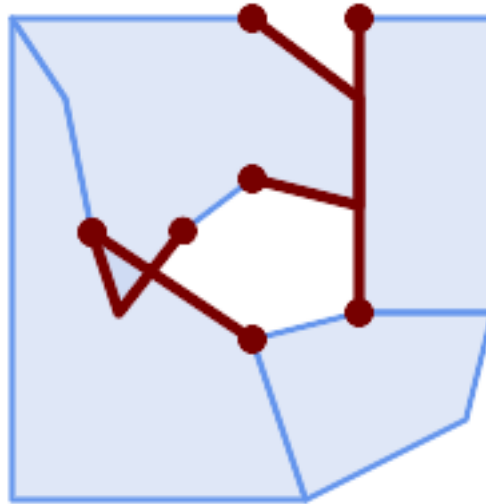
Die Bedingungen erlauben es, dass eine gültige Abdeckung Lücken (Lücken zwischen Polygonen) enthält, solange die umgebenden Polygone kantenangepasst sind. Sehr schmale Lücken sind jedoch oft unerwünscht. Wenn der Parameter `tolerance` mit einem Abstand ungleich Null angegeben wird, werden auch Kanten, die engere Lücken bilden, als ungültig zurückgegeben.

Die Polygone, die auf die Gültigkeit der Abdeckung geprüft werden, müssen ebenfalls gültige Geometrien sein. Dies kann mit [ST\\_IsValid](#) überprüft werden.

Verfügbarkeit: 3.4.0

Benötigt GEOS >= 3.12.0

## Beispiele



*Ungültige Kanten aufgrund von Überschneidungen und nicht übereinstimmenden Scheitelpunkten*

```
WITH coverage(id, geom) AS (VALUES
  (1, 'POLYGON ((10 190, 30 160, 40 110, 100 70, 120 10, 10 10, 10 190))'::geometry),
  (2, 'POLYGON ((100 190, 10 190, 30 160, 40 110, 50 80, 74 110.5, 100 130, 140 120, 140 ←
    160, 100 190))'::geometry),
  (3, 'POLYGON ((140 190, 190 190, 190 80, 140 80, 140 190))'::geometry),
  (4, 'POLYGON ((180 40, 120 10, 100 70, 140 80, 190 80, 180 40))'::geometry)
)
SELECT id, ST_AsText(ST_CoverageInvalidEdges(geom) OVER ())
FROM coverage;

id |          st_astext
---+-----
 1 | LINESTRING (40 110, 100 70)
 2 | MULTILINESTRING ((100 130, 140 120, 140 160, 100 190), (40 110, 50 80, 74 110.5))
 3 | LINESTRING (140 80, 140 190)
 4 | null

-- Test entire table for coverage validity
SELECT true = ALL (
  SELECT ST_CoverageInvalidEdges(geom) OVER () IS NULL
  FROM coverage
);
```

## Siehe auch

[ST\\_IsValid](#), [ST\\_CoverageUnion](#), [ST\\_CoverageSimplify](#)

### 7.15.2 ST\_CoverageSimplify

`ST_CoverageSimplify` — Fensterfunktion, die die Kanten einer polygonalen Abdeckung vereinfacht.

## Synopsis

geometry `ST_CoverageSimplify`(geometry winset geom, float8 tolerance, boolean simplifyBoundary = true);

## Beschreibung

Eine Fensterfunktion, die die Kanten von Polygonen in einer polygonalen Abdeckung vereinfacht. Durch die Vereinfachung bleibt die Topologie der Abdeckung erhalten. Das bedeutet, dass die vereinfachten Ausgabepolygone entlang gemeinsamer Kanten konsistent sind und immer noch eine gültige Abdeckung bilden.

Die Vereinfachung verwendet eine Variante des **Visvalingam-Whyatt-Algorithmus**. Der Parameter *Toleranz* hat die Einheit Abstand und ist ungefähr gleich der Quadratwurzel der zu vereinfachenden Dreiecksflächen.

Um nur die "internen" Kanten der Abdeckung zu vereinfachen (diejenigen, die von zwei Polygonen geteilt werden), setzen Sie den Parameter *simplifyBoundary* auf false.



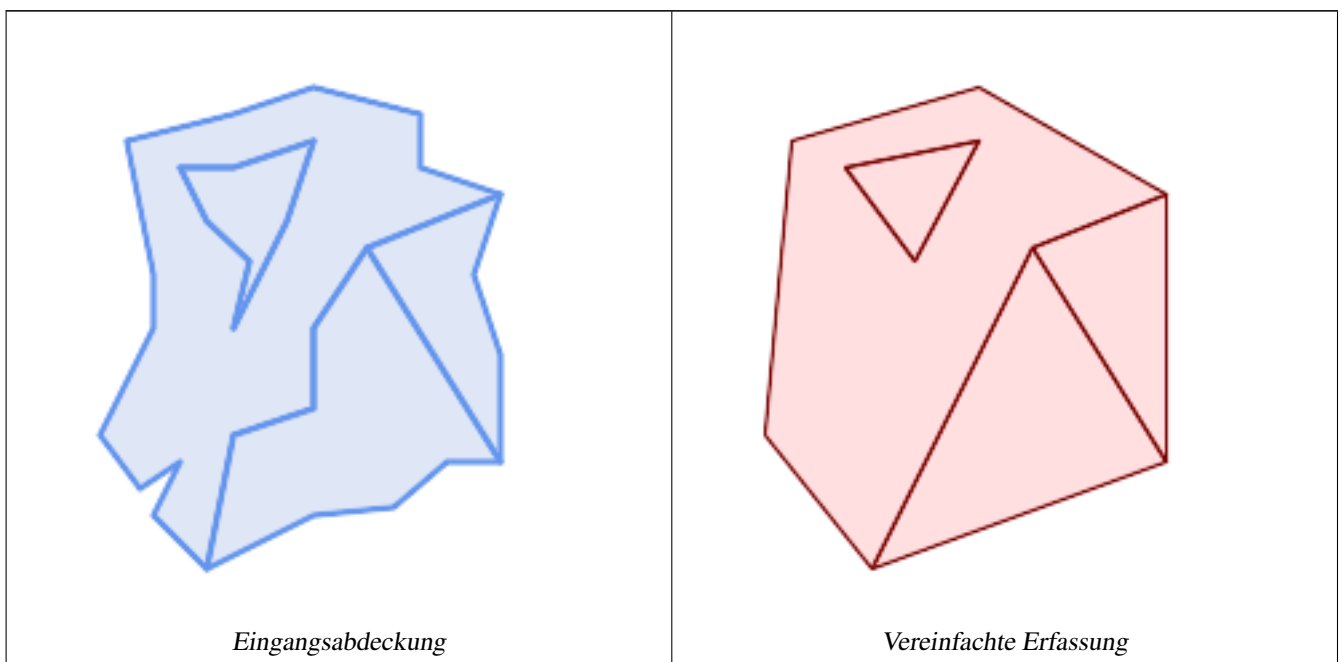
### Note

Wenn die Eingabe keine gültige Abdeckung ist, kann es zu unerwarteten Artefakten in der Ausgabe kommen (z. B. Überschneidungen von Grenzen oder getrennte Grenzen, die gemeinsam zu sein schienen). Verwenden Sie **ST\_CoverageInvalidEdges**, um festzustellen, ob eine Abdeckung gültig ist.

Verfügbarkeit: 3.4.0

Benötigt GEOS >= 3.12.0

## Beispiele



```
WITH coverage(id, geom) AS (VALUES
  (1, 'POLYGON ((160 150, 110 130, 90 100, 90 70, 60 60, 50 10, 30 30, 40 50, 25 40, 10 60, ↵
    30 100, 30 120, 20 170, 60 180, 90 190, 130 180, 130 160, 160 150), (40 160, 50 140, ↵
    66 125, 60 100, 80 140, 90 170, 60 160, 40 160))'::geometry),
  (2, 'POLYGON ((40 160, 60 160, 90 170, 80 140, 60 100, 66 125, 50 140, 40 160))':: ↵
    geometry),
  (3, 'POLYGON ((110 130, 160 50, 140 50, 120 33, 90 30, 50 10, 60 60, 90 70, 90 100, 110 ↵
    130))'::geometry),
  (4, 'POLYGON ((160 150, 150 120, 160 90, 160 50, 110 130, 160 150))'::geometry)
)
SELECT id, ST_AsText(ST_CoverageSimplify(geom, 30) OVER ())
```

```
FROM coverage;
```

```
id |          st_astext
---+-----
 1 | POLYGON ((160 150, 110 130, 50 10, 10 60, 20 170, 90 190, 160 150), (40 160, 66 125, ←
    | 90 170, 40 160))
 2 | POLYGON ((40 160, 66 125, 90 170, 40 160))
 3 | POLYGON ((110 130, 160 50, 50 10, 110 130))
 4 | POLYGON ((160 150, 160 50, 110 130, 160 150))
```

## Siehe auch

[ST\\_CoverageInvalidEdges](#)

### 7.15.3 ST\_CoverageUnion

**ST\_CoverageUnion** — Berechnet die Vereinigung einer Menge von Polygonen, die eine Abdeckung bilden, indem gemeinsame Kanten entfernt werden.

#### Synopsis

```
geometry ST_CoverageUnion(geometry set geom);
```

#### Beschreibung

Eine Aggregatfunktion, die einen Satz von Polygonen zu einer polygonalen Abdeckung zusammenfügt. Das Ergebnis ist eine polygonale Geometrie, die denselben Bereich wie die Abdeckung abdeckt. Diese Funktion liefert das gleiche Ergebnis wie [ST\\_Union](#), nutzt aber die Struktur der Abdeckung, um die Vereinigung viel schneller zu berechnen.

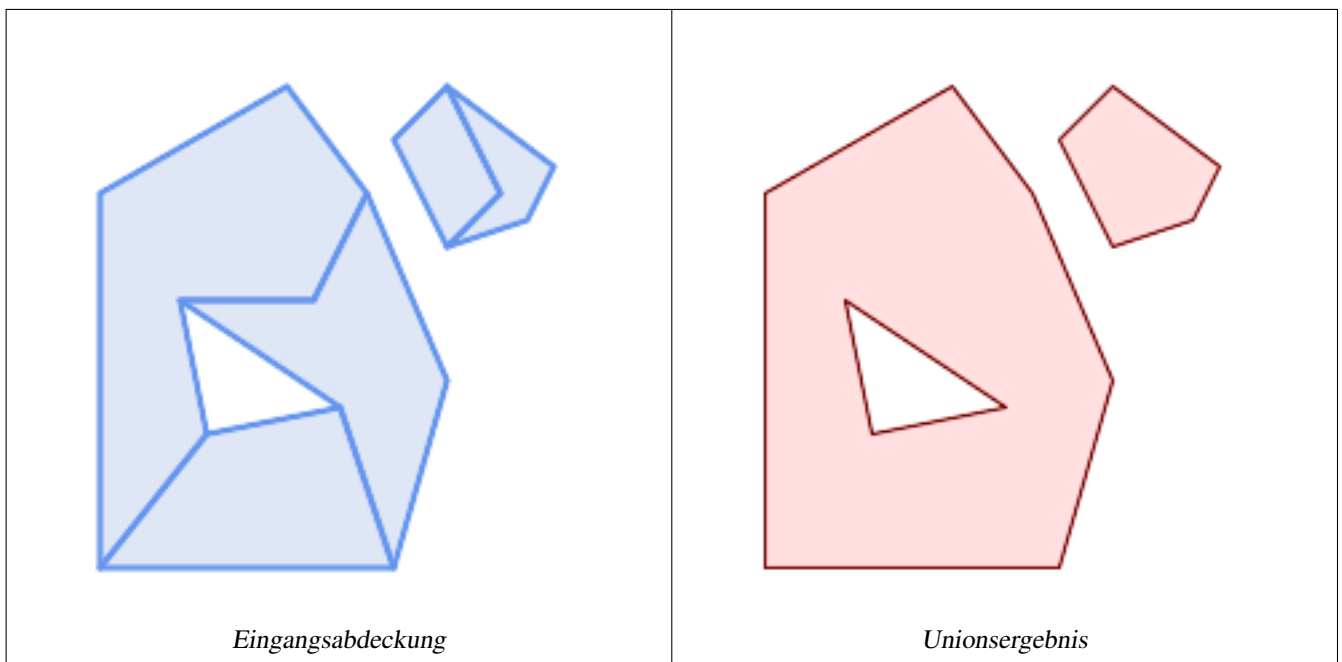


#### Note

Wenn die Eingabe keine gültige Abdeckung ist, kann es zu unerwarteten Artefakten in der Ausgabe kommen (z. B. nicht verschmolzene oder überlappende Polygone). Verwenden Sie [ST\\_CoverageInvalidEdges](#), um festzustellen, ob eine Abdeckung gültig ist.

Verfügbarkeit: 3.4.0 - erfordert GEOS >= 3.8.0

#### Beispiele



```

WITH coverage(id, geom) AS (VALUES
  (1, 'POLYGON ((10 10, 10 150, 80 190, 110 150, 90 110, 40 110, 50 60, 10 10))'::geometry) ←
  /
  (2, 'POLYGON ((120 10, 10 10, 50 60, 100 70, 120 10))'::geometry),
  (3, 'POLYGON ((140 80, 120 10, 100 70, 40 110, 90 110, 110 150, 140 80))'::geometry),
  (4, 'POLYGON ((140 190, 120 170, 140 130, 160 150, 140 190))'::geometry),
  (5, 'POLYGON ((180 160, 170 140, 140 130, 160 150, 140 190, 180 160))'::geometry)
)
SELECT ST_AsText(ST_CoverageUnion(geom))
FROM coverage;
-----
MULTIPOLYGON (((10 150, 80 190, 110 150, 140 80, 120 10, 10 10, 10 150), (50 60, 100 70, 40 ←
  110, 50 60)), ((120 170, 140 190, 180 160, 170 140, 140 130, 120 170)))

```

### Siehe auch

[ST\\_CoverageInvalidEdges](#), [ST\\_AsBinary](#)

## 7.16 Affine Transformationen

### 7.16.1 ST\_Affine

`ST_Affine` — Wenden Sie eine affine 3D-Transformation auf eine Geometrie an.

#### Synopsis

`geometry` **ST\_Affine**(`geometry geomA`, `float a`, `float b`, `float c`, `float d`, `float e`, `float f`, `float g`, `float h`, `float i`, `float xoff`, `float yoff`, `float zoff`);

`geometry` **ST\_Affine**(`geometry geomA`, `float a`, `float b`, `float d`, `float e`, `float xoff`, `float yoff`);

**Beschreibung**

Wendet eine affine 3D-Transformation auf die Geometrie an, um Dinge wie Verschieben, Drehen und Skalieren in einem Schritt durchzuführen.

Version 1: Der Aufruf

```
ST_Affine(geom, a, b, c, d, e, f, g, h, i, xoff, yoff, zoff)
```

stellt die Transformationsmatrix

```
/ a b c xoff \  
| d e f yoff |  
| g h i zoff |  
\ 0 0 0 1 /
```

und die Scheitelpunkte 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 nachstehenden Translations-/Skalierungsfunktionen werden durch eine solche affine Transformation ausgedrückt.

Version 2: Wendet eine 2d affine Transformation auf die Geometrie an. Der Aufruf

```
ST_Affine(geom, a, b, d, e, xoff, yoff)
```

stellt die Transformationsmatrix

```
/ a b 0 xoff \  
| d e 0 yoff |  
| 0 0 1 0 | \ 0 0 1 /  
\ 0 0 0 1 /
```

und die Scheitelpunkte werden wie folgt transformiert:

```
x' = a*x + b*y + xoff  
y' = d*x + e*y + yoff  
z' = z
```

Diese Methode ist ein Unterfall der obigen 3D-Methode.

Erweiterung: Mit 2.0.0 wurde die Unterstützung für polyedrische Oberflächen, Dreiecke und TIN eingeführt.

Verfügbarkeit: 1.1.2. Name geändert von Affine zu ST\_Affine in 1.2.2

**Note**

Vor 1.3.4 ist diese Funktion abgestürzt, wenn die Geometrien CURVES enthalten. Dies wurde mit 1.3.4+ behoben



Diese Funktion unterstützt polyedrische Flächen.



Diese Funktion unterstützt Dreiecke und dreieckige unregelmäßige Netzoberflächen (TIN).



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



Diese Methode unterstützt kreisförmige Strings und Kurven.



## Beispiele

```
--Rotate a 3d line 180 degrees about the z axis. Note this is long-hand for doing ←
ST_Rotate();
SELECT ST_AsEWKT(ST_Affine(geom, cos(pi()), -sin(pi()), 0, sin(pi()), cos(pi()), 0, 0, ←
0, 1, 0, 0, 0)) As using_affine,
ST_AsEWKT(ST_Rotate(geom, pi())) As using_rotate
FROM (SELECT ST_GeomFromEWKT('LINESTRING(1 2 3, 1 4 3)') As geom) As foo;
using_affine | using_rotate
-----+-----
LINESTRING(-1 -2 3,-1 -4 3) | LINESTRING(-1 -2 3,-1 -4 3)
(1 row)

--Rotate a 3d line 180 degrees in both the x and z axis
SELECT ST_AsEWKT(ST_Affine(geom, cos(pi()), -sin(pi()), 0, sin(pi()), cos(pi()), -sin(pi()) ←
, 0, sin(pi()), cos(pi()), 0, 0, 0))
FROM (SELECT ST_GeomFromEWKT('LINESTRING(1 2 3, 1 4 3)') As geom) As foo;
st_asewkt
-----
LINESTRING(-1 -2 -3,-1 -4 -3)
(1 row)
```

## Siehe auch

[ST\\_Rotate](#), [ST\\_Scale](#), [ST\\_Translate](#), [ST\\_TransScale](#)

## 7.16.2 ST\_Rotate

**ST\_Rotate** — Dreht eine Geometrie um einen Ursprungspunkt.

### Synopsis

```
geometry ST_Rotate(geometry geomA, float rotRadians);
geometry ST_Rotate(geometry geomA, float rotRadians, float x0, float y0);
geometry ST_Rotate(geometry geomA, float rotRadians, geometry pointOrigin);
```

### Beschreibung

Dreht die Geometrie rotRadian gegen den Uhrzeigersinn um den Ursprungspunkt. Der Rotationsursprung kann entweder als POINT-Geometrie oder als x- und y-Koordinaten angegeben werden. Wenn der Ursprung nicht angegeben wird, wird die Geometrie um POINT(0 0) gedreht.

Erweiterung: Mit 2.0.0 wurde die Unterstützung für polyedrische Oberflächen, Dreiecke und TIN eingeführt.

Verbessert: In Version 2.0.0 wurden zusätzliche Parameter zur Angabe des Ursprungs der Drehung hinzugefügt.

Verfügbarkeit: 1.1.2. Name geändert von Rotate zu ST\_Rotate in 1.2.2



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



Diese Methode unterstützt kreisförmige Strings und Kurven.



Diese Funktion unterstützt polyedrische Flächen.



Diese Funktion unterstützt Dreiecke und dreieckige unregelmäßige Netzoberflächen (TIN).

## Beispiele

```
--Rotate 180 degrees
SELECT ST_AsEWKT(ST_Rotate('LINESTRING (50 160, 50 50, 100 50)', pi()));
           st_asewkt
-----
LINESTRING(-50 -160,-50 -50,-100 -50)
(1 row)

--Rotate 30 degrees counter-clockwise at x=50, y=160
SELECT ST_AsEWKT(ST_Rotate('LINESTRING (50 160, 50 50, 100 50)', pi()/6, 50, 160));
           st_asewkt
-----
LINESTRING(50 160,105 64.7372055837117,148.301270189222 89.7372055837117)
(1 row)

--Rotate 60 degrees clockwise from centroid
SELECT ST_AsEWKT(ST_Rotate(geom, -pi()/3, ST_Centroid(geom)))
FROM (SELECT 'LINESTRING (50 160, 50 50, 100 50)::geometry AS geom) AS foo;
           st_asewkt
-----
LINESTRING(116.4225 130.6721,21.1597 75.6721,46.1597 32.3708)
(1 row)
```

## Siehe auch

[ST\\_Affine](#), [ST\\_RotateX](#), [ST\\_RotateY](#), [ST\\_RotateZ](#)

### 7.16.3 ST\_RotateX

**ST\_RotateX** — Dreht eine Geometrie um die X-Achse.

#### Synopsis

```
geometry ST_RotateX(geometry geomA, float rotRadians);
```

#### Beschreibung

Dreht eine Geometrie geomA - rotRadians um die X-Achse.



#### Note

`ST_RotateX(geomA, rotRadians)` ist die Kurzform für `ST_Affine(geomA, 1, 0, 0, 0, cos(rotRadians), -sin(rotRadians), 0, sin(rotRadians), cos(rotRadians), 0, 0, 0, 0)`.

Erweiterung: Mit 2.0.0 wurde die Unterstützung für polyedrische Oberflächen, Dreiecke und TIN eingeführt.

Verfügbarkeit: 1.1.2. Name geändert von `RotateX` zu `ST_RotateX` in 1.2.2



Diese Funktion unterstützt polyedrische Flächen.



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



Diese Funktion unterstützt Dreiecke und dreieckige unregelmäßige Netzoberflächen (TIN).

**Beispiele**

```
--Rotate a line 90 degrees along x-axis
SELECT ST_AsEWKT(ST_RotateX(ST_GeomFromEWKT('LINESTRING(1 2 3, 1 1 1)'), pi()/2));
           st_asewkt
-----
LINESTRING(1 -3 2,1 -1 1)
```

**Siehe auch**

[ST\\_Affine](#), [ST\\_RotateY](#), [ST\\_RotateZ](#)

**7.16.4 ST\_RotateY**

ST\_RotateY — Dreht eine Geometrie um die Y-Achse.

**Synopsis**

geometry **ST\_RotateY**(geometry geomA, float rotRadians);

**Beschreibung**

Dreht eine Geometrie geomA - rotRadians um die y-Achse.

**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, 0).

Verfügbarkeit: 1.1.2. Name geändert von RotateY zu ST\_RotateY in 1.2.2

Erweiterung: Mit 2.0.0 wurde die Unterstützung für polyedrische Oberflächen, Dreiecke und TIN eingeführt.



Diese Funktion unterstützt polyedrische Flächen.



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



Diese Funktion unterstützt Dreiecke und dreieckige unregelmäßige Netzoberflächen (TIN).

**Beispiele**

```
--Rotate a line 90 degrees along y-axis
SELECT ST_AsEWKT(ST_RotateY(ST_GeomFromEWKT('LINESTRING(1 2 3, 1 1 1)'), pi()/2));
           st_asewkt
-----
LINESTRING(3 2 -1,1 1 -1)
```

**Siehe auch**

[ST\\_Affine](#), [ST\\_RotateY](#), [ST\\_RotateZ](#)

## 7.16.5 ST\_RotateZ

ST\_RotateZ — Dreht eine Geometrie um die Z-Achse.

### Synopsis

geometry **ST\_RotateZ**(geometry geomA, float rotRadians);

### Beschreibung

Dreht eine Geometrie geomA - rotRadian um die Z-Achse.



#### 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, 0).

Erweiterung: Mit 2.0.0 wurde die Unterstützung für polyedrische Oberflächen, Dreiecke und TIN eingeführt.

Verfügbarkeit: 1.1.2. Name geändert von RotateZ zu ST\_RotateZ in 1.2.2



#### Note

Vor 1.3.4 ist diese Funktion abgestürzt, wenn die Geometrien CURVES enthalten. Dies wurde mit 1.3.4+ behoben



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



Diese Methode unterstützt kreisförmige Strings und Kurven.



Diese Funktion unterstützt polyedrische Flächen.



Diese Funktion unterstützt Dreiecke und dreieckige unregelmäßige Netzoberflächen (TIN).

### Beispiele

```
--Rotate a line 90 degrees along z-axis
SELECT ST_AsEWKT(ST_RotateZ(ST_GeomFromEWKT('LINESTRING(1 2 3, 1 1 1)'), pi()/2));
           st_asewkt
-----
LINESTRING(-2 1 3,-1 1 1)

--Rotate a curved circle around z-axis
SELECT ST_AsEWKT(ST_RotateZ(geom, pi()/2))
FROM (SELECT ST_LineToCurve(ST_Buffer(ST_GeomFromText('POINT(234 567)'), 3)) As geom) As foo;
```

```
CURVEPOLYGON(CIRCULARSTRING(-567 237,-564.87867965644 236.12132034356,-564 234,-569.12132034356 231.87867965644,-567 237))
```

### Siehe auch

[ST\\_Affine](#), [ST\\_RotateX](#), [ST\\_RotateY](#)

## 7.16.6 ST\_Scale

ST\_Scale — Skaliert eine Geometrie um bestimmte Faktoren.

### Synopsis

```
geometry ST_Scale(geometry geomA, float XFactor, float YFactor, float ZFactor);
geometry ST_Scale(geometry geomA, float XFactor, float YFactor);
geometry ST_Scale(geometry geom, geometry factor);
geometry ST_Scale(geometry geom, geometry factor, geometry origin);
```

### Beschreibung

Skaliert die Geometrie auf eine neue Größe, indem die Ordinaten mit den entsprechenden Faktorparametern multipliziert werden. Die Version, die eine Geometrie als `factor` Parameter nimmt, erlaubt die Übergabe eines 2d, 3dm, 3dz oder 4d Punktes, um den Skalierungsfaktor für alle unterstützten Dimensionen zu setzen. Fehlende Dimensionen im `factor` Punkt sind gleichbedeutend mit keiner Skalierung der entsprechenden Dimension.

Bei der Variante mit drei Geometrien kann ein "falscher Ursprung" für die Skalierung übergeben werden. Dies ermöglicht eine "Skalierung an Ort und Stelle", z. B. unter Verwendung des Schwerpunkts der Geometrie als falscher Ursprung. Ohne einen falschen Ursprung erfolgt die Skalierung relativ zum tatsächlichen Ursprung, so dass alle Koordinaten einfach mit dem Skalierungsfaktor multipliziert werden.



#### 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.

Verbessert: In Version 2.2.0 wurde die Unterstützung für die Skalierung aller Dimensionen (Parameter `factor`) eingeführt.

Verbessert: In Version 2.5.0 wurde die Unterstützung für die Skalierung relativ zu einem lokalen Ursprung (Parameter `origin`) eingeführt.



Diese Funktion unterstützt polyedrische Flächen.



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



Diese Methode unterstützt kreisförmige Strings und Kurven.



Diese Funktion unterstützt Dreiecke und dreieckige unregelmäßige Netzoberflächen (TIN).



Diese Funktion unterstützt M-Koordinaten.

## Beispiele

```
--Version 1: scale X, Y, Z
SELECT ST_AsEWKT(ST_Scale(ST_GeomFromEWKT('LINESTRING(1 2 3, 1 1 1)'), 0.5, 0.75, 0.8));
          st_asewkt
-----
LINESTRING(0.5 1.5 2.4,0.5 0.75 0.8)

--Version 2: Scale X Y
SELECT ST_AsEWKT(ST_Scale(ST_GeomFromEWKT('LINESTRING(1 2 3, 1 1 1)'), 0.5, 0.75));
          st_asewkt
-----
LINESTRING(0.5 1.5 3,0.5 0.75 1)

--Version 3: Scale X Y Z M
SELECT ST_AsEWKT(ST_Scale(ST_GeomFromEWKT('LINESTRING(1 2 3 4, 1 1 1 1)'),
  ST_MakePoint(0.5, 0.75, 2, -1)));
          st_asewkt
-----
LINESTRING(0.5 1.5 6 -4,0.5 0.75 2 -1)

--Version 4: Scale X Y using false origin
SELECT ST_AsText(ST_Scale('LINESTRING(1 1, 2 2)', 'POINT(2 2)', 'POINT(1 1)::geometry));
          st_astext
-----
LINESTRING(1 1,3 3)
```

## Siehe auch

[ST\\_Affine](#), [ST\\_TransScale](#)

### 7.16.7 ST\_Translate

ST\_Translate — Verschiebt eine Geometrie um vorgegebene Offsets.

#### Synopsis

```
geometry ST_Translate(geometry g1, float deltax, float deltay);
geometry ST_Translate(geometry g1, float deltax, float deltay, float deltaz);
```

#### Beschreibung

Gibt eine neue Geometrie zurück, deren Koordinaten in den Einheiten delta x,delta y,delta z übersetzt sind. Die Einheiten basieren auf den im Raumbezug (SRID) für diese Geometrie definierten Einheiten.



#### 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



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



Diese Methode unterstützt kreisförmige Strings und Kurven.

## 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)
```

### Verschieben eines Linienzugs um 1 Grad Länge und 1/2 Grad Breite

```
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)
```

### Verschieben eines 3D-Punktes

```
SELECT ST_AsEWKT(ST_Translate(CAST('POINT(0 0 0)' As geometry), 5, 12,3));
    st_asewkt
    -----
    POINT(5 12 3)
```

### Verschieben einer Kurve und eines Punktes

```
SELECT ST_AsText(ST_Translate(ST_Collect('CURVEPOLYGON(CIRCULARSTRING(4 3,3.12 0.878,1 ↵
    0,-1.121 5.1213,6 7, 8 9,4 3))','POINT(1 3)'),1,2));

-----

GEOMETRYCOLLECTION(CURVEPOLYGON(CIRCULARSTRING(5 5,4.12 2.878,2 2,-0.121 7.1213,7 9,9 11,5 ↵
    5)),POINT(2 5))
```

## Siehe auch

[ST\\_Affine](#), [ST\\_AsText](#), [ST\\_GeomFromText](#)

## 7.16.8 ST\_TransScale

`ST_TransScale` — Verschiebt und skaliert eine Geometrie mit vorgegebenen Offsets und Faktoren.

### Synopsis

```
geometry ST_TransScale(geometry geomA, float deltaX, float deltaY, float XFactor, float YFactor);
```

### Beschreibung

Verschiebt die Geometrie mit Hilfe der Args `deltaX` und `deltaY` und skaliert sie dann mit Hilfe der Args `XFactor` und `YFactor`, wobei nur in 2D gearbeitet wird.

**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.



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



Diese Methode unterstützt kreisförmige Strings und Kurven.

**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)

--Buffer a point to get an approximation of a circle, convert to curve and then translate ←
1,2 and scale it 3,4
SELECT ST_AsText(ST_Transscale(ST_LineToCurve(ST_Buffer('POINT(234 567)', 3)),1,2,3,4));

-----

CURVEPOLYGON(CIRCULARSTRING(714 2276,711.363961030679 2267.51471862576,705 ←
2264,698.636038969321 2284.48528137424,714 2276))
```

**Siehe auch**

[ST\\_Affine](#), [ST\\_Translate](#)

## 7.17 Clustering-Funktionen

### 7.17.1 ST\_ClusterDBSCAN

`ST_ClusterDBSCAN` — Fensterfunktion, die eine Cluster-ID für jede Eingabegeometrie unter Verwendung des DBSCAN-Algorithmus zurückgibt.

**Synopsis**

integer `ST_ClusterDBSCAN`(geometry winset geom, float8 eps, integer minpoints);



## Beschreibung

Eine Fensterfunktion, die eine Clusternummer für jede Eingabegeometrie zurückgibt, unter Verwendung des 2D **Dichte-basierten räumlichen Clustering von Anwendungen mit Rauschen (DBSCAN)** Algorithmus. Im Gegensatz zu **ST\_ClusterKMeans** muss die Anzahl der Cluster nicht angegeben werden, sondern es wird der gewünschte **Distanz** (`eps`) und Dichte (`minpoints`) zur Bestimmung der einzelnen Cluster.

Eine Eingangsgeometrie wird zu einem Cluster hinzugefügt, wenn sie entweder:

- Eine "Kern"-Geometrie, die innerhalb `eps` **Abstand** von mindestens `minpoints` Eingabegeometrien (einschließlich sich selbst) liegt; oder
- Eine "Rand"-Geometrie, die sich in `eps` **Entfernung** einer Kerngeometrie befindet.

Beachten Sie, dass Randgeometrien in `eps` Entfernung von Kerngeometrien in mehr als einem Cluster liegen können. Jede der beiden Zuordnungen wäre korrekt, so dass die Randgeometrie willkürlich einem der verfügbaren Cluster zugeordnet wird. In dieser Situation ist es möglich, dass ein korrekter Cluster mit weniger als `minpoints` Geometrien erzeugt wird. Um eine deterministische Zuordnung der Randgeometrien zu gewährleisten (so dass wiederholte Aufrufe von `ST_ClusterDBSCAN` identische Ergebnisse liefern), verwenden Sie eine `ORDER BY` Klausel in der Fensterdefinition. Mehrdeutige Clusterzuweisungen können sich von anderen DBSCAN-Implementierungen unterscheiden.



### Note

Geometrien, die die Kriterien für die Zugehörigkeit zu einem Cluster nicht erfüllen, erhalten eine Clusternummer von NULL.

---

Verfügbarkeit: 2.3.0



Diese Methode unterstützt kreisförmige Strings und Kurven.

## Beispiele

Clustering von Polygonen in einem Umkreis von 50 Metern, wobei mindestens 2 Polygone pro Cluster erforderlich sind.

---



*Cluster innerhalb von 50 Metern mit mindestens 2 Gegenständen pro Cluster. Singletons haben NULL für cid*

```

SELECT name, ST_ClusterDBSCAN(geom, eps =
> 50, minpoints =
> 2) over () AS cid
FROM boston_polys
WHERE name
> '' AND building
> ''
      AND ST_DWithin(geom,
      ST_Transform(
      ST_GeomFromText('POINT ↵
(-71.04054 42.35141)', 4326), 26986),
      500);
    
```

bucket	name		↵
0	Manulife Tower		↵
0	Park Lane Seaport I		↵
0	Park Lane Seaport II		↵
0	Renaissance Boston Waterfront Hotel		↵
0	Seaport Boston Hotel		↵
0	Seaport Hotel & World Trade Center		↵
0	Waterside Place		↵
0	World Trade Center East		↵
1	100 Northern Avenue		↵
1	100 Pier 4		↵
1	The Institute of Contemporary Art		↵
2	101 Seaport		↵
2	District Hall		↵
2	One Marina Park Drive		↵
2	Twenty Two Liberty		↵
2	Vertex		↵
2	Vertex		↵
2	Watermark Seaport		↵
NULL	Blue Hills Bank Pavilion		↵
NULL	World Trade Center West		↵

(20 rows)

Ein Beispiel für die Zusammenfassung von Flurstücken mit derselben Clusternummer zu geometrischen Sammlungen.

```

SELECT cid, ST_Collect(geom) AS cluster_geom, array_agg(parcel_id) AS ids_in_cluster FROM (
  SELECT parcel_id, ST_ClusterDBSCAN(geom, eps => 0.5, minpoints => 5) over () AS cid, ↵
  geom
  FROM parcels) sq
GROUP BY cid;
    
```

**Siehe auch**

[ST\\_DWithin](#), [ST\\_ClusterKMeans](#), [ST\\_ClusterIntersecting](#), [ST\\_ClusterIntersectingWin](#), [ST\\_ClusterWithin](#), [ST\\_ClusterWithinWin](#)

## 7.17.2 ST\_ClusterIntersecting

ST\_ClusterIntersecting — Aggregatfunktion, die Eingabegeometrien zu zusammenhängenden Mengen clustert.

### Synopsis

```
geometry[] ST_ClusterIntersecting(geometry set g);
```

### Beschreibung

Eine Aggregatfunktion, die ein Array von GeometryCollections zurückgibt, das die Eingabegeometrien in zusammenhängende, disjunkte Cluster partitioniert. Jede Geometrie in einem Cluster schneidet mindestens eine andere Geometrie in diesem Cluster und schneidet keine Geometrie in anderen Clustern.

Verfügbarkeit: 2.2.0

### Beispiele

```
WITH testdata AS
  (SELECT unnest(ARRAY['LINESTRING (0 0, 1 1)::geometry,
    'LINESTRING (5 5, 4 4)::geometry,
    'LINESTRING (6 6, 7 7)::geometry,
    'LINESTRING (0 0, -1 -1)::geometry,
    'POLYGON ((0 0, 4 0, 4 4, 0 4, 0 0))::geometry']) AS geom)

SELECT ST_AsText(unnest(ST_ClusterIntersecting(geom))) FROM testdata;

--result

st_astext
-----
GEOMETRYCOLLECTION(LINESTRING(0 0,1 1),LINESTRING(5 5,4 4),LINESTRING(0 0,-1 -1),POLYGON((0 ←
  0,4 0,4 4,0 4,0 0)))
GEOMETRYCOLLECTION(LINESTRING(6 6,7 7))
```

### Siehe auch

[ST\\_ClusterIntersectingWin](#), [ST\\_ClusterWithin](#), [ST\\_ClusterWithinWin](#)

## 7.17.3 ST\_ClusterIntersectingWin

ST\_ClusterIntersectingWin — Fensterfunktion, die für jede Eingabegeometrie eine Cluster-ID zurückgibt und die Eingabegeometrien in zusammenhängende Gruppen clustert.

### Synopsis

```
integer ST_ClusterIntersectingWin(geometry winset geom);
```

### Beschreibung

Eine Fensterfunktion, die zusammenhängende Cluster von sich schneidenden Geometrien bildet. Es ist möglich, alle Geometrien in einem Cluster zu durchlaufen, ohne den Cluster zu verlassen. Der Rückgabewert ist die Clusternummer, an der das Geometrieargument beteiligt ist, oder null für Nulleingaben.

Verfügbarkeit: 3.4.0

## Beispiele

```
WITH testdata AS (
  SELECT id, geom::geometry FROM (
    VALUES (1, 'LINESTRING (0 0, 1 1)'),
           (2, 'LINESTRING (5 5, 4 4)'),
           (3, 'LINESTRING (6 6, 7 7)'),
           (4, 'LINESTRING (0 0, -1 -1)'),
           (5, 'POLYGON ((0 0, 4 0, 4 4, 0 4, 0 0))')) AS t(id, geom)
)
SELECT id,
       ST_AsText(geom),
       ST_ClusterIntersectingWin(geom) OVER () AS cluster
FROM testdata;
```

id	st_astext	cluster
1	LINESTRING(0 0,1 1)	0
2	LINESTRING(5 5,4 4)	0
3	LINESTRING(6 6,7 7)	1
4	LINESTRING(0 0,-1 -1)	0
5	POLYGON((0 0,4 0,4 4,0 4,0 0))	0

## Siehe auch

[ST\\_ClusterIntersecting](#), [ST\\_ClusterWithin](#), [ST\\_ClusterWithinWin](#)

## 7.17.4 ST\_ClusterKMeans

`ST_ClusterKMeans` — Fensterfunktion, die eine Cluster-ID für jede Eingabegeometrie unter Verwendung des K-Means-Algorithmus zurückgibt.

### Synopsis

```
integer ST_ClusterKMeans( geometry winset geom , integer k , float8 max_radius );
```

### Beschreibung

Gibt **K-means** Clusternummer für jede Eingabegeometrie zurück. Der für das Clustering verwendete Abstand ist der Abstand zwischen den Zentren für 2D-Geometrien und der Abstand zwischen den Bounding-Box-Zentren für 3D-Geometrien. Bei POINT-Eingaben wird die Koordinate M als Gewicht der Eingabe behandelt und muss größer als 0 sein.

`max_radius`, falls gesetzt, veranlasst `ST_ClusterKMeans`, mehr Cluster als `k` zu erzeugen, um sicherzustellen, dass kein Cluster in der Ausgabe einen größeren Radius als `max_radius` hat. Dies ist bei der Erreichbarkeitsanalyse nützlich.

Verbessert: 3.2.0 Unterstützung für `max_radius`

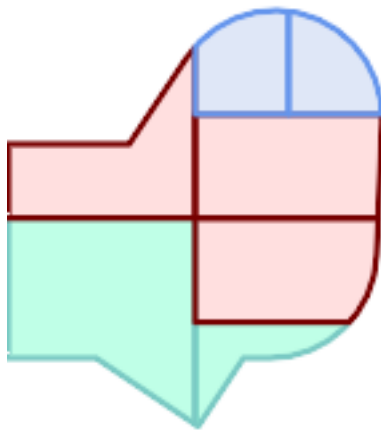
Verbessert: 3.1.0 Unterstützung für 3D-Geometrien und Gewichte

Verfügbarkeit: 2.3.0

### Beispiele

Erzeugen Sie eine Dummy-Parzellengruppe für Beispiele:

```
CREATE TABLE parcels AS
SELECT lpad((row_number() over())::text,3,'0') As parcel_id, geom,
('{residential, commercial}'::text[])[1 + mod(row_number()OVER(),2)] As type
FROM
  ST_Subdivide(ST_Buffer('SRID=3857;LINESTRING(40 100, 98 100, 100 150, 60 90)'::geometry ←
  40, 'endcap=square'),12) As geom;
```



*Parzellen farbcodiert nach Clusternummer (cid)*

```
SELECT ST_ClusterKMeans(geom, 3) OVER() AS cid, parcel_id, geom
FROM parcels;
```

cid	parcel_id	geom
0	001	0103000000...
0	002	0103000000...
1	003	0103000000...
0	004	0103000000...
1	005	0103000000...
2	006	0103000000...
2	007	0103000000...

#### Unterteilung von Parzellenclustern nach Typ:

```
SELECT ST_ClusterKMeans(geom, 3) over (PARTITION BY type) AS cid, parcel_id, type
FROM parcels;
```

cid	parcel_id	type
1	005	commercial
1	003	commercial
2	007	commercial
0	001	commercial
1	004	residential
0	002	residential
2	006	residential

Beispiel: Clustering eines voraggregierten Bevölkerungsdatensatzes auf planetarischer Ebene unter Verwendung von 3D-Clustering und Gewichtung. Identifizierung von mindestens 20 Regionen auf der Grundlage von **Kontur-Bevölkerungsdaten**, die nicht mehr als 3000 km von ihrem Zentrum entfernt sind:

```

create table kontur_population_3000km_clusters as
select
  geom,
  ST_ClusterKMeans(
    ST_Force4D(
      ST_Transform(ST_Force3D(geom), 4978), -- cluster in 3D XYZ CRS
      mvalue => population -- set clustering to be weighed by population
    ),
    20, -- aim to generate at least 20 clusters
    max_radius => 3000000 -- but generate more to make each under 3000 km radius
  ) over () as cid
from
  kontur_population;

```



Die Weltbevölkerung, geclustert nach den obigen Angaben, ergibt 46 Cluster. Die Cluster konzentrieren sich auf gut bevölkerte Regionen (New York, Moskau). Grönland ist ein Cluster. Es gibt Insel-Cluster, die sich über den Antimeridian erstrecken. Die Ränder der Cluster folgen der Erdkrümmung.

#### Siehe auch

[ST\\_ClusterDBSCAN](#), [ST\\_ClusterIntersectingWin](#), [ST\\_ClusterWithinWin](#), [ST\\_ClusterIntersecting](#), [ST\\_ClusterWithin](#), [ST\\_Subdivide](#), [ST\\_Force3D](#), [ST\\_Force4D](#),

### 7.17.5 ST\_ClusterWithin

`ST_ClusterWithin` — Aggregatfunktion, die Geometrien nach Trennungsabstand gruppiert.

#### Synopsis

```
geometry[] ST_ClusterWithin(geometry set g, float8 distance);
```

#### Beschreibung

Eine Aggregatfunktion, die ein Array von GeometryCollections zurückgibt, wobei jede Collection ein Cluster ist, das einige Eingabegeometrien enthält. Bei der Clusterbildung werden die Eingabegeometrien in Gruppen unterteilt, in denen jede Geometrie innerhalb des angegebenen Abstands von mindestens einer anderen Geometrie im selben Cluster liegt. Die Abstände sind kartesische Abstände in den Einheiten des SRID.

`ST_ClusterWithin` ist gleichbedeutend mit der Ausführung von [ST\\_ClusterDBSCAN](#) mit `minpoints => 0`.

Verfügbarkeit: 2.2.0



Diese Methode unterstützt kreisförmige Strings und Kurven.

## Beispiele

```
WITH testdata AS
  (SELECT unnest(ARRAY['LINESTRING (0 0, 1 1)::geometry,
                      'LINESTRING (5 5, 4 4)::geometry,
                      'LINESTRING (6 6, 7 7)::geometry,
                      'LINESTRING (0 0, -1 -1)::geometry,
                      'POLYGON ((0 0, 4 0, 4 4, 0 4, 0 0))::geometry]) AS geom)

SELECT ST_AsText(unnest(ST_ClusterWithin(geom, 1.4))) FROM testdata;

--result

st_astext
-----
GEOMETRYCOLLECTION(LINESTRING(0 0,1 1),LINESTRING(5 5,4 4),LINESTRING(0 0,-1 -1),POLYGON((0 ←
  0,4 0,4 4,0 4,0 0)))
GEOMETRYCOLLECTION(LINESTRING(6 6,7 7))
```

## Siehe auch

[ST\\_ClusterWithinWin](#), [ST\\_ClusterDBSCAN](#), [ST\\_ClusterIntersecting](#), [ST\\_ClusterIntersectingWin](#)

### 7.17.6 ST\_ClusterWithinWin

`ST_ClusterWithinWin` — Fensterfunktion, die eine Cluster-ID für jede Eingabegeometrie zurückgibt, Clustering anhand des Trennungsabstands.

#### Synopsis

integer `ST_ClusterWithinWin`(geometry winset geom, float8 distance);

#### Beschreibung

Eine Fensterfunktion, die für jede Eingabegeometrie eine Clusternummer zurückgibt. Bei der Clusterbildung werden die Geometrien in Gruppen unterteilt, in denen jede Geometrie innerhalb des angegebenen Abstands von mindestens einer anderen Geometrie im selben Cluster liegt. Die Abstände sind kartesische Abstände in den Einheiten des SRID.

`ST_ClusterWithinWin` ist gleichbedeutend mit der Ausführung von `ST_ClusterDBSCAN` mit `minpoints => 0`.

Verfügbarkeit: 3.4.0



Diese Methode unterstützt kreisförmige Strings und Kurven.

## Beispiele

```
WITH testdata AS (
  SELECT id, geom::geometry FROM (
    VALUES (1, 'LINESTRING (0 0, 1 1)'),
           (2, 'LINESTRING (5 5, 4 4)'),
           (3, 'LINESTRING (6 6, 7 7)'),
           (4, 'LINESTRING (0 0, -1 -1)'),
           (5, 'POLYGON ((0 0, 4 0, 4 4, 0 4, 0 0))') AS t(id, geom)
  )
)
SELECT id,
       ST_AsText(geom),
```

```
ST_ClusterWithinWin(geom, 1.4) OVER () AS cluster
FROM testdata;
```

id	st_astext	cluster
1	LINestring(0 0,1 1)	0
2	LINestring(5 5,4 4)	0
3	LINestring(6 6,7 7)	1
4	LINestring(0 0,-1 -1)	0
5	POLYGON((0 0,4 0,4 4,0 4,0 0))	0

### Siehe auch

[ST\\_ClusterWithin](#), [ST\\_ClusterDBSCAN](#), [ST\\_ClusterIntersecting](#), [ST\\_ClusterIntersectingWin](#),

## 7.18 Bounding Box-Funktionen

### 7.18.1 Box2D

Box2D — Gibt ein BOX2D zurück, das die 2D-Ausdehnung einer Geometrie darstellt.

#### Synopsis

```
box2d Box2D(geometry geom);
```

#### Beschreibung

Gibt eine **box2d** zurück, die die 2D-Ausdehnung der Geometrie darstellt.

Erweiterung: Mit 2.0.0 wurde die Unterstützung für polyedrische Oberflächen, Dreiecke und TIN eingeführt.



Diese Methode unterstützt kreisförmige Strings und Kurven.



Diese Funktion unterstützt polyedrische Flächen.



Diese Funktion unterstützt Dreiecke und dreieckige unregelmäßige Netzoberflächen (TIN).

#### Beispiele

```
SELECT Box2D(ST_GeomFromText('LINestring(1 2, 3 4, 5 6)'));
```

```
box2d
-----
BOX(1 2,5 6)
```

```
SELECT Box2D(ST_GeomFromText('CIRCULARSTRING(220268 150415,220227 150505,220227 150406)'));
```

```
box2d
-----
BOX(220186.984375 150406,220288.25 150506.140625)
```



**Siehe auch**[Box3D](#), [ST\\_GeomFromText](#)**7.18.2 Box3D**

Box3D — Gibt ein BOX3D zurück, das die 3D-Ausdehnung einer Geometrie darstellt.

**Synopsis**

```
box3d Box3D(geometry geom);
```

**Beschreibung**

Gibt eine [box3d](#) zurück, die die 3D-Ausdehnung der Geometrie darstellt.

Erweiterung: Mit 2.0.0 wurde die Unterstützung für polyedrische Oberflächen, Dreiecke und TIN eingeführt.



Diese Methode unterstützt kreisförmige Strings und Kurven.



Diese Funktion unterstützt polyedrische Flächen.



Diese Funktion unterstützt Dreiecke und dreieckige unregelmäßige Netzoberflächen (TIN).



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

**Beispiele**

```
SELECT Box3D(ST_GeomFromEWKT('LINESTRING(1 2 3, 3 4 5, 5 6 5)'));
```

```
Box3d
```

```
-----
```

```
BOX3D(1 2 3,5 6 5)
```

```
SELECT Box3D(ST_GeomFromEWKT('CIRCULARSTRING(220268 150415 1,220227 150505 1,220227 150406 <->
1)'));
```

```
Box3d
```

```
-----
```

```
BOX3D(220227 150406 1,220268 150415 1)
```

**Siehe auch**[Box2D](#), [ST\\_GeomFromEWKT](#)**7.18.3 ST\_EstimatedExtent**

ST\_EstimatedExtent — Gibt die geschätzte Ausdehnung einer räumlichen Tabelle zurück.

**Synopsis**

```
box2d ST_EstimatedExtent(text schema_name, text table_name, text geocolumn_name, boolean parent_only);
```

```
box2d ST_EstimatedExtent(text schema_name, text table_name, text geocolumn_name);
```

```
box2d ST_EstimatedExtent(text table_name, text geocolumn_name);
```

## Beschreibung

Gibt die geschätzte Ausdehnung einer räumlichen Tabelle als **box2d** zurück. Falls nicht angegeben, wird das aktuelle Schema verwendet. Die geschätzte Ausdehnung wird aus den Statistiken der Geometriespalte entnommen. Dies ist in der Regel viel schneller als die Berechnung der genauen Ausdehnung der Tabelle mit **ST\_Extent** oder **ST\_3DExtent**.

Standardmäßig werden auch Statistiken aus Kindtabellen (Tabellen mit INHERITS) verwendet, sofern diese verfügbar sind. Wenn `parent_only` auf TRUE gesetzt ist, werden nur Statistiken für die angegebene Tabelle verwendet und untergeordnete Tabellen ignoriert.

Für PostgreSQL  $\geq 8.0.0$  werden die Statistiken durch VACUUM ANALYZE erfasst, und der Ergebnisumfang entspricht etwa 95% des tatsächlichen Umfangs. Für PostgreSQL  $< 8.0.0$  werden die Statistiken durch Ausführen von `update_geometry_stats()` erfasst und der Ergebnisumfang ist exakt.



### Note

Wenn keine Statistiken vorhanden sind (leere Tabelle oder kein ANALYZE-Aufruf), gibt diese Funktion NULL zurück. Vor Version 1.5.4 wurde stattdessen eine Ausnahme ausgelöst.



### Note

Escaping names for tables and/or namespaces that include special characters and quotes may require special handling. A user notes: "For schemas and tables, use identifier escaping rules to produce a double-quoted string, and afterwards remove the first and last double-quote character. For geometry column pass as is."

Verfügbarkeit: 1.0.0

Geändert: 2.1.0. Bis zu 2.0.x hieß dies `ST_Estimated_Extent`.



Diese Methode unterstützt kreisförmige Strings und Kurven.

## Beispiele

```
SELECT ST_EstimatedExtent('ny', 'edges', 'geom');
--result--
BOX(-8877653 4912316,-8010225.5 5589284)

SELECT ST_EstimatedExtent('feature_poly', 'geom');
--result--
BOX(-124.659652709961 24.6830825805664,-67.7798080444336 49.0012092590332)
```

## Siehe auch

[ST\\_Extent](#), [ST\\_GeomFromText](#)

## 7.18.4 ST\_Expand

`ST_Expand` — Gibt einen Begrenzungsrahmen zurück, der aus einem anderen Begrenzungsrahmen oder einer Geometrie erweitert wurde.

## Synopsis

```
geometry ST_Expand(geometry geom, float units_to_expand);
geometry ST_Expand(geometry geom, float dx, float dy, float dz=0, float dm=0);
box2d ST_Expand(box2d box, float units_to_expand);
box2d ST_Expand(box2d box, float dx, float dy);
box3d ST_Expand(box3d box, float units_to_expand);
box3d ST_Expand(box3d box, float dx, float dy, float dz=0);
```

## Beschreibung

Gibt einen Begrenzungsrahmen zurück, der aus dem Begrenzungsrahmen der Eingabe expandiert wurde, indem entweder ein einzelner Abstand angegeben wird, mit dem der Rahmen auf beiden Achsen expandiert werden soll, oder indem ein Expansionsabstand für jede Achse angegeben wird. Verwendet Doppelpräzision. Kann für Abstandsabfragen verwendet werden oder um einen Bounding-Box-Filter zu einer Abfrage hinzuzufügen, um einen räumlichen Index zu nutzen.

Zusätzlich zu der Version von `ST_Expand`, die eine Geometrie akzeptiert und zurückgibt, werden Varianten bereitgestellt, die die Datentypen `box2d` und `box3d` akzeptieren und zurückgeben.

Die Entfernungen sind in den Einheiten des räumlichen Bezugssystems der Eingabe angegeben.

`ST_Expand` ist ähnlich wie `ST_Buffer`, mit dem Unterschied, dass `ST_Expand` das Begrenzungsrechteck entlang jeder Achse erweitert, während Pufferung eine Geometrie in alle Richtungen erweitert.



### Note

Vor Version 1.3 wurde `ST_Expand` in Verbindung mit `ST_Distance` verwendet, um indizierbare Abstandsabfragen durchzuführen. Zum Beispiel: `geom && ST_Expand('POINT(10 20)', 10) AND ST_Distance(geom, 'POINT(10 20)') < 10`. Diese Funktion wurde durch die einfachere und effizientere Funktion `ST_DWithin` ersetzt.

Verfügbarkeit: 1.5.0 Verhalten geändert, um double precision statt float4 Koordinaten auszugeben.

Erweiterung: Mit 2.0.0 wurde die Unterstützung für polyedrische Oberflächen, Dreiecke und TIN eingeführt.

Verbessert: In Version 2.3.0 wurde die Möglichkeit hinzugefügt, eine Box um unterschiedliche Beträge in verschiedenen Dimensionen zu erweitern.



Diese Funktion unterstützt polyedrische Flächen.



Diese Funktion unterstützt Dreiecke und dreieckige unregelmäßige Netzoberflächen (TIN).

## Beispiele



### Note

Die folgenden Beispiele verwenden den US National Atlas Equal Area (SRID=2163), der eine Meterprojektion ist

```
--10 meter expanded box around bbox of a linestring
SELECT CAST(ST_Expand(ST_GeomFromText('LINESTRING(2312980 110676,2312923 110701,2312892 110714)', 2163),10) As box2d);
                                st_expand
-----
BOX(2312882 110666,2312990 110724)

--10 meter expanded 3D box of a 3D box
```

```

SELECT ST_Expand(CAST('BOX3D(778783 2951741 1,794875 2970042.61545891 10)' As box3d),10)
                                st_expand
-----
BOX3D(778773 2951731 -9,794885 2970052.61545891 20)

--10 meter geometry astext rep of a expand box around a point geometry
SELECT ST_AsEWKT(ST_Expand(ST_GeomFromEWKT('SRID=2163;POINT(2312980 110676)'),10));
                                st_asewkt ↔
-----
SRID=2163;POLYGON((2312970 110666,2312970 110686,2312990 110686,2312990 110666,2312970 110666))
                                ↔

```

**Siehe auch**

[ST\\_Buffer](#), [ST\\_DWithin](#), [ST\\_SRID](#)

**7.18.5 ST\_Extent**

`ST_Extent` — Aggregatfunktion, die den Begrenzungsrahmen von Geometrien zurückgibt.

**Synopsis**

`box2d ST_Extent(geometry set geomfield);`

**Beschreibung**

Eine aggregierte Funktion, die eine `box2d` Bounding Box zurückgibt, die einen Satz von Geometrien begrenzt.

Die Bounding-Box-Koordinaten sind im räumlichen Bezugssystem der Eingabegeometrien.

`ST_Extent` ist vom Konzept her ähnlich dem `SDO_AGGR_MBR` von Oracle Spatial/Locator.

**Note**

`ST_Extent` gibt auch bei 3D-Geometrien nur Boxen mit X- und Y-Ordinaten zurück. Um XYZ-Ordinaten zurückzugeben, verwenden Sie [ST\\_3DExtent](#).

**Note**

Der zurückgegebene Wert `box3d` enthält keinen SRID. Verwenden Sie [ST\\_SetSRID](#), um ihn in eine Geometrie mit SRID-Metadaten umzuwandeln. Der SRID ist derselbe wie bei den Eingabegeometrien.

Erweiterung: Mit 2.0.0 wurde die Unterstützung für polyedrische Oberflächen, Dreiecke und TIN eingeführt.



Diese Funktion unterstützt polyedrische Flächen.



Diese Funktion unterstützt Dreiecke und dreieckige unregelmäßige Netzoberflächen (TIN).

## Beispiele



### Note

Die folgenden Beispiele verwenden Massachusetts State Plane ft (SRID=2249)

```
SELECT ST_Extent(geom) as bextent FROM sometable;
                st_bextent
-----
BOX(739651.875 2908247.25,794875.8125 2970042.75)

--Return extent of each category of geometries
SELECT ST_Extent(geom) as bextent
FROM sometable
GROUP BY category ORDER BY category;

                bextent | name
-----+-----
BOX(778783.5625 2951741.25,794875.8125 2970042.75) | A
BOX(751315.8125 2919164.75,765202.6875 2935417.25) | B
BOX(739651.875 2917394.75,756688.375 2935866)      | C

--Force back into a geometry
-- and render the extended text representation of that geometry
SELECT ST_SetSRID(ST_Extent(geom),2249) as bextent FROM sometable;

                bextent
-----
SRID=2249;POLYGON((739651.875 2908247.25,739651.875 2970042.75,794875.8125 2970042.75,
794875.8125 2908247.25,739651.875 2908247.25))
```

## Siehe auch

[ST\\_EstimatedExtent](#), [ST\\_3DExtent](#), [ST\\_SetSRID](#)

### 7.18.6 ST\_3DExtent

`ST_3DExtent` — Aggregatfunktion, die den 3D-Begrenzungsrahmen von Geometrien zurückgibt.

#### Synopsis

```
box3d ST_3DExtent(geometry set geomfield);
```

#### Beschreibung

Eine Aggregatfunktion, die eine `box3d` (einschließlich Z-Ordinate) Bounding Box zurückgibt, die einen Satz von Geometrien begrenzt.

Die Bounding-Box-Koordinaten sind im räumlichen Bezugssystem der Eingabegeometrien.



### Note

Der zurückgegebene Wert `box3d` enthält keinen SRID. Verwenden Sie [ST\\_SetSRID](#), um ihn in eine Geometrie mit SRID-Metadaten umzuwandeln. Der SRID ist derselbe wie bei den Eingabegeometrien.

Erweiterung: Mit 2.0.0 wurde die Unterstützung für polyedrische Oberflächen, Dreiecke und TIN eingeführt.

Geändert: 2.0.0 In früheren Versionen wurde dies als ST\_Extent3D bezeichnet.



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



Diese Methode unterstützt kreisförmige Strings und Kurven.



Diese Funktion unterstützt polyedrische Flächen.



Diese Funktion unterstützt Dreiecke und dreieckige unregelmäßige Netzoberflächen (TIN).

## Beispiele

```
SELECT ST_3DExtent(foo.geom) As b3extent
FROM (SELECT ST_MakePoint(x,y,z) As geom
      FROM generate_series(1,3) As x
      CROSS JOIN generate_series(1,2) As y
      CROSS JOIN generate_series(0,2) As Z) As foo;
-----
b3extent
-----
BOX3D(1 1 0,3 2 2)

--Get the extent of various elevated circular strings
SELECT ST_3DExtent(foo.geom) As b3extent
FROM (SELECT ST_Translate(ST_Force_3DZ(ST_LineToCurve(ST_Buffer(ST_Point(x,y),1))),0,0,z) ←
      As geom
      FROM generate_series(1,3) As x
      CROSS JOIN generate_series(1,2) As y
      CROSS JOIN generate_series(0,2) As Z) As foo;
-----
b3extent
-----
BOX3D(1 0 0,4 2 2)
```

## Siehe auch

[ST\\_Extent](#), [ST\\_Force3DZ](#), [ST\\_SetSRID](#)

### 7.18.7 ST\_MakeBox2D

ST\_MakeBox2D — Erzeugt ein BOX2D, das durch zwei 2D-Punktgeometrien definiert ist.

#### Synopsis

```
box2d ST_MakeBox2D(geometry pointLowLeft, geometry pointUpRight);
```

#### Beschreibung

Erzeugt eine **box2d**, die durch zwei Punktgeometrien definiert ist. Dies ist nützlich für Bereichsabfragen.

## Beispiele

```
--Return all features that fall reside or partly reside in a US national atlas coordinate ←
  bounding box
--It is assumed here that the geometries are stored with SRID = 2163 (US National atlas ←
  equal area)
SELECT feature_id, feature_name, geom
FROM features
WHERE geom && ST_SetSRID(ST_MakeBox2D(ST_Point(-989502.1875, 528439.5625),
  ST_Point(-987121.375 ,529933.1875)),2163)
```

## Siehe auch

[ST\\_Point](#), [ST\\_SetSRID](#), [ST\\_SRID](#)

## 7.18.8 ST\_3DMakeBox

ST\_3DMakeBox — Erzeugt einen BOX3D, der durch zwei 3D-Punktgeometrien definiert ist.

### Synopsis

box3d ST\_3DMakeBox(geometry point3DLowLeftBottom, geometry point3DUpRightTop);

### Beschreibung

Erzeugt eine **box3d**, die durch zwei 3D-Punktgeometrien definiert ist.



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

Geändert: 2.0.0 In früheren Versionen hieß diese Funktion ST\_MakeBox3D

### Beispiele

```
SELECT ST_3DMakeBox(ST_MakePoint(-989502.1875, 528439.5625, 10),
  ST_MakePoint(-987121.375 ,529933.1875, 10)) As abb3d

--bb3d--
-----
BOX3D(-989502.1875 528439.5625 10,-987121.375 529933.1875 10)
```

## Siehe auch

[ST\\_MakePoint](#), [ST\\_SetSRID](#), [ST\\_SRID](#)

## 7.18.9 ST\_XMax

ST\_XMax — Gibt die X-Maxima eines 2D- oder 3D-Begrenzungsrahmens oder einer Geometrie zurück.

### Synopsis

float ST\_XMax(box3d aGeomorBox2DorBox3D);

## Beschreibung

Gibt die X-Maxima eines 2D- oder 3D-Begrenzungsrahmens oder einer Geometrie zurück.



### Note

Obwohl diese Funktion nur für `box3d` definiert ist, funktioniert sie aufgrund des automatischen Castings auch für `box2d`- und Geometriewerte. Sie akzeptiert jedoch keine Geometrie- oder `box2d`-Textdarstellung, da diese nicht automatisch gecastet werden.



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



Diese Methode unterstützt kreisförmige Strings und Kurven.

## Beispiele

```
SELECT ST_XMax('BOX3D(1 2 3, 4 5 6)');
st_xmax
-----
4

SELECT ST_XMax(ST_GeomFromText('LINESTRING(1 3 4, 5 6 7)'));
st_xmax
-----
5

SELECT ST_XMax(CAST('BOX(-3 2, 3 4)' As box2d));
st_xmax
-----
3
--Observe THIS DOES NOT WORK because it will try to auto-cast the string representation to ←
a BOX3D
SELECT ST_XMax('LINESTRING(1 3, 5 6)');

--ERROR: BOX3D parser - doesn't start with BOX3D(

SELECT ST_XMax(ST_GeomFromEWKT('CIRCULARSTRING(220268 150415 1,220227 150505 2,220227 ←
150406 3)'));
st_xmax
-----
220288.248780547
```

## Siehe auch

[ST\\_XMin](#), [ST\\_YMax](#), [ST\\_YMin](#), [ST\\_ZMax](#), [ST\\_ZMin](#)

### 7.18.10 ST\_XMin

`ST_XMin` — Gibt die X-Minima eines 2D- oder 3D-Begrenzungsrahmens oder einer Geometrie zurück.

## Synopsis

```
float ST_XMin(box3d aGeomorBox2DorBox3D);
```



## Beschreibung

Gibt die X-Minima eines 2D- oder 3D-Begrenzungsrahmens oder einer Geometrie zurück.



### Note

Obwohl diese Funktion nur für `box3d` definiert ist, funktioniert sie aufgrund des automatischen Castings auch für `box2d`- und Geometriewerte. Sie akzeptiert jedoch keine Geometrie- oder `box2d`-Textdarstellung, da diese nicht automatisch gecastet werden.



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



Diese Methode unterstützt kreisförmige Strings und Kurven.

## Beispiele

```
SELECT ST_XMin('BOX3D(1 2 3, 4 5 6)');
st_xmin
-----
1

SELECT ST_XMin(ST_GeomFromText('LINESTRING(1 3 4, 5 6 7)'));
st_xmin
-----
1

SELECT ST_XMin(CAST('BOX(-3 2, 3 4)' As box2d));
st_xmin
-----
-3
--Observe THIS DOES NOT WORK because it will try to auto-cast the string representation to ←
a BOX3D
SELECT ST_XMin('LINESTRING(1 3, 5 6)');

--ERROR: BOX3D parser - doesn't start with BOX3D(

SELECT ST_XMin(ST_GeomFromEWKT('CIRCULARSTRING(220268 150415 1,220227 150505 2,220227 ←
150406 3)'));
st_xmin
-----
220186.995121892
```

## Siehe auch

[ST\\_XMax](#), [ST\\_YMax](#), [ST\\_YMin](#), [ST\\_ZMax](#), [ST\\_ZMin](#)

### 7.18.11 ST\_YMax

`ST_YMax` — Gibt die Y-Maxima eines 2D- oder 3D-Begrenzungsrahmens oder einer Geometrie zurück.

## Synopsis

```
float ST_YMax(box3d aGeomorBox2DorBox3D);
```

## Beschreibung

Gibt die Y-Maxima eines 2D- oder 3D-Begrenzungsrahmens oder einer Geometrie zurück.



### Note

Obwohl diese Funktion nur für `box3d` definiert ist, funktioniert sie aufgrund des automatischen Castings auch für `box2d`- und Geometriewerte. Sie akzeptiert jedoch keine Geometrie- oder `box2d`-Textdarstellung, da diese nicht automatisch gecastet werden.



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



Diese Methode unterstützt kreisförmige Strings und Kurven.

## Beispiele

```
SELECT ST_YMax('BOX3D(1 2 3, 4 5 6)');
st_ymax
-----
5

SELECT ST_YMax(ST_GeomFromText('LINESTRING(1 3 4, 5 6 7)'));
st_ymax
-----
6

SELECT ST_YMax(CAST('BOX(-3 2, 3 4)' As box2d));
st_ymax
-----
4
--Observe THIS DOES NOT WORK because it will try to auto-cast the string representation to ←
a BOX3D
SELECT ST_YMax('LINESTRING(1 3, 5 6)');

--ERROR: BOX3D parser - doesn't start with BOX3D(

SELECT ST_YMax(ST_GeomFromEWKT('CIRCULARSTRING(220268 150415 1,220227 150505 2,220227 ←
150406 3)'));
st_ymax
-----
150506.126829327
```

## Siehe auch

[ST\\_XMin](#), [ST\\_XMax](#), [ST\\_YMin](#), [ST\\_ZMax](#), [ST\\_ZMin](#)

### 7.18.12 ST\_YMin

`ST_YMin` — Gibt die Y-Minima eines 2D- oder 3D-Begrenzungsrahmens oder einer Geometrie zurück.

## Synopsis

```
float ST_YMin(box3d aGeomorBox2DorBox3D);
```

## Beschreibung

Gibt die Y-Minima eines 2D- oder 3D-Begrenzungsrahmens oder einer Geometrie zurück.



### Note

Obwohl diese Funktion nur für box3d definiert ist, funktioniert sie aufgrund des automatischen Castings auch für box2d- und Geometriewerte. Sie akzeptiert jedoch keine Geometrie- oder box2d-Textdarstellung, da diese nicht automatisch gecastet werden.



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



Diese Methode unterstützt kreisförmige Strings und Kurven.

## Beispiele

```
SELECT ST_YMin('BOX3D(1 2 3, 4 5 6)');
st_ymin
-----
2

SELECT ST_YMin(ST_GeomFromText('LINESTRING(1 3 4, 5 6 7)'));
st_ymin
-----
3

SELECT ST_YMin(CAST('BOX(-3 2, 3 4)' As box2d));
st_ymin
-----
2
--Observe THIS DOES NOT WORK because it will try to auto-cast the string representation to ←
a BOX3D
SELECT ST_YMin('LINESTRING(1 3, 5 6)');

--ERROR: BOX3D parser - doesn't start with BOX3D(

SELECT ST_YMin(ST_GeomFromEWKT('CIRCULARSTRING(220268 150415 1,220227 150505 2,220227 ←
150406 3)'));
st_ymin
-----
150406
```

## Siehe auch

[ST\\_GeomFromEWKT](#), [ST\\_XMin](#), [ST\\_XMax](#), [ST\\_YMax](#), [ST\\_ZMax](#), [ST\\_ZMin](#)

### 7.18.13 ST\_ZMax

**ST\_ZMax** — Gibt die Z-Maxima eines 2D- oder 3D-Begrenzungsrahmens oder einer Geometrie zurück.

## Synopsis

```
float ST_ZMax(box3d aGeomorBox2DorBox3D);
```

## Beschreibung

Gibt die Z-Maxima eines 2D- oder 3D-Begrenzungsrahmens oder einer Geometrie zurück.



### Note

Obwohl diese Funktion nur für `box3d` definiert ist, funktioniert sie aufgrund des automatischen Castings auch für `box2d`- und Geometriewerte. Sie akzeptiert jedoch keine Geometrie- oder `box2d`-Textdarstellung, da diese nicht automatisch gecastet werden.



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



Diese Methode unterstützt kreisförmige Strings und Kurven.

## Beispiele

```
SELECT ST_ZMax('BOX3D(1 2 3, 4 5 6)');
st_zmax
-----
6

SELECT ST_ZMax(ST_GeomFromEWKT('LINESTRING(1 3 4, 5 6 7)'));
st_zmax
-----
7

SELECT ST_ZMax('BOX3D(-3 2 1, 3 4 1) ');
st_zmax
-----
1
--Observe THIS DOES NOT WORK because it will try to auto-cast the string representation to ←
a BOX3D
SELECT ST_ZMax('LINESTRING(1 3 4, 5 6 7)');
--ERROR: BOX3D parser - doesn't start with BOX3D(

SELECT ST_ZMax(ST_GeomFromEWKT('CIRCULARSTRING(220268 150415 1,220227 150505 2,220227 ←
150406 3)'));
st_zmax
-----
3
```

## Siehe auch

[ST\\_GeomFromEWKT](#), [ST\\_XMin](#), [ST\\_XMax](#), [ST\\_YMax](#), [ST\\_YMin](#), [ST\\_ZMax](#)

### 7.18.14 ST\_ZMin

`ST_ZMin` — Gibt die Z-Minima eines 2D- oder 3D-Begrenzungsrahmens oder einer Geometrie zurück.

## Synopsis

```
float ST_ZMin(box3d aGeomorBox2DorBox3D);
```

## Beschreibung

Gibt die Z-Minima eines 2D- oder 3D-Begrenzungsrahmens oder einer Geometrie zurück.



### Note

Obwohl diese Funktion nur für `box3d` definiert ist, funktioniert sie aufgrund des automatischen Castings auch für `box2d`- und Geometriewerte. Sie akzeptiert jedoch keine Geometrie- oder `box2d`-Textdarstellung, da diese nicht automatisch gecastet werden.



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



Diese Methode unterstützt kreisförmige Strings und Kurven.

## Beispiele

```
SELECT ST_ZMin('BOX3D(1 2 3, 4 5 6)');
st_zmin
-----
3

SELECT ST_ZMin(ST_GeomFromEWKT('LINESTRING(1 3 4, 5 6 7)'));
st_zmin
-----
4

SELECT ST_ZMin('BOX3D(-3 2 1, 3 4 1)');
st_zmin
-----
1
--Observe THIS DOES NOT WORK because it will try to auto-cast the string representation to ←
a BOX3D
SELECT ST_ZMin('LINESTRING(1 3 4, 5 6 7)');
--ERROR: BOX3D parser - doesn't start with BOX3D(

SELECT ST_ZMin(ST_GeomFromEWKT('CIRCULARSTRING(220268 150415 1,220227 150505 2,220227 ←
150406 3)'));
st_zmin
-----
1
```

## Siehe auch

[ST\\_GeomFromEWKT](#), [ST\\_GeomFromText](#), [ST\\_XMin](#), [ST\\_XMax](#), [ST\\_YMax](#), [ST\\_YMin](#), [ST\\_ZMax](#)

## 7.19 Kilometrierung

### 7.19.1 ST\_LineInterpolatePoint

`ST_LineInterpolatePoint` — Gibt einen oder mehrere, entlang einer Linie interpolierte Punkte zurück.

## Synopsis

```
geometry ST_LineInterpolatePoint(geometry a_linestring, float8 a_fraction);  
geography ST_LineInterpolatePoint(geography a_linestring, float8 a_fraction, boolean use_spheroid = true);
```

## Beschreibung

Fügt einen Punkt entlang einer Linie ein. Der erste Parameter muss einen Linienzug beschreiben. Der zweite Parameter, in Float8-Darstellung mit den Werten von 0 bis 1, gibt jenen Bruchteil der Gesamtlänge des Linienzuges an, wo der Punkt liegen soll.

Siehe [ST\\_LineLocatePoint](#) um die nächstliegende Linie zu einem Punkt zu berechnen.



### Note

Diese Funktion berechnet Punkte in 2D und interpoliert dann Werte für Z und M, während [ST\\_3DLineInterpolatePoint](#) Punkte in 3D berechnet und nur den M-Wert interpoliert.



### 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.

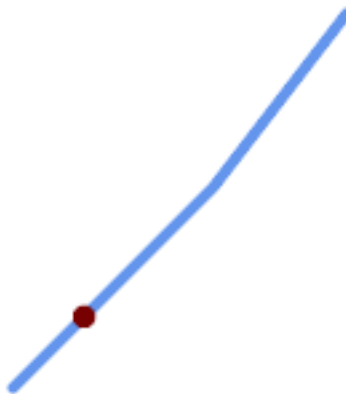
Verfügbarkeit: 0.8.2, Z und M Unterstützung wurde mit 1.1.1 hinzugefügt

Änderung: 2.1.0. Bis zu 2.0.x wurde diese Funktion mit `ST_Line_Interpolate_Point` bezeichnet.



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

## Beispiele



*Ein Linienzug mit dem interpolierten Punkt bei Position 0.20 (20%)*

```
-- The point 20% along a line  
  
SELECT ST_AsEWKT( ST_LineInterpolatePoint(  
    'LINESTRING(25 50, 100 125, 150 190)',
```

```

    0.2 ));
-----
POINT(51.5974135047432 76.5974135047432)

```

Gibt einen oder mehrere, entlang einer Linie interpolierte Punkte zurück.

```

SELECT ST_AsEWKT( ST_LineInterpolatePoint(
    LINESTRING(1 2 3, 4 5 6, 6 7 8)',
    0.5 ));
-----
POINT(3.5 4.5 5.5)

```

Der Punkt auf einer Linie, der einem Punkt am nächsten liegt:

```

SELECT ST_AsText( ST_LineInterpolatePoint( line.geom,
    ST_LineLocatePoint( line.geom, 'POINT(4 3)'))
FROM (SELECT ST_GeomFromText('LINESTRING(1 2, 4 5, 6 7)') As geom) AS line;
-----
POINT(3 4)

```

### Siehe auch

[ST\\_LineInterpolatePoints](#), [ST\\_LineInterpolatePoint](#), [ST\\_LineMerge](#)

## 7.19.2 ST\_3DLineInterpolatePoint

`ST_3DLineInterpolatePoint` — Gibt einen oder mehrere, entlang einer Linie interpolierte Punkte zurück.

### Synopsis

geometry **ST\_3DLineInterpolatePoint**(geometry a\_linestring, float8 a\_fraction);

### Beschreibung

Fügt einen Punkt entlang einer Linie ein. Der erste Parameter muss einen Linienzug beschreiben. Der zweite Parameter, in Float8-Darstellung mit den Werten von 0 bis 1, gibt jenen Bruchteil der Gesamtlänge des Linienzuges an, wo der Punkt liegen soll.



#### Note

`ST_LineInterpolatePoint` berechnet Punkte in 2D und interpoliert dann die Werte für Z und M, während diese Funktion Punkte in 3D berechnet und nur den M-Wert interpoliert.

Verfügbarkeit: 2.1.0



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

### Beispiele

Gibt einen oder mehrere, entlang einer Linie interpolierte Punkte zurück.

```
SELECT ST_AsText (
  ST_3DLineInterpolatePoint('LINESTRING(25 50 70, 100 125 90, 150 190 200)',
    0.20));

st_asetext
-----
POINT Z (59.0675892910822 84.0675892910822 79.0846904776219)
```

### Siehe auch

[ST\\_LineInterpolatePoint](#), [ST\\_LineInterpolatePoint](#), [ST\\_LineMerge](#)

## 7.19.3 ST\_LineInterpolatePoints

`ST_LineInterpolatePoints` — Gibt einen oder mehrere, entlang einer Linie interpolierte Punkte zurück.

### Synopsis

geometry **ST\_LineInterpolatePoints**(geometry a\_linestring, float8 a\_fraction, boolean repeat);  
geography **ST\_LineInterpolatePoints**(geography a\_linestring, float8 a\_fraction, boolean use\_spheroid = true, boolean repeat = true);

### Beschreibung

Gibt einen oder mehrere Punkte zurück, die entlang einer Linie in einem Bruchteilintervall interpoliert wurden. Das erste Argument muss ein `LINESTRING` sein. Das zweite Argument ist ein `float8` zwischen 0 und 1, der den Abstand zwischen den Punkten als Bruchteil der Linienlänge angibt. Wenn das dritte Argument `false` ist, wird höchstens ein Punkt konstruiert (was [ST\\_LineInterpolatePoint](#) entspricht).

Wenn das Ergebnis null oder einen Punkt hat, wird es als `POINT` zurückgegeben. Wenn es zwei oder mehr Punkte hat, wird es als `MULTIPOINT` zurückgegeben.

Verfügbarkeit: 2.5.0



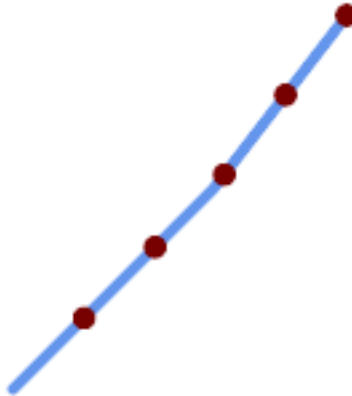
Diese Funktion unterstützt 3d und lässt den Z-Index nicht fallen.



Diese Funktion unterstützt M-Koordinaten.



## Beispiele



*Ein LineString mit alle 20% interpolierten Punkten*

```
--Return points each 20% along a 2D line
SELECT ST_AsText(ST_LineInterpolatePoints('LINESTRING(25 50, 100 125, 150 190)', 0.20))
-----
MULTIPOINT((51.5974135047432 76.5974135047432), (78.1948270094864 103.194827009486) ←
, (104.132163186446 130.37181214238), (127.066081593223 160.18590607119), (150 190))
```

## Siehe auch

**ST\_LineInterpolatePoint** Diese Funktion ändert den Namen einer bestehenden TopoGeometry-Spalte und stellt sicher, dass die Metadateninformationen entsprechend aktualisiert werden.

### 7.19.4 ST\_LineLocatePoint

**ST\_LineLocatePoint** — Liefert die gebrochene Position des Punktes auf einer Linie, der einem Punkt am nächsten liegt.

#### Synopsis

```
float8 ST_LineLocatePoint(geometry a_linestring, geometry a_point);
float8 ST_LineLocatePoint(geography a_linestring, geography a_point, boolean use_spheroid = true);
```

#### Beschreibung

Gibt eine Gleitpunktzahl zwischen 0 und 1 zurück, welche die Lage des Punktes auf einer Linie angibt, der zu einem gegebenen Punkt am nächsten liegt. Die Lage wird als Anteil an der Gesamtlänge der **2D Linie** angegeben.

Sie können die zurückgegebene Lage nutzen, um einen Punkt (**ST\_LineInterpolatePoint**) oder eine Teilzeichenfolge (**ST\_LineSubstring**) zu extrahieren.

Nützlich, um die Hausnummern von Adressen anzunähern

Verfügbarkeit: 1.1.0

Änderung: 2.1.0. Bis zu 2.0.x wurde diese Funktion mit `ST_Line_Locate_Point` bezeichnet.

## Beispiele

```
--Rough approximation of finding the street number of a point along the street
--Note the whole foo thing is just to generate dummy data that looks
--like house centroids and street
--We use ST_DWithin to exclude
--houses too far away from the street to be considered on the street
SELECT ST_AsText(house_loc) As as_text_house_loc,
       startstreet_num +
       CAST( (endstreet_num - startstreet_num)
            * ST_LineLocatePoint(street_line, house_loc) As integer) As ←
       street_num
FROM
  (SELECT ST_GeomFromText('LINESTRING(1 2, 3 4)') As street_line,
        ST_Point(x*1.01,y*1.03) As house_loc, 10 As startstreet_num,
        20 As endstreet_num
  FROM generate_series(1,3) x CROSS JOIN generate_series(2,4) As y)
As foo
WHERE ST_DWithin(street_line, house_loc, 0.2);

  as_text_house_loc | street_num
-----+-----
POINT(1.01 2.06)   |          10
POINT(2.02 3.09)   |          15
POINT(3.03 4.12)   |          20

--find closest point on a line to a point or other geometry
SELECT ST_AsText(ST_LineInterpolatePoint(foo.the_line, ST_LineLocatePoint(foo.the_line, ←
  ST_GeomFromText('POINT(4 3)'))))
FROM (SELECT ST_GeomFromText('LINESTRING(1 2, 4 5, 6 7)') As the_line) As foo;
  st_astext
-----
POINT(3 4)
```

## Siehe auch

[ST\\_DWithin](#), [ST\\_Length2D](#), [ST\\_LineInterpolatePoint](#), [ST\\_LineSubstring](#)

### 7.19.5 ST\_LineSubstring

`ST_LineSubstring` — Gibt den Teil einer Linie zwischen zwei gebrochenen Stellen zurück.

#### Synopsis

```
geometry ST_LineSubstring(geometry a_linestring, float8 startfraction, float8 endfraction);
geography ST_LineSubstring(geography a_linestring, float8 startfraction, float8 endfraction);
```

#### Beschreibung

Berechnet die Zeile, die der Abschnitt der Eingabezeile ist, der an den angegebenen Bruchstellen beginnt und endet. Das erste Argument muss ein `LINESTRING` sein. Das zweite und dritte Argument sind Werte im Bereich `[0, 1]`, die die Anfangs- und Endpunkte als Bruchteile der Zeilenlänge darstellen. Die Werte `Z` und `M` werden für hinzugefügte Endpunkte interpoliert, falls vorhanden.

Gleichbedeutend mit [ST\\_LineInterpolatePoint](#), wenn Anfangswert und Endwert ident sind.

**Note**

Dies funktioniert nur bei LINESTRINGS. Zur Verwendung bei zusammenhängenden MULTILINESTRINGS müssen diese zunächst mit `ST_LineMerge` verbunden werden.

**Note**

Ab Version 1.1.1 interpoliert diese Funktion auch M- und Z-Werte (falls vorhanden), während frühere Versionen unbestimmte Werte setzen.

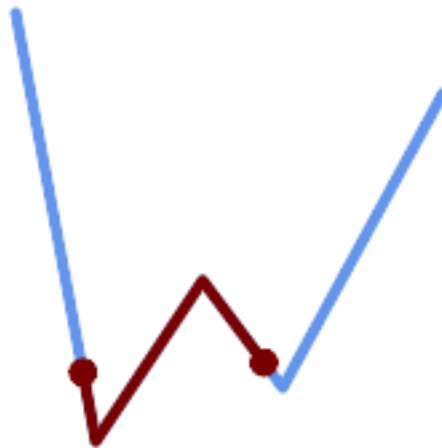
Verbessert: 3.4.0 - Unterstützung für Geographie wurde eingeführt.

Änderung: 2.1.0. Bis zu 2.0.x wurde diese Funktion mit `ST_Line_Substring` bezeichnet.

Verfügbarkeit: 1.1.0, mit 1.1.1 wurde die Unterstützung für Z und M hinzugefügt



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

**Beispiele**

*Ein Liniensegment von einem Mittelstück mit 1/3 Länge überlagert (0.333, 0.666)*

```
SELECT ST_AsText(ST_LineSubstring( 'LINESTRING (20 180, 50 20, 90 80, 120 40, 180 150)', ←
  0.333, 0.666));
```

```
LINESTRING (45.17311810399485 45.74337011202746, 50 20, 90 80, 112.97593050157862 ←
  49.36542599789519)
```

Wenn Anfangs- und Endpunkt gleich sind, ist das Ergebnis ein PUNKT.

```
SELECT ST_AsText(ST_LineSubstring( 'LINESTRING(25 50, 100 125, 150 190)', 0.333, 0.333));
-----
POINT (69.2846934853974 94.2846934853974)
```

Eine Abfrage, um einen LineString in Abschnitte der Länge 100 oder kürzer zu schneiden. Sie verwendet `generate_series()` mit einem `CROSS JOIN LATERAL`, um das Äquivalent einer FOR-Schleife zu erzeugen.

```

WITH data(id, geom) AS (VALUES
  ( 'A', 'LINESTRING( 0 0, 200 0)::geometry ),
  ( 'B', 'LINESTRING( 0 100, 350 100)::geometry ),
  ( 'C', 'LINESTRING( 0 200, 50 200)::geometry )
)
SELECT id, i,
       ST_AsText( ST_LineSubstring( geom, startfrac, LEAST( endfrac, 1 ) ) ) AS geom
FROM (
  SELECT id, geom, ST_Length(geom) len, 100 sublen FROM data
) AS d
CROSS JOIN LATERAL (
  SELECT i, (sublen * i) / len AS startfrac,
         (sublen * (i+1)) / len AS endfrac
  FROM generate_series(0, floor( len / sublen )::integer ) AS t(i)
  -- skip last i if line length is exact multiple of sublen
  WHERE (sublen * i) / len <
> 1.0
) AS d2;

```

id	i	geom
A	0	LINESTRING(0 0,100 0)
A	1	LINESTRING(100 0,200 0)
B	0	LINESTRING(0 100,100 100)
B	1	LINESTRING(100 100,200 100)
B	2	LINESTRING(200 100,300 100)
B	3	LINESTRING(300 100,350 100)
C	0	LINESTRING(0 200,50 200)

### Geografische Umsetzungsmaßnahmen entlang eines Sphäroids, Geometrie entlang einer Linie

```

SELECT ST_AsText(ST_LineSubstring( 'LINESTRING(-118.2436 34.0522, -71.0570 42.3611):: ↵
      geography, 0.333, 0.666),6) AS geog_sub
, ST_AsText(ST_LineSubstring('LINESTRING(-118.2436 34.0522, -71.0570 42.3611)::geometry, ↵
      0.333, 0.666),6) AS geom_sub;
-----
geog_sub | LINESTRING(-104.167064 38.854691,-87.674646 41.849854)
geom_sub | LINESTRING(-102.530462 36.819064,-86.817324 39.585927)

```

#### Siehe auch

[ST\\_Length](#), [ST\\_LineExtend](#), [ST\\_LineInterpolatePoint](#), [ST\\_LineMerge](#)

## 7.19.6 ST\_LocateAlong

`ST_LocateAlong` — Gibt die Punkte auf einer Geometrie zurück, die einem Messwert entsprechen.

#### Synopsis

```
geometry ST_LocateAlong(geometry geom_with_measure, float8 measure, float8 offset = 0);
```

#### Beschreibung

Gibt die Position(en) entlang einer gemessenen Geometrie zurück, die die angegebenen Messwerte aufweisen. Das Ergebnis ist ein Point oder MultiPoint. Polygonale Eingaben werden nicht unterstützt.

Wenn `offset` angegeben ist, wird das Ergebnis um den angegebenen Abstand nach links oder rechts von der Eingabezeile verschoben. Ein positiver Versatz geht nach links, ein negativer nach rechts.



#### Note

Verwenden Sie diese Funktion nur für lineare Geometrien mit einer M-Komponente

Die Semantik ist in der Norm *ISO/IEC 13249-3 SQL/MM Spatial* festgelegt.

Verfügbarkeit: 1.1.0 über die alte Bezeichnung `ST_Locate_Along_Measure`.

Änderung: 2.0.0 In Vorgängerversionen als `ST_Locate_Along_Measure` bezeichnet. Der alte Name ist überholt und wird in der Zukunft entfernt ist aber noch verfügbar.



Diese Funktion unterstützt M-Koordinaten.



Diese Methode setzt die SQL/MM-Spezifikation um. SQL-MM IEC 13249-3: 5.1.13

#### Beispiele

```
SELECT ST_AsText (
  ST_LocateAlong (
    'MULTILINESTRINGM((1 2 3, 3 4 2, 9 4 3),(1 2 3, 5 4 5))'::geometry,
    3 ));
```

```
-----
MULTIPOINT M ((1 2 3), (9 4 3), (1 2 3))
```

#### Siehe auch

[ST\\_LocateBetween](#), [ST\\_LocateBetweenElevations](#), [ST\\_InterpolatePoint](#)

### 7.19.7 ST\_LocateBetween

`ST_LocateBetween` — Gibt die Teile einer Geometrie zurück, die einem Messbereich entsprechen.

#### Synopsis

```
geometry ST_LocateBetween(geometry geom, float8 measure_start, float8 measure_end, float8 offset = 0);
```

#### Beschreibung

Gibt eine abgeleitete Sammelgeometrie mit jenen Elementen zurück, die in dem gegebenen Kilometrierungsintervall liegen; das Intervall ist unbeschränkt. Polygonale Elemente werden nicht unterstützt.

Wenn ein Versatz angegeben ist, werden die Resultierenden um diese Anzahl an Einheiten nach links oder rechts von der gegebenen Linie versetzt. Ein positiver Versatz geschieht nach links, ein negativer nach rechts.

Das Beschneiden eines nicht konvexen POLYGONs kann zu einer ungültigen Geometrie führen.

Die Semantik ist in der Norm *ISO/IEC 13249-3 SQL/MM Spatial* festgelegt.

Verfügbarkeit: 1.1.0 über die alte Bezeichnung `ST_Locate_Between_Measures`.

Änderung: 2.0.0 In Vorgängerversionen als `ST_Locate_Along_Measure` bezeichnet. Der alte Name ist überholt und wird in der Zukunft entfernt ist aber noch verfügbar.

Verbessert: 3.0.0 - Unterstützung für POLYGON, TIN, TRIANGLE hinzugefügt.



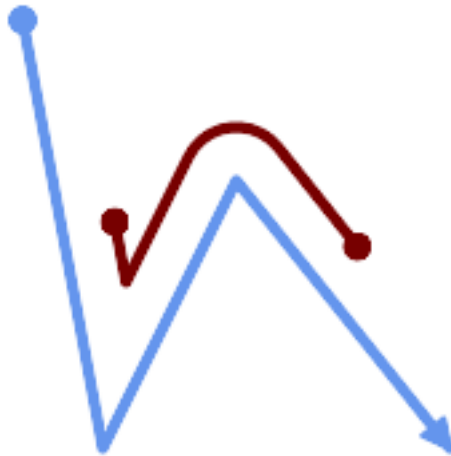
Diese Funktion unterstützt M-Koordinaten.



Diese Methode setzt die SQL/MM-Spezifikation um. SQL-MM IEC 13249-3: 5.1

## Beispiele

```
SELECT ST_AsText (
  ST_LocateBetween(
    'MULTILINESTRING M ((1 2 3, 3 4 2, 9 4 3),(1 2 3, 5 4 5))':: geometry,
    1.5, 3 ));
-----
GEOMETRYCOLLECTION M (LINESTRING M (1 2 3,3 4 2,9 4 3),POINT M (1 2 3))
```



*Ein LineString mit dem Abschnitt zwischen den Takten 2 und 8, nach links versetzt*

```
SELECT ST_AsText( ST_LocateBetween(
  ST_AddMeasure('LINESTRING (20 180, 50 20, 100 120, 180 20)', 0, 10),
  2, 8,
  20
));
-----
MULTILINESTRING((54.49835019899045 104.53426957938231,58.70056060327303 ↔
82.12248075654186,69.16695286779743 103.05526528559065,82.11145618000168 ↔
128.94427190999915,84.24893681714357 132.32493442618113,87.01636951231555 ↔
135.21267035596549,90.30307285299679 137.49198684843182,93.97759758337769 ↔
139.07172433557758,97.89298381958797 139.8887023914453,101.89263860095893 ↔
139.9102465862721,105.81659870902816 139.13549527600819,109.50792827749828 ↔
137.5954340631298,112.81899532549731 135.351656550512,115.6173761888606 ↔
132.49390095108848,145.31017306064817 95.37790486135405))
```

## Siehe auch

[ST\\_LocateAlong](#), [ST\\_LocateAlong](#), [ST\\_LocateBetween](#)

## 7.19.8 ST\_LocateBetweenElevations

ST\_LocateBetweenElevations — Gibt die Teile einer Geometrie zurück, die in einem Höhenbereich (Z) liegen.

### Synopsis

geometry **ST\_LocateBetweenElevations**(geometry geom, float8 elevation\_start, float8 elevation\_end);

### Beschreibung

Gibt eine Geometrie (Sammlung) mit den Teilen einer Geometrie zurück, die in einem Höhenbereich (Z) liegen.

Das Beschneiden eines nicht konvexen POLYGONs kann zu einer ungültigen Geometrie führen.

Verfügbarkeit: 1.4.0

Verbessert: 3.0.0 - Unterstützung für POLYGON, TIN, TRIANGLE hinzugefügt.



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

### Beispiele

```
SELECT ST_AsText (
  ST_LocateBetweenElevations (
    'LINESTRING(1 2 3, 4 5 6)::geometry,
    2, 4 ));

          st_astext
-----
MULTILINESTRING Z ((1 2 3,2 3 4))

SELECT ST_AsText (
  ST_LocateBetweenElevations (
    'LINESTRING(1 2 6, 4 5 -1, 7 8 9)',
    6, 9)) As ewelev;

          ewelev
-----
GEOMETRYCOLLECTION Z (POINT Z (1 2 6),LINESTRING Z (6.1 7.1 6,7 8 9))
```

### Siehe auch

[ST\\_Dump](#), [ST\\_LocateAlong](#), [ST\\_LocateBetween](#)

## 7.19.9 ST\_InterpolatePoint

ST\_InterpolatePoint — Für einen gegebenen Punkt wird die Kilometrierung auf dem nächstliegenden Punkt einer Geometrie zurück.

### Synopsis

float8 **ST\_InterpolatePoint**(geometry linear\_geom\_with\_measure, geometry point);

## Beschreibung

Gibt einen interpolierten Messwert einer linear gemessenen Geometrie an der Stelle zurück, die dem angegebenen Punkt am nächsten liegt.



### Note

Verwenden Sie diese Funktion nur für lineare Geometrien mit einer M-Komponente

Verfügbarkeit: 2.0.0



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

## Beispiele

```
SELECT ST_InterpolatePoint('LINESTRING M (0 0 0, 10 0 20)', 'POINT(5 5)');
-----
10
```

## Siehe auch

[ST\\_AddMeasure](#), [ST\\_LocateAlong](#), [ST\\_LocateBetween](#)

## 7.19.10 ST\_AddMeasure

ST\_AddMeasure — Interpoliert Maße entlang einer linearen Geometrie.

### Synopsis

geometry **ST\_AddMeasure**(geometry geom\_mline, float8 measure\_start, float8 measure\_end);

### Beschreibung

Gibt eine abgeleitete Geometrie mit einer zwischen Anfangs- und Endpunkt linear interpolierten Kilometrierung zurück. Wenn die Geometrie keine Dimension für die Kilometrierung aufweist, wird diese hinzugefügt. Wenn die Geometrie eine Dimension für die Kilometrierung hat, wird diese mit den neuen Werten überschrieben. Es werden nur LINESTRINGs und MULTILINESTRINGs unterstützt.

Verfügbarkeit: 1.5.0



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

## Beispiele

```
SELECT ST_AsText(ST_AddMeasure(
ST_GeomFromEWKT('LINESTRING(1 0, 2 0, 4 0)'),1,4)) As ewelev;
-----
LINESTRINGM(1 0 1,2 0 2,4 0 4)

SELECT ST_AsText(ST_AddMeasure(
```



```

ST_GeomFromEWKT('LINESTRING(1 0 4, 2 0 4, 4 0 4)'),10,40)) As ewelev;
      ewelev
-----
LINESTRING(1 0 4 10,2 0 4 20,4 0 4 40)

SELECT ST_AsText(ST_AddMeasure(
ST_GeomFromEWKT('LINESTRINGM(1 0 4, 2 0 4, 4 0 4)'),10,40)) As ewelev;
      ewelev
-----
LINESTRINGM(1 0 10,2 0 20,4 0 40)

SELECT ST_AsText(ST_AddMeasure(
ST_GeomFromEWKT('MULTILINESTRINGM((1 0 4, 2 0 4, 4 0 4),(1 0 4, 2 0 4, 4 0 4)'),10,70)) As ←
      ewelev;
      ewelev
-----
MULTILINESTRINGM((1 0 10,2 0 20,4 0 40),(1 0 40,2 0 50,4 0 70))

```

## 7.20 Trajektorie-Funktionen

### 7.20.1 ST\_IsValidTrajectory

`ST_IsValidTrajectory` — Prüft, ob die Geometrie eine gültige Flugbahn ist.

#### Synopsis

boolean `ST_IsValidTrajectory`(geometry line);

#### Beschreibung

Prüft, ob eine Geometrie eine gültige Trajektorie kodiert. Eine gültige Flugbahn wird als `LINESTRING` mit Maßen (M-Werten) dargestellt. Die Messwerte müssen von jedem Scheitelpunkt zum nächsten ansteigen.

Gültige Trajektorien werden als Eingabe für räumlich-zeitliche Funktionen wie [ST\\_ClosestPointOfApproach](#)

Verfügbarkeit: 2.2.0



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

#### Beispiele

```

-- A valid trajectory
SELECT ST_IsValidTrajectory(ST_MakeLine(
  ST_MakePointM(0,0,1),
  ST_MakePointM(0,1,2))
);
t

-- An invalid trajectory
SELECT ST_IsValidTrajectory(ST_MakeLine(ST_MakePointM(0,0,1), ST_MakePointM(0,1,0)));
NOTICE:  Measure of vertex 1 (0) not bigger than measure of vertex 0 (1)
st_isvalidtrajectory
-----
f

```

**Siehe auch**[ST\\_ClosestPointOfApproach](#)**7.20.2 ST\_ClosestPointOfApproach**

`ST_ClosestPointOfApproach` — Liefert ein Maß für den nächstgelegenen Punkt der Annäherung von zwei Flugbahnen.

**Synopsis**

```
float8 ST_ClosestPointOfApproach(geometry track1, geometry track2);
```

**Beschreibung**

Gibt das kleinste Maß zurück, bei dem die entlang der angegebenen Trajektorien interpolierten Punkte den geringsten Abstand zueinander haben.

Die Eingaben müssen gültige Flugbahnen sein, wie von [ST\\_IsValidTrajectory](#) geprüft. Null wird zurückgegeben, wenn sich die Trajektorien in ihren M-Bereichen nicht überschneiden.

Um die tatsächlichen Punkte am berechneten Maß zu erhalten, verwenden Sie [ST\\_LocateAlong](#).

Verfügbarkeit: 2.2.0



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

**Beispiele**

```
-- Return the time in which two objects moving between 10:00 and 11:00
-- are closest to each other and their distance at that point
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_GeometryN(ST_LocateAlong(a,m),1) pa,
    ST_GeometryN(ST_LocateAlong(b,m),1) pb
  FROM inp, cpa
)
SELECT to_timestamp(m) t,
  ST_Distance(pa,pb) distance,
  ST_AsText(pa, 2) AS pa, ST_AsText(pb, 2) AS pb
FROM points, cpa;
```

t	distance	pa	pb
2015-05-26 10:45:31.034483-07	1.9603683315139542	POINT ZM (7.59 0 3.79 1432662331.03)	POINT ZM (9.1 1.24 3.93 1432662331.03)

**Siehe auch**

[ST\\_IsValidTrajectory](#), [ST\\_DistanceCPA](#), [ST\\_LocateAlong](#), [ST\\_AddMeasure](#)

**7.20.3 ST\_DistanceCPA**

`ST_DistanceCPA` — Liefert den Abstand zwischen dem nächstgelegenen Punkt der Annäherung zweier Flugbahnen.

**Synopsis**

```
float8 ST_DistanceCPA(geometry track1, geometry track2);
```

**Beschreibung**

Liefert den Abstand (in 2D) zwischen zwei Flugbahnen an ihrem engsten Annäherungspunkt.

Die Eingaben müssen gültige Flugbahnen sein, wie von [ST\\_IsValidTrajectory](#) geprüft. Null wird zurückgegeben, wenn sich die Trajektorien in ihren M-Bereichen nicht überschneiden.

Verfügbarkeit: 2.2.0



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

**Beispiele**

```
-- Return the minimum distance of two objects moving between 10:00 and 11:00
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](#), [l=](#)

**7.20.4 ST\_CPAWithin**

`ST_CPAWithin` — Prüft, ob der nächstgelegene Punkt der Annäherung zweier Flugbahnen innerhalb der angegebenen Entfernung liegt.

**Synopsis**

```
boolean ST_CPAWithin(geometry track1, geometry track2, float8 dist);
```

## Beschreibung

Prüft, ob sich zwei bewegte Objekte jemals näher als die angegebene Entfernung befunden haben.

Die Eingaben müssen gültige Flugbahnen sein, wie von [ST\\_IsValidTrajectory](#) geprüft. False wird zurückgegeben, wenn sich die Trajektorien in ihren M-Bereichen nicht überschneiden.

Verfügbarkeit: 2.2.0



Diese Funktion unterstützt 3d und lässt den Z-Index nicht fallen.

## Beispiele

```
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_CPAWithin(a,b,2), ST_DistanceCPA(a,b) distance FROM inp;
```

st_cpawithin	distance
t	1.96521473776207

## Siehe auch

[ST\\_IsValidTrajectory](#), [ST\\_ClosestPointOfApproach](#), [ST\\_DistanceCPA](#), [l=](#)

## 7.21 Version Funktionen

### 7.21.1 PostGIS\_Extensions\_Upgrade

PostGIS\_Extensions\_Upgrade — Packt und aktualisiert PostGIS-Erweiterungen (z.B. `postgis_raster`, `postgis_topology`, `postgis_sfcgal`) auf die angegebene oder neueste Version.

#### Synopsis

```
text PostGIS_Extensions_Upgrade(text target_version=null);
```

#### Beschreibung

Pakete und Upgrades von PostGIS-Erweiterungen auf die angegebene oder neueste Version. Nur die Erweiterungen, die Sie in der Datenbank installiert haben, werden gepackt und aktualisiert, falls erforderlich. Meldet die vollständige PostGIS-Version und die Informationen zur Build-Konfiguration nach. Dies ist eine Abkürzung für die Durchführung mehrerer `CREATE EXTENSION .. FROM unpackaged` und `ALTER EXTENSION .. UPDATE` für jede PostGIS-Erweiterung. Derzeit wird nur versucht, die Erweiterungen `postgis`, `postgis_raster`, `postgis_sfcgal`, `postgis_topology` und `postgis_tiger_geocoder` zu aktualisieren.

Verfügbarkeit: 2.5.0

**Note**

Geändert: 3.4.0 um das Argument `target_version` hinzuzufügen.  
 Geändert: 3.3.0 Unterstützung für Upgrades von jeder PostGIS-Version. Funktioniert nicht auf allen Systemen.  
 Geändert: 3.0.0, um lose Erweiterungen neu zu packen und `postgis_raster` zu unterstützen.

**Beispiele**

```
SELECT PostGIS_Extensions_Upgrade();
```

```
NOTICE: Packaging extension postgis
NOTICE: Packaging extension postgis_raster
NOTICE: Packaging extension postgis_sfcgal
NOTICE: Extension postgis_topology is not available or not packagable for some reason
NOTICE: Extension postgis_tiger_geocoder is not available or not packagable for some ↔
        reason

        postgis_extensions_upgrade
-----
Upgrade completed, run SELECT postgis_full_version(); for details
(1 row)
```

**Siehe auch**

Section 3.4, [PostGIS\\_GEOS\\_Version](#), [PostGIS\\_Lib\\_Version](#), [PostGIS\\_LibXML\\_Version](#), [PostGIS\\_PROJ\\_Version](#), [PostGIS\\_Version](#)

**7.21.2 PostGIS\_Full\_Version**

`PostGIS_Full_Version` — Meldet die vollständige PostGIS-Version und Informationen zur Build-Konfiguration.

**Synopsis**

```
text PostGIS_Full_Version();
```

**Beschreibung**

Meldet die vollständige PostGIS-Version und Informationen zur Build-Konfiguration. Informiert auch über die Synchronisation zwischen Bibliotheken und Skripten und schlägt bei Bedarf Upgrades vor.

Verbessert: 3.4.0 enthält jetzt zusätzliche PROJ-Konfigurationen `NETWORK_ENABLED`, `URL_ENDPOINT` und `DATABASE_PATH` des `proj.db`-Speicherorts

**Beispiele**

```
SELECT PostGIS_Full_Version();

        postgis_full_version
-----
POSTGIS="3.4.0dev 3.3.0rc2-993-g61bdf43a7" [EXTENSION] PGSQL="160" GEOS="3.12.0dev-CAPI ↔
-1.18.0" SFCGAL="1.3.8" PROJ="7.2.1 NETWORK_ENABLED=OFF URL_ENDPOINT=https://cdn.proj. ↔
org USER_WRITABLE_DIRECTORY=/tmp/proj DATABASE_PATH=/usr/share/proj/proj.db" GDAL="GDAL ↔
3.2.2, released 2021/03/05" LIBXML="2.9.10" LIBJSON="0.15" LIBPROTOBUF="1.3.3" WAGYU ↔
="0.5.0 (Internal)" TOPOLOGY RASTER
(1 row)
```

**Siehe auch**

Section 3.4, [PostGIS\\_GEOS\\_Version](#), [PostGIS\\_Lib\\_Version](#), [PostGIS\\_LibXML\\_Version](#), [PostGIS\\_PROJ\\_Version](#), [PostGIS\\_Wagyu\\_Version](#), [PostGIS\\_Version](#)

**7.21.3 PostGIS\_GEOS\_Version**

`PostGIS_GEOS_Version` — Gibt die Versionsnummer der GEOS-Bibliothek zurück.

**Synopsis**

```
text PostGIS_GEOS_Version();
```

**Beschreibung**

Gibt die Versionsnummer der GEOS-Bibliothek zurück, oder NULL wenn die GEOS-Unterstützung nicht aktiviert ist.

**Beispiele**

```
SELECT PostGIS_GEOS_Version() ;
  postgis_geos_version
-----
3.12.0dev-CAPI-1.18.0
(1 row)
```

**Siehe auch**

[PostGIS\\_Full\\_Version](#), [PostGIS\\_Lib\\_Version](#), [PostGIS\\_LibXML\\_Version](#), [PostGIS\\_PROJ\\_Version](#), [PostGIS\\_Version](#)

**7.21.4 PostGIS\_GEOS\_Compiled\_Version**

`PostGIS_GEOS_Compiled_Version` — Gibt die Versionsnummer der GEOS-Bibliothek zurück, mit der PostGIS erstellt wurde.

**Synopsis**

```
text PostGIS_GEOS_Compiled_Version();
```

**Beschreibung**

Gibt die Versionsnummer der GEOS-Bibliothek zurück, mit der PostGIS erstellt wurde.

Verfügbarkeit: 3.4.0

**Beispiele**

```
SELECT PostGIS_GEOS_Compiled_Version() ;
  postgis_geos_compiled_version
-----
3.12.0
(1 row)
```

**Siehe auch**

[PostGIS\\_GEOS\\_Version](#), [PostGIS\\_Full\\_Version](#)

**7.21.5 PostGIS\_Liblwgeom\_Version**

`PostGIS_Liblwgeom_Version` — Gibt die Versionsnummer der liblwgeom-Bibliothek zurück. Diese sollte mit der Version von PostGIS übereinstimmen.

**Synopsis**

```
text PostGIS_Liblwgeom_Version();
```

**Beschreibung**

Liefert die Versionsnummer der Bibliothek liblwgeom/

**Beispiele**

```
SELECT PostGIS_Liblwgeom_Version();
postgis_liblwgeom_version
-----
3.4.0dev 3.3.0rc2-993-g61bdf43a7
(1 row)
```

**Siehe auch**

[PostGIS\\_Full\\_Version](#), [PostGIS\\_Lib\\_Version](#), [PostGIS\\_LibXML\\_Version](#), [PostGIS\\_PROJ\\_Version](#), [PostGIS\\_Version](#)

**7.21.6 PostGIS\_LibXML\_Version**

`PostGIS_LibXML_Version` — Gibt die Versionsnummer der libxml2-Bibliothek zurück.

**Synopsis**

```
text PostGIS_LibXML_Version();
```

**Beschreibung**

Gibt die Versionsnummer der LibXML2-Bibliothek zurück.

Verfügbarkeit: 1.5

**Beispiele**

```
SELECT PostGIS_LibXML_Version();
postgis_libxml_version
-----
2.9.10
(1 row)
```

**Siehe auch**

[PostGIS\\_Full\\_Version](#), [PostGIS\\_Lib\\_Version](#), [PostGIS\\_PROJ\\_Version](#), [PostGIS\\_GEOS\\_Version](#), [PostGIS\\_Version](#)

**7.21.7 PostGIS\_LibJSON\_Version**

PostGIS\_LibJSON\_Version — Returns the version number of the libjson-c library.

**Synopsis**

```
text PostGIS_LibJSON_Version();
```

**Beschreibung**

Returns the version number of the LibJSON-C library.

**Beispiele**

```
SELECT PostGIS_LibJSON_Version();
 postgis_libjson_version
-----
0.17
```

**Siehe auch**

[PostGIS\\_Full\\_Version](#), [PostGIS\\_Lib\\_Version](#), [PostGIS\\_PROJ\\_Version](#), [PostGIS\\_GEOS\\_Version](#), [PostGIS\\_Version](#)

**7.21.8 PostGIS\_Lib\_Build\_Date**

PostGIS\_Lib\_Build\_Date — Gibt das Erstellungsdatum der PostGIS-Bibliothek zurück.

**Synopsis**

```
text PostGIS_Lib_Build_Date();
```

**Beschreibung**

Gibt das Erstellungsdatum der PostGIS-Bibliothek zurück.

**Beispiele**

```
SELECT PostGIS_Lib_Build_Date();
 postgis_lib_build_date
-----
2023-06-22 03:56:11
(1 row)
```

**7.21.9 PostGIS\_Lib\_Version**

PostGIS\_Lib\_Version — Gibt die Versionsnummer der PostGIS-Bibliothek zurück.

---



## Synopsis

```
text PostGIS_Lib_Version();
```

## Beschreibung

Gibt die Versionsnummer der PostGIS-Bibliothek zurück.

## Beispiele

```
SELECT PostGIS_Lib_Version();
   postgis_lib_version
-----
3.4.0dev
(1 row)
```

## Siehe auch

[PostGIS\\_Full\\_Version](#), [PostGIS\\_GEOS\\_Version](#), [PostGIS\\_LibXML\\_Version](#), [PostGIS\\_PROJ\\_Version](#), [PostGIS\\_Version](#)

### 7.21.10 PostGIS\_PROJ\_Version

`PostGIS_PROJ_Version` — Gibt die Versionsnummer der PROJ4-Bibliothek zurück.

## Synopsis

```
text PostGIS_PROJ_Version();
```

## Beschreibung

Gibt die Versionsnummer der PROJ-Bibliothek und einige Konfigurationsoptionen von proj zurück.

Verbessert: 3.4.0 enthält jetzt `NETWORK_ENABLED`, `URL_ENDPOINT` und `DATABASE_PATH` des proj.db-Speicherorts

## Beispiele

```
SELECT PostGIS_PROJ_Version();
   postgis_proj_version
-----
7.2.1 NETWORK_ENABLED=OFF URL_ENDPOINT=https://cdn.proj.org USER_WRITABLE_DIRECTORY=/tmp/ ↵
   proj DATABASE_PATH=/usr/share/proj/proj.db
(1 row)
```

## Siehe auch

[PostGIS\\_PROJ\\_Compiled\\_Version](#), [PostGIS\\_Full\\_Version](#), [PostGIS\\_GEOS\\_Version](#), [PostGIS\\_Lib\\_Version](#), [PostGIS\\_LibXML\\_Version](#), [PostGIS\\_Version](#)

### 7.21.11 PostGIS\_PROJ\_Compiled\_Version

`PostGIS_PROJ_Compiled_Version` — Returns the version number of the PROJ library against which PostGIS was built.

## Synopsis

text **PostGIS\_PROJ\_Compiled\_Version()**;

## Beschreibung

Returns the version number of the PROJ library, or against which PostGIS was built.

Verfügbarkeit: 3.5.0

## Beispiele

```
SELECT PostGIS_PROJ_Compiled_Version();
 postgis_proj_compiled_version
-----
 9.1.1
(1 row)
```

## Siehe auch

[PostGIS\\_PROJ\\_Version](#), [PostGIS\\_Full\\_Version](#)

## 7.21.12 PostGIS\_Wagyu\_Version

PostGIS\_Wagyu\_Version — Gibt die Versionsnummer der internen Wagyu-Bibliothek zurück.

## Synopsis

text **PostGIS\_Wagyu\_Version()**;

## Beschreibung

Gibt die Versionsnummer der internen Wagyu-Bibliothek zurück, oder NULL wenn die Wagyu-Unterstützung nicht aktiviert ist.

## Beispiele

```
SELECT PostGIS_Wagyu_Version();
 postgis_wagyu_version
-----
 0.5.0 (Internal)
(1 row)
```

## Siehe auch

[PostGIS\\_Full\\_Version](#), [PostGIS\\_GEOS\\_Version](#), [PostGIS\\_PROJ\\_Version](#), [PostGIS\\_Lib\\_Version](#), [PostGIS\\_LibXML\\_Version](#), [PostGIS\\_Version](#)

## 7.21.13 PostGIS\_Scripts\_Build\_Date

PostGIS\_Scripts\_Build\_Date — Gibt das Erstellungsdatum der PostGIS-Skripte zurück.

## Synopsis

```
text PostGIS_Scripts_Build_Date();
```

## Beschreibung

Gibt das Erstellungsdatum der PostGIS-Skripte zurück.

Verfügbarkeit: 1.0.0RC1

## Beispiele

```
SELECT PostGIS_Scripts_Build_Date();
 postgis_scripts_build_date
-----
2023-06-22 03:56:11
(1 row)
```

## Siehe auch

[PostGIS\\_Full\\_Version](#), [PostGIS\\_GEOS\\_Version](#), [PostGIS\\_Lib\\_Version](#), [PostGIS\\_LibXML\\_Version](#), [PostGIS\\_Version](#)

## 7.21.14 PostGIS\_Scripts\_Installed

PostGIS\_Scripts\_Installed — Gibt die Version der in dieser Datenbank installierten PostGIS-Skripte zurück.

## Synopsis

```
text PostGIS_Scripts_Installed();
```

## Beschreibung

Gibt die Version der in dieser Datenbank installierten PostGIS-Skripte zurück.



### Note

Wenn die Ausgabe dieser Funktion nicht mit der Ausgabe von [PostGIS\\_Scripts\\_Released](#) übereinstimmt, haben Sie wahrscheinlich versäumt, eine bestehende Datenbank ordnungsgemäß zu aktualisieren. Weitere Informationen finden Sie im Abschnitt [Upgrading](#).

Verfügbarkeit: 0.9.0

## Beispiele

```
SELECT PostGIS_Scripts_Installed();
 postgis_scripts_installed
-----
3.4.0dev 3.3.0rc2-993-g61bdf43a7
(1 row)
```

**Siehe auch**

[PostGIS\\_Full\\_Version](#), [PostGIS\\_Scripts\\_Released](#), [PostGIS\\_Version](#)

**7.21.15 PostGIS\_Scripts\_Released**

`PostGIS_Scripts_Released` — Gibt die Versionsnummer des Skripts `postgis.sql` zurück, das mit der installierten PostGIS-Bibliothek veröffentlicht wurde.

**Synopsis**

```
text PostGIS_Scripts_Released();
```

**Beschreibung**

Gibt die Versionsnummer des Skripts `postgis.sql` zurück, das mit der installierten PostGIS-Bibliothek veröffentlicht wurde.

**Note**

Ab Version 1.1.0 gibt diese Funktion denselben Wert zurück wie [PostGIS\\_Lib\\_Version](#). Aus Gründen der Abwärtskompatibilität wurde diese Funktion beibehalten.

Verfügbarkeit: 0.9.0

**Beispiele**

```
SELECT PostGIS_Scripts_Released() ;
   postgis_scripts_released
-----
 3.4.0dev 3.3.0rc2-993-g61bdf43a7
(1 row)
```

**Siehe auch**

[PostGIS\\_Full\\_Version](#), [PostGIS\\_Scripts\\_Installed](#), [PostGIS\\_Lib\\_Version](#)

**7.21.16 PostGIS\_Version**

`PostGIS_Version` — Gibt die PostGIS-Versionsnummer und die Kompileroptionen zurück.

**Synopsis**

```
text PostGIS_Version();
```

**Beschreibung**

Gibt die PostGIS-Versionsnummer und die Kompileroptionen zurück.

## Beispiele

```
SELECT PostGIS_Version();
               postgis_version
-----
 3.4 USE_GEOS=1 USE_PROJ=1 USE_STATS=1
(1 row)
```

## Siehe auch

[PostGIS\\_Full\\_Version](#), [PostGIS\\_GEOS\\_Version](#), [PostGIS\\_Lib\\_Version](#), [PostGIS\\_LibXML\\_Version](#), [PostGIS\\_PROJ\\_Version](#)

## 7.22 PostGIS Grand Unified Custom Variables (GUCs)

### 7.22.1 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

[Chapter 8](#)

### 7.22.2 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](#)

### 7.22.3 `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.

Aktiviere 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.
- `VSI_CURL` 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**

To set and reset `postgis.gdal_enabled_drivers` for current session

```
SET postgis.gdal_enabled_drivers = 'ENABLE_ALL';
SET postgis.gdal_enabled_drivers = default;
```

Set for all new connections to a specific database to specific drivers

```
ALTER DATABASE mygisdb SET postgis.gdal_enabled_drivers TO 'GTiff PNG JPEG';
```

Setting for whole database cluster to enable all drivers. Requires super user access. Also note that database, session, and user settings override this.

```
--writes to postgres.auto.conf
ALTER SYSTEM SET postgis.gdal_enabled_drivers TO 'ENABLE_ALL';
--Reloads postgres conf
SELECT pg_reload_conf();
```

**Siehe auch**

[ST\\_FromGDALRaster](#), [ST\\_AsGDALRaster](#), [ST\\_AsTIFF](#), [ST\\_AsPNG](#), [ST\\_AsJPEG](#), [postgis.enable\\_outdb\\_rasters](#)

**7.22.4 postgis.enable\_outdb\_rasters**

`postgis.enable_outdb_rasters` — Eine boolesche Konfigurationsmöglichkeit um den Zugriff auf out-db Rasterbänder zu ermöglichen

## Beschreibung

Eine boolesche Konfigurationsmöglichkeit um den Zugriff auf out-db Rasterbänder zu ermöglichen. Diese Option kann in der PostgreSQL Konfigurationsdatei "postgresql.conf" gesetzt werden. Kann aber auch pro Verbindung oder pro Transaktion gesetzt werden.

Der Ausgangswert von `postgis.enable_outdb_rasters` kann auch beim Startprozess von PostgreSQL gesetzt werden, nämlich durch die Übergabe der Umgebungsvariablen `POSTGIS_ENABLE_OUTDB_RASTERS`, welche ungleich null sein muss.



### Note

Auch wenn `postgis.enable_outdb_rasters` `True` ist, bestimmt die GUC `postgis.enable_outdb_rasters` die zugänglichen Rasterformate.



### Note

Bei der Standardinstallation von PostGIS ist `postgis.enable_outdb_rasters` auf `False` gesetzt.

Verfügbarkeit: 2.2.0

## Beispiele

`postgis.enable_outdb_rasters` setzen oder zurücksetzen

```
SET postgis.enable_outdb_rasters TO True;
SET postgis.enable_outdb_rasters = default;
SET postgis.enable_outdb_rasters = True;
SET postgis.enable_outdb_rasters = False;
```

Set for all new connections to a specific database

```
ALTER DATABASE gisdb SET postgis.enable_outdb_rasters = true;
```

Setting for whole database cluster. Requires super user access. Also note that database, session, and user settings override this.

```
--writes to postgres.auto.conf
ALTER SYSTEM SET postgis.enable_outdb_rasters = true;
--Reloads postgres conf
SELECT pg_reload_conf();
```

## Siehe auch

[postgis.gdal\\_enabled\\_drivers](#) [postgis.gdal\\_vsi\\_options](#)

### 7.22.5 postgis.gdal\_vsi\_options

`postgis.gdal_vsi_options` — Eine boolesche Konfigurationsmöglichkeit um den Zugriff auf out-db Rasterbänder zu ermöglichen

## Beschreibung

**Konfigurationsoptionen** steuern z.B. wie viel Platz GDAL dem lokalen Datencache zuweist, ob Übersichten gelesen werden sollen und welche Zugriffsschlüssel für entfernte Out-DB-Datenquellen verwendet werden sollen.

Verfügbarkeit: 3.2.0



## Beispiele

`postgis.enable_outdb_rasters` setzen oder zurücksetzen

```
SET postgis.gdal_vsi_options = 'AWS_ACCESS_KEY_ID=xxxxxxxxxxxxxxxxx AWS_SECRET_ACCESS_KEY= ↵  
YYYYYYYYYYYYYYYYYYYYYYYYYYYYYYY';
```

Setzen Sie `postgis.gdal_vsi_options` nur für die aktuelle Transaktion mit dem Schlüsselwort `LOCAL`:

```
SET LOCAL postgis.gdal_vsi_options = 'AWS_ACCESS_KEY_ID=xxxxxxxxxxxxxxxxx ↵  
AWS_SECRET_ACCESS_KEY=YYYYYYYYYYYYYYYYYYYYYYYYYYYYYYY';
```

## Siehe auch

[postgis.enable\\_outdb\\_rasters](#) [postgis.gdal\\_enabled\\_drivers](#)

## 7.23 Funktionen zur Fehlersuche

### 7.23.1 PostGIS\_AddBBox

`PostGIS_AddBBox` — Fügt der Geometrie ein umschreibendes Rechteck bei.

#### Synopsis

```
geometry PostGIS_AddBBox(geometry geomA);
```

#### Beschreibung

Fügt der Geometrie ein umschreibendes Rechteck bei. Dies beschleunigt Abfragen, die sich auf die umschreibenden Rechtecke beziehen, erhöht aber die Größe der Geometrie.



#### Note

Die umschreibenden Rechtecke werden üblicherweise automatisch der Geometrie beigefügt, so dass es nicht nötig ist dies händisch zu tun, außer das umschreibende Rechteck wurde irgendwie beschädigt, oder Sie verwenden eine alte Installation bei der die umschreibenden Rechtecke noch fehlten. Falls dies der Fall ist, müssen Sie die alten löschen und neu hinzufügen.



Diese Methode unterstützt kreisförmige Strings und Kurven.

## Beispiele

```
UPDATE sometable  
SET geom = PostGIS_AddBBox(geom)  
WHERE PostGIS_HasBBox(geom) = false;
```

## Siehe auch

[PostGIS\\_DropBBox](#), [PostGIS\\_HasBBox](#)

### 7.23.2 PostGIS\_DropBBox

PostGIS\_DropBBox — Löscht die umschreibenden Rechtecke der Geometrie.

#### Synopsis

```
geometry PostGIS_DropBBox(geometry geomA);
```

#### Beschreibung

Löscht die umschreibenden Rechtecke der Geometrie. Dies verringert die Größe der Geometrie, macht aber die auf umschreibenden Rechtecken aufbauenden Abfragen langsamer. Wird auch zum Löschen eines beschädigten umschreibenden Rechtecks benutzt. Ein guter Hinweis auf ein beschädigtes umschreibendes Rechteck ist, wenn `ST_Intersects` und andere Abfragen über räumliche Beziehungen, Geometrien auslassen die eigentlich TRUE zurückgeben sollten.

#### Note



Umschreibende Rechtecke werden üblicherweise automatisch der Geometrie beigefügt, um die Geschwindigkeit von Abfragen zu steigern. Somit ist es nicht nötig dies händisch zu tun, außer das Umgebungsrechteck wurde irgendwie beschädigt, oder Sie verwenden eine alte Installation bei der die umschreibenden Rechtecke noch fehlten. Falls dies der Fall ist, müssen Sie die alten löschen und neu hinzufügen. Diese Art der Beschädigung wurde in den Versionen 8.3-8.3.6 beobachtet. Der Grund war, dass die zwischengespeicherten umschreibenden Rechtecke nicht immer neu berechnet wurden, wenn sich die Geometrie geändert hat und das Upgraden auf eine neuere Version ohne einen PostgreSQL-Dump erfolgte. Dies kann händisch korrigiert werden, indem man die umschreibenden Rechtecke (BBox) wie unten angeführt neu hinzufügt, oder einen PostgreSQL-Dump einspielt.



Diese Methode unterstützt kreisförmige Strings und Kurven.

#### Beispiele

```
--This example drops bounding boxes where the cached box is not correct
--The force to ST_AsBinary before applying Box2D forces a ↔
  recalculation of the box, and Box2D applied to the table ↔
  geometry always
-- returns the cached bounding box.
UPDATE sometable
SET geom = PostGIS_DropBBox(geom)
WHERE Not (Box2D(ST_AsBinary(geom)) = Box2D(geom));

UPDATE sometable
SET geom = PostGIS_AddBBox(geom)
WHERE Not PostGIS_HasBBOX(geom);
```

#### Siehe auch

[PostGIS\\_AddBBox](#), [PostGIS\\_HasBBox](#), [Box2D](#)

### 7.23.3 PostGIS\_HasBBox

PostGIS\_HasBBox — Gibt TRUE zurück, wenn die BBox der Geometrie zwischengespeichert ist, andernfalls wird FALSE zurückgegeben.

## Synopsis

boolean **PostGIS\_HasBBox**(geometry geomA);

## Beschreibung

Gibt TRUE zurück, wenn die BBox der Geometrie zwischengespeichert ist, sonst FALSE. Benutzen Sie bitte **PostGIS\_AddBBox** und **PostGIS\_DropBBox** um das Zwischenspeichern zu steuern.



Diese Methode unterstützt kreisförmige Strings und Kurven.

## Beispiele

```
SELECT geom
FROM sometable WHERE PostGIS_HasBBox(geom) = false;
```

## Siehe auch

**PostGIS\_AddBBox**, **PostGIS\_DropBBox**

## Chapter 8

# Referenz der SFCGAL-Funktionen

SFCGAL ist eine C++-Wrapper-Bibliothek für CGAL, die erweiterte räumliche 2D- und 3D-Funktionen bietet. Aus Gründen der Robustheit haben die Geometriekoordinaten eine exakte rationale Zahlendarstellung.

Installationsanweisungen für die Bibliothek sind auf der SFCGAL-Homepage zu finden (<http://www.sfcgal.org>). Um die Funktionen zu aktivieren, verwenden Sie Erstellen der Erweiterung `postgis_sfcgal`.

### 8.1 Verwaltungsfunktionen der SFCGAL

#### 8.1.1 `postgis_sfcgal_version`

`postgis_sfcgal_version` — Gibt die verwendete Version von SFCGAL zurück

##### Synopsis

```
text postgis_sfcgal_version(void);
```

##### Beschreibung

Gibt die verwendete Version von SFCGAL zurück

Verfügbarkeit: 2.1.0



Diese Methode benötigt ein SFCGAL-Backend.

##### Siehe auch

[postgis\\_sfcgal\\_full\\_version](#)

#### 8.1.2 `postgis_sfcgal_full_version`

`postgis_sfcgal_full_version` — Liefert die vollständige Version von SFCGAL, einschließlich der CGAL- und Boost-Versionen

##### Synopsis

```
text postgis_sfcgal_full_version(void);
```

---

### Beschreibung

Liefert die vollständige Version von SFCGAL, einschließlich der CGAL- und Boost-Versionen

Verfügbarkeit: 3.3.0



Diese Methode benötigt ein SFCGAL-Backend.

### Siehe auch

[postgis\\_sfcgal\\_version](#)

## 8.2 SFCGAL-Accessoren und -Setzer

### 8.2.1 CG\_ForceLHR

CG\_ForceLHR — LHR-Ausrichtung erzwingen

#### Synopsis

```
geometry CG_ForceLHR(geometry geom);
```

#### Beschreibung

Verfügbarkeit: 3.5.0



Diese Methode benötigt ein SFCGAL-Backend.



Diese Funktion unterstützt 3d und lässt den Z-Index nicht fallen.



Diese Funktion unterstützt polyedrische Flächen.



Diese Funktion unterstützt Dreiecke und dreieckige unregelmäßige Netzoberflächen (TIN).

### 8.2.2 CG\_IsPlanar

CG\_IsPlanar — Prüfen, ob eine Fläche planar ist oder nicht

#### Synopsis

```
boolean CG_IsPlanar(geometry geom);
```

#### Beschreibung

Verfügbarkeit: 3.5.0



Diese Methode benötigt ein SFCGAL-Backend.



Diese Funktion unterstützt 3d und lässt den Z-Index nicht fallen.



Diese Funktion unterstützt polyedrische Flächen.



Diese Funktion unterstützt Dreiecke und dreieckige unregelmäßige Netzoberflächen (TIN).

### 8.2.3 CG\_IsSolid

CG\_IsSolid — Prüfen, ob die Geometrie ein Solid ist. Es wird keine Gültigkeitsprüfung durchgeführt.

#### Synopsis

```
boolean CG_IsSolid(geometry geom1);
```

#### Beschreibung

Verfügbarkeit: 3.5.0



Diese Methode benötigt ein SFCGAL-Backend.



Diese Funktion unterstützt 3d und lässt den Z-Index nicht fallen.



Diese Funktion unterstützt polyedrische Flächen.



Diese Funktion unterstützt Dreiecke und dreieckige unregelmäßige Netzoberflächen (TIN).

### 8.2.4 CG\_MakeSolid

CG\_MakeSolid — Gießen Sie die Geometrie in einen Körper. Es wird keine Prüfung durchgeführt. Um ein gültiges Solid zu erhalten, muss die Eingabegeometrie eine geschlossene polyedrische Fläche oder ein geschlossenes TIN sein.

#### Synopsis

```
geometry CG_MakeSolid(geometry geom1);
```

#### Beschreibung

Verfügbarkeit: 3.5.0



Diese Methode benötigt ein SFCGAL-Backend.



Diese Funktion unterstützt 3d und lässt den Z-Index nicht fallen.



Diese Funktion unterstützt polyedrische Flächen.



Diese Funktion unterstützt Dreiecke und dreieckige unregelmäßige Netzoberflächen (TIN).

### 8.2.5 CG\_Orientation

CG\_Orientation — Bestimmung der Oberflächenausrichtung

#### Synopsis

```
integer CG_Orientation(geometry geom);
```

## Beschreibung

Die Funktion gilt nur für Polygone. Sie gibt -1 zurück, wenn das Polygon gegen den Uhrzeigersinn orientiert ist, und 1, wenn das Polygon im Uhrzeigersinn orientiert ist.

Verfügbarkeit: 3.5.0



Diese Methode benötigt ein SFCGAL-Backend.



Diese Funktion unterstützt 3d und lässt den Z-Index nicht fallen.

## 8.2.6 CG\_Area

CG\_Area — Calculates the area of a geometry

### Synopsis

```
double precision CG_Area( geometry geom );
```

### Beschreibung

Calculates the area of a geometry.

Performed by the SFCGAL module



#### Note

NOTE: this function returns a double precision value representing the area.

Verfügbarkeit: 3.5.0



Diese Methode benötigt ein SFCGAL-Backend.

### Beispiele mit dem geometrischen Datentyp

```
SELECT CG_Area('Polygon ((0 0, 0 5, 5 5, 5 0, 0 0), (1 1, 2 1, 2 2, 1 2, 1 1), (3 3, 4 3, 4 4, 3 4, 3 3))');
   cg_area
   -----
        25
(1 row)
```

### Siehe auch

[ST\\_3DArea](#), [ST\\_Area](#)

## 8.2.7 CG\_3DArea

CG\_3DArea — Berechnet die Fläche von 3D-Oberflächengeometrien. Gibt 0 für Solids zurück.

## Synopsis

```
float CG_3DArea(geometry geom1);
```

## Beschreibung

Verfügbarkeit: 3.5.0



Diese Methode benötigt ein SFCGAL-Backend.



Diese Methode setzt die SQL/MM-Spezifikation um. SQL-MM IEC 13249-3: 8.1, 10.5



Diese Funktion unterstützt 3d und lässt den Z-Index nicht fallen.



Diese Funktion unterstützt polyedrische Flächen.



Diese Funktion unterstützt Dreiecke und dreieckige unregelmäßige Netzoberflächen (TIN).

## Beispiele

Hinweis: Standardmäßig ist eine aus WKT erstellte PolyhedralSurface eine Flächengeometrie, kein Solid. Sie hat daher einen Oberflächenbereich. Nach der Umwandlung in einen Festkörper gibt es keine Fläche mehr.

```
SELECT CG_3DArea(geom) As cube_surface_area,
       CG_3DArea(CG_MakeSolid(geom)) As solid_surface_area
FROM (SELECT 'POLYHEDRALSURFACE( ((0 0 0, 0 0 1, 0 1 1, 0 1 0, 0 0 0)),
  ((0 0 0, 0 1 0, 1 1 0, 1 0 0, 0 0 0)),
  ((0 0 0, 1 0 0, 1 0 1, 0 0 1, 0 0 0)),
  ((1 1 0, 1 1 1, 1 0 1, 1 0 0, 1 1 0)),
  ((0 1 0, 0 1 1, 1 1 1, 1 1 0, 0 1 0)),
  ((0 0 1, 1 0 1, 1 1 1, 0 1 1, 0 0 1)) )'::geometry) As f(geom);

cube_surface_area | solid_surface_area
-----+-----
6 | 0
```

## Siehe auch

[CG\\_Area](#), [CG\\_MakeSolid](#), [CG\\_IsSolid](#), [CG\\_Area](#)

## 8.2.8 CG\_Volume

**CG\_Volume** — Berechnet das Volumen eines 3D-Volumens. Bei Anwendung auf (auch geschlossene) Flächengeometrien wird 0 zurückgegeben.

## Synopsis

```
float CG_Volume(geometry geom1);
```



## Beschreibung

Verfügbarkeit: 3.5.0



Diese Methode benötigt ein SFCGAL-Backend.



Diese Funktion unterstützt 3d und lässt den Z-Index nicht fallen.



Diese Funktion unterstützt polyedrische Flächen.



Diese Funktion unterstützt Dreiecke und dreieckige unregelmäßige Netzoberflächen (TIN).



Diese Methode setzt die SQL/MM-Spezifikation um. SQL-MM IEC 13249-3: 9.1 (same as CG\_3DVolume)

## Beispiel

When closed surfaces are created with WKT, they are treated as areal rather than solid. To make them solid, you need to use [CG\\_MakeSolid](#). Areal geometries have no volume. Here is an example to demonstrate.

```
SELECT CG_Volume(geom) As cube_surface_vol,
       CG_Volume(CG_MakeSolid(geom)) As solid_surface_vol
FROM (SELECT 'POLYHEDRALSURFACE( ((0 0 0, 0 0 1, 0 1 1, 0 1 0, 0 0 0)),
  ((0 0 0, 0 1 0, 1 1 0, 1 0 0, 0 0 0)),
  ((0 0 0, 1 0 0, 1 0 1, 0 0 1, 0 0 0)),
  ((1 1 0, 1 1 1, 1 0 1, 1 0 0, 1 1 0)),
  ((0 1 0, 0 1 1, 1 1 1, 1 1 0, 0 1 0)),
  ((0 0 1, 1 0 1, 1 1 1, 0 1 1, 0 0 1)) )'::geometry) As f(geom);
```

cube_surface_vol	solid_surface_vol
0	1

## Siehe auch

[CG\\_3DArea](#), [CG\\_MakeSolid](#), [CG\\_IsSolid](#)

## 8.2.9 ST\_ForceLHR

ST\_ForceLHR — LHR-Ausrichtung erzwingen

### Synopsis

geometry **ST\_ForceLHR**(geometry geom);

### Beschreibung



#### Warning

**ST\_ForceLHR** is deprecated as of 3.5.0. Use [CG\\_ForceLHR](#) instead.

Verfügbarkeit: 2.1.0



Diese Methode benötigt ein SFCGAL-Backend.



Diese Funktion unterstützt 3d und lässt den Z-Index nicht fallen.



Diese Funktion unterstützt polyedrische Flächen.



Diese Funktion unterstützt Dreiecke und dreieckige unregelmäßige Netzoberflächen (TIN).

## 8.2.10 ST\_IsPlanar

ST\_IsPlanar — Prüfen, ob eine Fläche planar ist oder nicht

### Synopsis

boolean **ST\_IsPlanar**(geometry geom);

### Beschreibung

---



#### Warning

**ST\_IsPlanar** is deprecated as of 3.5.0. Use **CG\_IsPlanar** instead.

---

Verfügbarkeit: 2.2.0: Dies war in 2.1.0 dokumentiert, wurde aber versehentlich in der Version 2.1 ausgelassen.



Diese Methode benötigt ein SFCGAL-Backend.



Diese Funktion unterstützt 3d und lässt den Z-Index nicht fallen.



Diese Funktion unterstützt polyedrische Flächen.



Diese Funktion unterstützt Dreiecke und dreieckige unregelmäßige Netzoberflächen (TIN).

## 8.2.11 ST\_IsSolid

ST\_IsSolid — Prüfen, ob die Geometrie ein Solid ist. Es wird keine Gültigkeitsprüfung durchgeführt.

### Synopsis

boolean **ST\_IsSolid**(geometry geom1);

### Beschreibung

---



#### Warning

**ST\_IsSolid** is deprecated as of 3.5.0. Use **CG\_IsSolid** instead.

---

Verfügbarkeit: 2.2.0



Diese Methode benötigt ein SFCGAL-Backend.



Diese Funktion unterstützt 3d und lässt den Z-Index nicht fallen.



Diese Funktion unterstützt polyedrische Flächen.



Diese Funktion unterstützt Dreiecke und dreieckige unregelmäßige Netzoberflächen (TIN).

## 8.2.12 ST\_MakeSolid

`ST_MakeSolid` — Gießen Sie die Geometrie in einen Körper. Es wird keine Prüfung durchgeführt. Um ein gültiges Solid zu erhalten, muss die Eingabegeometrie eine geschlossene polyedrische Fläche oder ein geschlossenes TIN sein.

### Synopsis

```
geometry ST_MakeSolid(geometry geom1);
```

### Beschreibung

---



#### Warning

`ST_MakeSolid` is deprecated as of 3.5.0. Use `CG_MakeSolid` instead.

---

Verfügbarkeit: 2.2.0



Diese Methode benötigt ein SFCGAL-Backend.



Diese Funktion unterstützt 3d und lässt den Z-Index nicht fallen.



Diese Funktion unterstützt polyedrische Flächen.



Diese Funktion unterstützt Dreiecke und dreieckige unregelmäßige Netzoberflächen (TIN).

## 8.2.13 ST\_Orientation

`ST_Orientation` — Bestimmung der Oberflächenausrichtung

### Synopsis

```
integer ST_Orientation(geometry geom);
```

### Beschreibung

---



#### Warning

`ST_Orientation` is deprecated as of 3.5.0. Use `CG_Orientation` instead.

---

Die Funktion gilt nur für Polygone. Sie gibt -1 zurück, wenn das Polygon gegen den Uhrzeigersinn orientiert ist, und 1, wenn das Polygon im Uhrzeigersinn orientiert ist.

Verfügbarkeit: 2.1.0



Diese Methode benötigt ein SFCGAL-Backend.



Diese Funktion unterstützt 3d und lässt den Z-Index nicht fallen.

## 8.2.14 ST\_3DArea

ST\_3DArea — Berechnet die Fläche von 3D-Oberflächengeometrien. Gibt 0 für Solids zurück.

### Synopsis

```
floatST_3DArea(geometry geom1);
```

### Beschreibung



#### Warning

ST\_3DArea is deprecated as of 3.5.0. Use [CG\\_3DArea](#) instead.

Verfügbarkeit: 2.1.0



Diese Methode benötigt ein SFCGAL-Backend.



Diese Methode setzt die SQL/MM-Spezifikation um. SQL-MM IEC 13249-3: 8.1, 10.5



Diese Funktion unterstützt 3d und lässt den Z-Index nicht fallen.



Diese Funktion unterstützt polyedrische Flächen.



Diese Funktion unterstützt Dreiecke und dreieckige unregelmäßige Netzoberflächen (TIN).

### Beispiele

Hinweis: Standardmäßig ist eine aus WKT erstellte PolyhedralSurface eine Flächengeometrie, kein Solid. Sie hat daher einen Oberflächenbereich. Nach der Umwandlung in einen Festkörper gibt es keine Fläche mehr.

```
SELECT ST_3DArea(geom) As cube_surface_area,
       ST_3DArea(ST_MakeSolid(geom)) As solid_surface_area
FROM (SELECT 'POLYHEDRALSURFACE( ((0 0 0, 0 0 1, 0 1 1, 0 1 0, 0 0 0)),
  ((0 0 0, 0 1 0, 1 1 0, 1 0 0, 0 0 0)),
  ((0 0 0, 1 0 0, 1 0 1, 0 0 1, 0 0 0)),
  ((1 1 0, 1 1 1, 1 0 1, 1 0 0, 1 1 0)),
  ((0 1 0, 0 1 1, 1 1 1, 1 1 0, 0 1 0)),
  ((0 0 1, 1 0 1, 1 1 1, 0 1 1, 0 0 1)) )'::geometry) As f(geom);

cube_surface_area | solid_surface_area
-----+-----
6 | 0
```

**Siehe auch**

[ST\\_Area](#), [ST\\_ConcaveHull](#), [ST\\_Dump](#), [ST\\_Tessellate](#)

**8.2.15 ST\_Volume**

`ST_Volume` — Berechnet das Volumen eines 3D-Volumens. Bei Anwendung auf (auch geschlossene) Flächengeometrien wird 0 zurückgegeben.

**Synopsis**

```
float ST_Volume(geometry geom1);
```

**Beschreibung****Warning**

`ST_Volume` is deprecated as of 3.5.0. Use `CG_Volume` instead.

Verfügbarkeit: 2.2.0



Diese Methode benötigt ein SFCGAL-Backend.



Diese Funktion unterstützt 3d und lässt den Z-Index nicht fallen.



Diese Funktion unterstützt polyedrische Flächen.



Diese Funktion unterstützt Dreiecke und dreieckige unregelmäßige Netzoberflächen (TIN).



Diese Methode setzt die SQL/MM-Spezifikation um. SQL-MM IEC 13249-3: 9.1 (identisch mit `ST_3DVolume`)

**Beispiel**

Wenn geschlossene Flächen mit WKT erstellt werden, werden sie als flächig und nicht als massiv behandelt. Um sie zu Volumenkörpern zu machen, müssen Sie `ST_MakeSolid` verwenden. Flächenhafte Geometrien haben kein Volumen. Hier ist ein Beispiel zur Veranschaulichung.

```
SELECT ST_Volume(geom) As cube_surface_vol,
       ST_Volume(ST_MakeSolid(geom)) As solid_surface_vol
FROM (SELECT 'POLYHEDRALSURFACE( ((0 0 0, 0 0 1, 0 1 1, 0 1 0, 0 0 0)),
  ((0 0 0, 0 1 0, 1 1 0, 1 0 0, 0 0 0)),
  ((0 0 0, 1 0 0, 1 0 1, 0 0 1, 0 0 0)),
  ((1 1 0, 1 1 1, 1 0 1, 1 0 0, 1 1 0)),
  ((0 1 0, 0 1 1, 1 1 1, 1 1 0, 0 1 0)),
  ((0 0 1, 1 0 1, 1 1 1, 0 1 1, 0 0 1)) )'::geometry) As f(geom);

cube_surface_vol | solid_surface_vol
-----+-----
0 | 1
```

**Siehe auch**

[ST\\_3DArea](#), [ST\\_VoronoiLines](#), [ST\\_Collect](#)

## 8.3 SFCGAL-Verarbeitung und Beziehungsfunktionen

### 8.3.1 CG\_Intersection

CG\_Intersection — Computes the intersection of two geometries

#### Synopsis

```
geometry CG_Intersection( geometry geomA , geometry geomB );
```

#### Beschreibung

Computes the intersection of two geometries.

Performed by the SFCGAL module



#### Note

NOTE: this function returns a geometry representing the intersection.

Verfügbarkeit: 3.5.0



Diese Methode benötigt ein SFCGAL-Backend.

#### Beispiele mit dem geometrischen Datentyp

```
SELECT ST_AsText(CG_Intersection('LINESTRING(0 0, 5 5)', 'LINESTRING(5 0, 0 5)'));
      cg_intersection
      -----
      POINT(2.5 2.5)
      (1 row)
```

#### Siehe auch

[ST\\_3DIntersection](#), [ST\\_Intersection](#)

### 8.3.2 CG\_Intersects

CG\_Intersects — Prüft, ob sich zwei Geometrien schneiden (sie haben mindestens einen Punkt gemeinsam)

#### Synopsis

```
boolean CG_Intersects( geometry geomA , geometry geomB );
```

## Beschreibung

Gibt `true` zurück, wenn sich zwei Geometrien überschneiden. Geometrien überschneiden sich, wenn sie einen Punkt gemeinsam haben.

Performed by the SFCGAL module

**Note**

HINWEIS: Dies ist die "zulässige" Version, die einen booleschen Wert und keine ganze Zahl zurückgibt.

Verfügbarkeit: 3.5.0



Diese Methode benötigt ein SFCGAL-Backend.



Diese Funktion unterstützt Dreiecke und dreieckige unregelmäßige Netzoberflächen (TIN).

## Beispiele mit dem geometrischen Datentyp

```
SELECT CG_Intersects('POINT(0 0)::geometry, 'LINESTRING ( 2 0, 0 2 ) '::geometry);
cg_intersects
-----
f
(1 row)
SELECT CG_Intersects('POINT(0 0)::geometry, 'LINESTRING ( 0 0, 0 2 ) '::geometry);
cg_intersects
-----
t
(1 row)
```

## Siehe auch

[CG\\_3DIntersects](#), [ST\\_3DIntersects](#), [ST\\_Intersects](#), [ST\\_Disjoint](#)

### 8.3.3 CG\_3DIntersects

CG\_3DIntersects — Tests if two 3D geometries intersect

#### Synopsis

```
boolean CG_3DIntersects( geometry geomA , geometry geomB );
```

#### Beschreibung

Tests if two 3D geometries intersect. 3D geometries intersect if they have any point in common in the three-dimensional space.

Performed by the SFCGAL module

**Note**

HINWEIS: Dies ist die "zulässige" Version, die einen booleschen Wert und keine ganze Zahl zurückgibt.

Verfügbarkeit: 3.5.0



Diese Methode benötigt ein SFCGAL-Backend.



Diese Funktion unterstützt Dreiecke und dreieckige unregelmäßige Netzoberflächen (TIN).

### Beispiele mit dem geometrischen Datentyp

```
SELECT CG_3DIntersects('POINT(1.2 0.1 0)', 'POLYHEDRALSURFACE(((0 0 0,0.5 0.5 0,1 0 0,1 1 0,0 1 0,0 0 0)), ((1 0 0,2 0 0,2 1 0,1 1 0,1 0 0), (1.2 0.2 0,1.2 0.8 0,1.8 0.8 0,1.8 0.2 0,1.2 0.2 0)))');
      cg_3dintersects
      -----
      t
      (1 row)
```

### Siehe auch

[CG\\_Intersects](#), [ST\\_3DIntersects](#), [ST\\_Intersects](#), [ST\\_Disjoint](#)

## 8.3.4 CG\_Difference

CG\_Difference — Computes the geometric difference between two geometries

### Synopsis

geometry **CG\_Difference**( geometry geomA , geometry geomB );

### Beschreibung

Computes the geometric difference between two geometries. The resulting geometry is a set of points that are present in geomA but not in geomB.

Performed by the SFCGAL module



#### Note

NOTE: this function returns a geometry.

Verfügbarkeit: 3.5.0



Diese Methode benötigt ein SFCGAL-Backend.



Diese Funktion unterstützt Dreiecke und dreieckige unregelmäßige Netzoberflächen (TIN).

### Beispiele mit dem geometrischen Datentyp

```
SELECT ST_AsText(CG_Difference('POLYGON((0 0, 0 1, 1 1, 1 0, 0 0))'::geometry, 'LINESTRING(0 0, 2 2)'::geometry));
      cg_difference
      -----
      POLYGON((0 0,1 0,1 1,0 1,0 0))
      (1 row)
```



**Siehe auch**

[ST\\_3DDifference](#), [ST\\_Difference](#)

### 8.3.5 ST\_3DDifference

ST\_3DDifference — 3D-Differenz durchführen

**Synopsis**

```
geometry ST_3DDifference(geometry geom1, geometry geom2);
```

**Beschreibung****Warning**

[ST\\_3DDifference](#) is deprecated as of 3.5.0. Use [CG\\_3DDifference](#) instead.

---

Gibt den geometrischen Schwerpunkt einer Geometrie zurück.

Verfügbarkeit: 2.2.0



Diese Methode benötigt ein SFCGAL-Backend.



Diese Methode setzt die SQL/MM-Spezifikation um. SQL-MM IEC 13249-3: 5.1



Diese Funktion unterstützt 3d und lässt den Z-Index nicht fallen.



Diese Funktion unterstützt polyedrische Flächen.



Diese Funktion unterstützt Dreiecke und dreieckige unregelmäßige Netzoberflächen (TIN).

### 8.3.6 CG\_3DDifference

CG\_3DDifference — 3D-Differenz durchführen

**Synopsis**

```
geometry CG_3DDifference(geometry geom1, geometry geom2);
```

**Beschreibung**

Gibt den geometrischen Schwerpunkt einer Geometrie zurück.

Verfügbarkeit: 3.5.0



Diese Methode benötigt ein SFCGAL-Backend.



Diese Methode setzt die SQL/MM-Spezifikation um. SQL-MM IEC 13249-3: 5.1



Diese Funktion unterstützt 3d und lässt den Z-Index nicht fallen.



Diese Funktion unterstützt polyedrische Flächen.

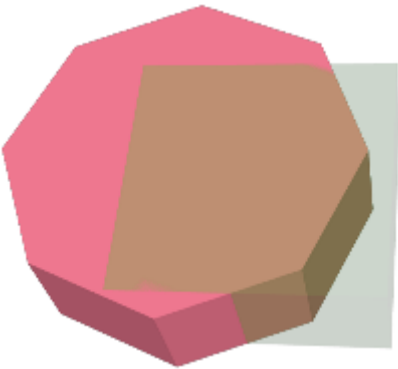
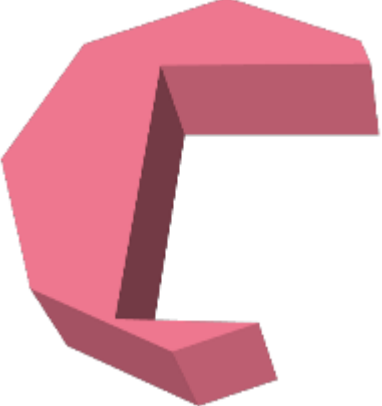


Diese Funktion unterstützt Dreiecke und dreieckige unregelmäßige Netzoberflächen (TIN).

---

## Beispiele

Die 3D-Bilder wurden mit PostGIS [ST\\_AsX3D](#) erzeugt und in HTML mit [X3Dom HTML Javascript Rendering Library](#) gerendert.

<pre>SELECT CG_Extrude(ST_Buffer(↵   ST_GeomFromText('POINT(100 90)'),                         50, '↵ quad_segs=2'),0,0,30) AS geom1,   ↵ CG_Extrude(ST_Buffer(ST_GeomFromText('POINT(80 80)'),                         50, '↵ quad_segs=1'),0,0,30) AS geom2;</pre>  <p><i>Die ursprünglichen 3D-Geometrien werden überlagert. geom2 ist der Teil, der entfernt wird.</i></p>	<pre>SELECT CG_3DDifference(geom1,geom2)       FROM ( SELECT ↵         CG_Extrude(ST_Buffer(ST_GeomFromText('POINT(100 90)'),                         50, '↵ quad_segs=2'),0,0,30) AS geom1,         ↵         CG_Extrude(↵           ST_Buffer(ST_GeomFromText('POINT(80 80)'),                     50, '↵ quad_segs=1'),0,0,30) AS geom2 ) As t;</pre>  <p><i>Was nach dem Entfernen von geom2 übrig ist</i></p>
--	---

### Siehe auch

[CG\\_Extrude](#), [ST\\_AsX3D](#), [CG\\_3DIntersection](#) [CG\\_3DUnion](#)

### 8.3.7 CG\_Distance

**CG\_Distance** — Computes the minimum distance between two geometries

#### Synopsis

```
double precision CG_Distance( geometry geomA , geometry geomB );
```

#### Beschreibung

Computes the minimum distance between two geometries.

Performed by the SFCGAL module



#### Note

NOTE: this function returns a double precision value representing the distance.

Verfügbarkeit: 3.5.0



Diese Methode benötigt ein SFCGAL-Backend.



Diese Funktion unterstützt Dreiecke und dreieckige unregelmäßige Netzoberflächen (TIN).

### Beispiele mit dem geometrischen Datentyp

```
SELECT CG_Distance('LINESTRING(0.0 0.0,-1.0 -1.0)', 'LINESTRING(3.0 4.0,4.0 5.0)');
   cg_distance
   -----
    2.0
(1 row)
```

### Siehe auch

[CG\\_3DDistance](#), [CG\\_Distance](#)

## 8.3.8 CG\_3DDistance

CG\_3DDistance — Computes the minimum 3D distance between two geometries

### Synopsis

double precision **CG\_3DDistance**( geometry geomA , geometry geomB );

### Beschreibung

Computes the minimum 3D distance between two geometries.

Performed by the SFCGAL module



#### Note

NOTE: this function returns a double precision value representing the 3D distance.

Verfügbarkeit: 3.5.0



Diese Methode benötigt ein SFCGAL-Backend.



Diese Funktion unterstützt Dreiecke und dreieckige unregelmäßige Netzoberflächen (TIN).

### Beispiele mit dem geometrischen Datentyp

```
SELECT CG_3DDistance('LINESTRING(-1.0 0.0 2.0,1.0 0.0 3.0)', 'TRIANGLE((-4.0 0.0 1.0,4.0 ↔
   0.0 1.0,0.0 4.0 1.0,-4.0 0.0 1.0))');
   cg_3ddistance
   -----
    1
(1 row)
```

**Siehe auch**

[CG\\_Distance](#), [ST\\_3DDistance](#)

**8.3.9 ST\_3DConvexHull**

ST\_3DConvexHull — Berechnet die konvexe Hülle einer Geometrie.

**Synopsis**

```
geometry ST_3DConvexHull(geometry geom1);
```

**Beschreibung****Warning**

[ST\\_3DConvexHull](#) is deprecated as of 3.5.0. Use [CG\\_3DConvexHull](#) instead.

---

Verfügbarkeit: 3.3.0



Diese Methode benötigt ein SFCGAL-Backend.



Diese Funktion unterstützt 3d und lässt den Z-Index nicht fallen.



Diese Funktion unterstützt polyedrische Flächen.



Diese Funktion unterstützt Dreiecke und dreieckige unregelmäßige Netzoberflächen (TIN).

**8.3.10 CG\_3DConvexHull**

CG\_3DConvexHull — Berechnet die konvexe Hülle einer Geometrie.

**Synopsis**

```
geometry CG_3DConvexHull(geometry geom1);
```

**Beschreibung**

Verfügbarkeit: 3.5.0



Diese Methode benötigt ein SFCGAL-Backend.



Diese Funktion unterstützt 3d und lässt den Z-Index nicht fallen.



Diese Funktion unterstützt polyedrische Flächen.



Diese Funktion unterstützt Dreiecke und dreieckige unregelmäßige Netzoberflächen (TIN).

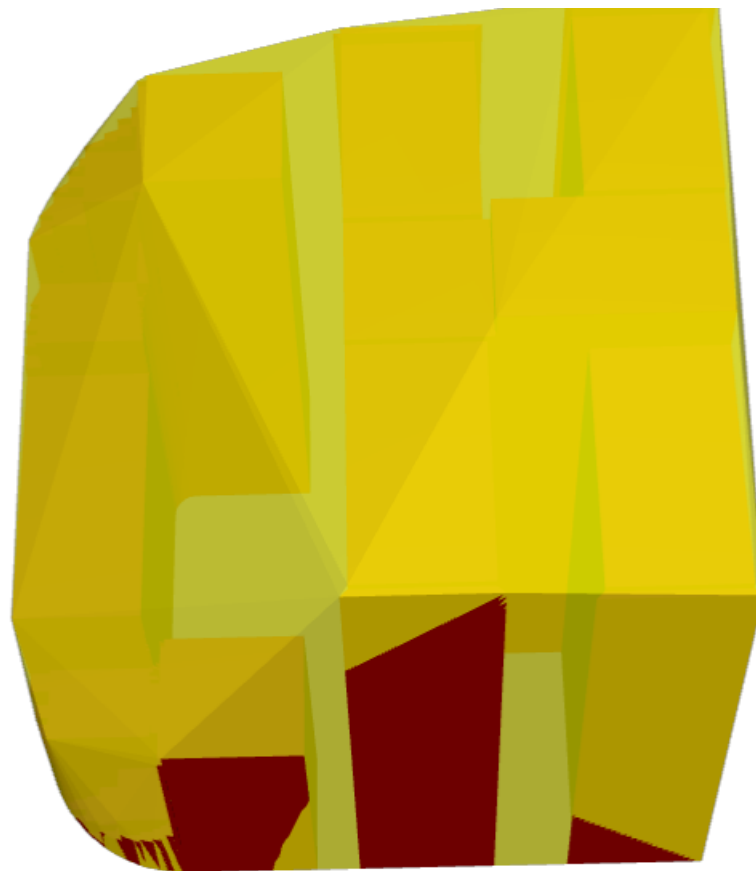
---

**Beispiele**

```
SELECT ST_AsText(CG_3DConvexHull('LINESTRING Z(0 0 5, 1 5 3, 5 7 6, 9 5 3, 5 7 5, 6 3 5) ←
  '::geometry));
```

```
POLYHEDRALSURFACE Z (((1 5 3,9 5 3,0 0 5,1 5 3)),((1 5 3,0 0 5,5 7 6,1 5 3)),((5 7 6,5 7 ←
  5,1 5 3,5 7 6)),((0 0 5,6 3 5,5 7 6,0 0 5)),((6 3 5,9 5 3,5 7 6,6 3 5)),((0 0 5,9 5 3,6 ←
  3 5,0 0 5)),((9 5 3,5 7 5,5 7 6,9 5 3)),((1 5 3,5 7 5,9 5 3,1 5 3)))
```

```
WITH f AS (SELECT i, CG_Extrude(geom, 0,0, i ) AS geom
  FROM ST_Subdivide(ST_Letters('CH'),5) WITH ORDINALITY AS sd(geom,i)
 )
SELECT CG_3DConvexHull(ST_Collect(f.geom) )
FROM f;
```



*Originalgeometrie überlagert mit konvexer 3D-Hülle*

**Siehe auch**

[ST\\_Letters](#), [ST\\_AsX3D](#)

**8.3.11 ST\_3DIntersection**

ST\_3DIntersection — 3D-Schnitte durchführen

## Synopsis

geometry **ST\_3DIntersection**(geometry geom1, geometry geom2);

## Beschreibung



### Warning

**ST\_3DIntersection** is deprecated as of 3.5.0. Use **CG\_3DIntersection** instead.

Gibt eine Geometrie zurück, die der gemeinsame Teil von geom1 und geom2 ist.

Verfügbarkeit: 2.1.0



Diese Methode benötigt ein SFCGAL-Backend.



Diese Methode setzt die SQL/MM-Spezifikation um. SQL-MM IEC 13249-3: 5.1



Diese Funktion unterstützt 3d und lässt den Z-Index nicht fallen.



Diese Funktion unterstützt polyedrische Flächen.



Diese Funktion unterstützt Dreiecke und dreieckige unregelmäßige Netzoberflächen (TIN).

## 8.3.12 CG\_3DIntersection

CG\_3DIntersection — 3D-Schnitte durchführen

## Synopsis

geometry **CG\_3DIntersection**(geometry geom1, geometry geom2);

## Beschreibung

Gibt eine Geometrie zurück, die der gemeinsame Teil von geom1 und geom2 ist.

Verfügbarkeit: 3.5.0



Diese Methode benötigt ein SFCGAL-Backend.



Diese Methode setzt die SQL/MM-Spezifikation um. SQL-MM IEC 13249-3: 5.1



Diese Funktion unterstützt 3d und lässt den Z-Index nicht fallen.



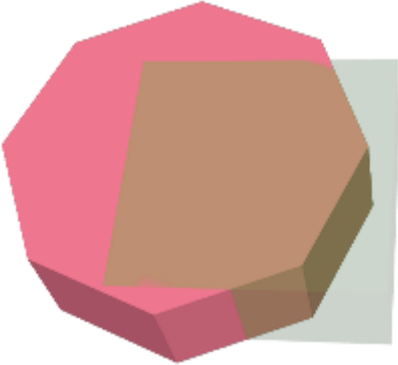
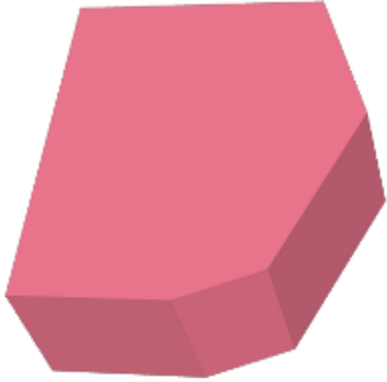
Diese Funktion unterstützt polyedrische Flächen.



Diese Funktion unterstützt Dreiecke und dreieckige unregelmäßige Netzoberflächen (TIN).

## Beispiele

Die 3D-Bilder wurden mit PostGIS **ST\_AsX3D** erzeugt und in HTML mit **X3Dom HTML Javascript Rendering Library** gerendert.

<pre>SELECT CG_Extrude(ST_Buffer(↵     ST_GeomFromText('POINT(100 90)'),     50, '↵ quad_segs=2'),0,0,30) AS geom1,     ↵ CG_Extrude(ST_Buffer(ST_GeomFromText('POINT(80 80)'),     50, '↵ quad_segs=1'),0,0,30) AS geom2;</pre>  <p><i>Die ursprünglichen 3D-Geometrien werden überlagert. geom2 wird halbtransparent dargestellt.</i></p>	<pre>SELECT CG_3DIntersection(geom1,geom2)     FROM (↵ SELECT CG_Extrude(ST_Buffer(ST_GeomFromText('POINT(100 90)'),     50, '↵ quad_segs=2'),0,0,30) AS geom1,     ↵ CG_Extrude(ST_Buffer(ST_GeomFromText('POINT(80 80)'),     50, '↵ quad_segs=1'),0,0,30) AS geom2 ) As t;</pre>  <p><i>Schnittpunkt von geom1 und geom2</i></p>
--	---

### Ein MultiLineString und ein LineString

```
SELECT ST_AsText(CG_3DIntersection(linestring, polygon)) As wkt
FROM ST_GeomFromText('LINESTRING Z (2 2 6,1.5 1.5 7,1 1 8,0.5 0.5 8,0 0 10)') AS ↵
linestring
CROSS JOIN ST_GeomFromText('POLYGON((0 0 8, 0 1 8, 1 1 8, 1 0 8, 0 0 8))') AS polygon;
```

wkt  
-----  
LINESTRING Z (1 1 8,0.5 0.5 8)

### Würfel (geschlossene polyedrische Fläche) und Polygon Z

```
SELECT ST_AsText(CG_3DIntersection(
ST_GeomFromText('POLYHEDRALSURFACE Z( ((0 0 0, 0 0 1, 0 1 1, 0 1 0, 0 0 0)),
((0 0 0, 0 1 0, 1 1 0, 1 0 0, 0 0 0)), ((0 0 0, 1 0 0, 1 0 1, 0 0 1, 0 0 0)),
((1 1 0, 1 1 1, 1 0 1, 1 0 0, 1 1 0)),
((0 1 0, 0 1 1, 1 1 1, 1 1 0, 0 1 0)), ((0 0 1, 1 0 1, 1 1 1, 0 1 1, 0 0 1)) )'),
'POLYGON Z ((0 0 0, 0 0 0.5, 0 0.5 0.5, 0 0.5 0, 0 0 0))'::geometry)
```

```
TIN Z (((0 0 0,0 0 0.5,0 0.5 0.5,0 0 0)),((0 0.5 0,0 0 0,0 0.5 0.5,0 0.5 0)))
```

Die Schnittmenge von 2 Solids, die eine volumetrische Schnittmenge ergeben, ist ebenfalls ein Solid (ST\_Dimension liefert 3)

```
SELECT ST_AsText(CG_3DIntersection( CG_Extrude(ST_Buffer('POINT(10 20)'::geometry,10,1) ↵
,0,0,30),
CG_Extrude(ST_Buffer('POINT(10 20)'::geometry,10,1),2,0,10) ));
```

```
POLYHEDRALSURFACE Z (((13.3333333333333 13.3333333333333 10,20 20 0,20 20 ↵
10,13.3333333333333 13.3333333333333 10)),
((20 20 10,16.6666666666667 23.3333333333333 10,13.3333333333333 13.3333333333333 ↵
10,20 20 10)),
```

```
((20 20 0,16.66666666666667 23.3333333333333 10,20 20 10,20 20 0)),
((13.3333333333333 13.3333333333333 10,10 10 0,20 20 0,13.3333333333333 ←
 13.3333333333333 10)),
((16.6666666666667 23.3333333333333 10,12 28 10,13.3333333333333 13.3333333333333 ←
 10,16.6666666666667 23.3333333333333 10)),
((20 20 0,9.99999999999995 30 0,16.6666666666667 23.3333333333333 10,20 20 0)),
((10 10 0,9.99999999999995 30 0,20 20 0,10 10 0)),((13.3333333333333 ←
 13.3333333333333 10,12 12 10,10 10 0,13.3333333333333 13.3333333333333 10)),
((12 28 10,12 12 10,13.3333333333333 13.3333333333333 10,12 28 10)),
((16.6666666666667 23.3333333333333 10,9.99999999999995 30 0,12 28 ←
 10,16.6666666666667 23.3333333333333 10)),
((10 10 0,0 20 0,9.99999999999995 30 0,10 10 0)),
((12 12 10,11 11 10,10 10 0,12 12 10)),((12 28 10,11 11 10,12 12 10,12 28 10)),
((9.99999999999995 30 0,11 29 10,12 28 10,9.99999999999995 30 0)),((0 20 0,2 20 ←
 10,9.99999999999995 30 0,0 20 0)),
((10 10 0,2 20 10,0 20 0,10 10 0)),((11 11 10,2 20 10,10 10 0,11 11 10)),((12 28 ←
 10,11 29 10,11 11 10,12 28 10)),
((9.99999999999995 30 0,2 20 10,11 29 10,9.99999999999995 30 0)),((11 11 10,11 29 ←
 10,2 20 10,11 11 10)))
```

### 8.3.13 CG\_Union

CG\_Union — Computes the union of two geometries

#### Synopsis

geometry **CG\_Union**( geometry geomA , geometry geomB );

#### Beschreibung

Computes the union of two geometries.

Performed by the SFCGAL module



#### Note

NOTE: this function returns a geometry representing the union.

Verfügbarkeit: 3.5.0



Diese Methode benötigt ein SFCGAL-Backend.

#### Beispiele mit dem geometrischen Datentyp

```
SELECT CG_Union('POINT(.5 0)', 'LINESTRING(-1 0,1 0)');
      cg_union
-----
LINESTRING(-1 0,0.5 0,1 0)
(1 row)
```

#### Siehe auch

[ST\\_3DUnion](#), [ST\\_AsBinary](#)



### 8.3.14 ST\_3DUnion

ST\_3DUnion — 3D-Vereinigung durchführen.

#### Synopsis

```
geometry ST_3DUnion(geometry geom1, geometry geom2);  
geometry ST_3DUnion(geometry set g1field);
```

#### Beschreibung



#### Warning

ST\_3DUnion is deprecated as of 3.5.0. Use [CG\\_3DUnion](#) instead.

Verfügbarkeit: 2.2.0

Verfügbarkeit: 3.3.0 Aggregatvariante wurde hinzugefügt



Diese Methode benötigt ein SFCGAL-Backend.



Diese Methode setzt die SQL/MM-Spezifikation um. SQL-MM IEC 13249-3: 5.1



Diese Funktion unterstützt 3d und lässt den Z-Index nicht fallen.



Diese Funktion unterstützt polyedrische Flächen.



Diese Funktion unterstützt Dreiecke und dreieckige unregelmäßige Netzoberflächen (TIN).

**Aggregat-Variante:** gibt eine Geometrie zurück, die die 3D-Vereinigung eines Rowsets von Geometrien ist. Die Funktion ST\_3DUnion() ist eine "Aggregat"-Funktion in der Terminologie von PostgreSQL. Das bedeutet, dass sie mit Datenzeilen arbeitet, so wie es auch die Funktionen SUM() und AVG() tun, und wie die meisten Aggregate ignoriert sie auch NULL-Geometrien.

### 8.3.15 CG\_3DUnion

CG\_3DUnion — Perform 3D union using postgis\_sfcgal.

#### Synopsis

```
geometry CG_3DUnion(geometry geom1, geometry geom2);  
geometry CG_3DUnion(geometry set g1field);
```

#### Beschreibung

Verfügbarkeit: 3.5.0



Diese Methode benötigt ein SFCGAL-Backend.



Diese Methode setzt die SQL/MM-Spezifikation um. SQL-MM IEC 13249-3: 5.1



Diese Funktion unterstützt 3d und lässt den Z-Index nicht fallen.

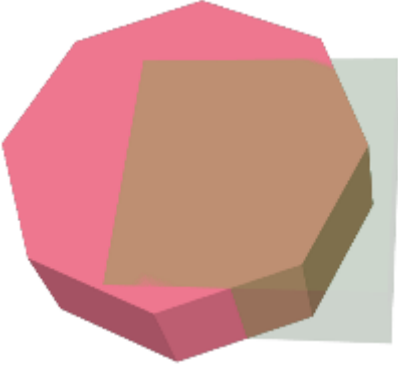
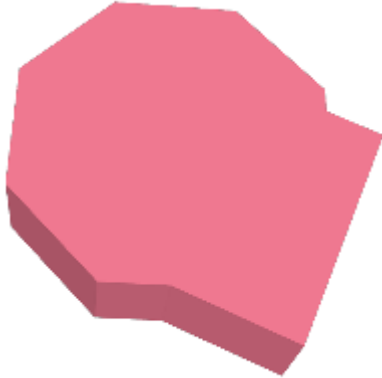
✔ Diese Funktion unterstützt polyedrische Flächen.

✔ Diese Funktion unterstützt Dreiecke und dreieckige unregelmäßige Netzoberflächen (TIN).

**Aggregate variant:** returns a geometry that is the 3D union of a rowset of geometries. The `CG_3DUnion()` function is an "aggregate" function in the terminology of PostgreSQL. That means that it operates on rows of data, in the same way the `SUM()` and `AVG()` functions do and like most aggregates, it also ignores NULL geometries.

## Beispiele

Die 3D-Bilder wurden mit PostGIS `ST_AsX3D` erzeugt und in HTML mit `X3Dom HTML Javascript Rendering Library` gerendert.

<pre>SELECT CG_Extrude(ST_Buffer(↵     ST_GeomFromText('POINT(100 90)'),     50, '↵ quad_segs=2'),0,0,30) AS geom1, ↵ CG_Extrude(ST_Buffer(ST_GeomFromText('POINT(80 80)'),     50, '↵ quad_segs=1'),0,0,30) AS geom2;</pre>  <p><i>Original 3D-Geometrien überlagert. geom2 ist diejenige mit Transparenz.</i></p>	<pre>SELECT CG_3DUnion(geom1,geom2) FROM (↵ SELECT CG_Extrude(ST_Buffer(ST_G     50, '↵ quad_segs=2'),0,0,30) AS geom1, ↵ CG_Extrude(ST_Buffer(ST_GeomFrom     50, '↵ quad_segs=1'),0,0,30) AS geom2 )</pre>  <p><i>Vereinigung von geom1 und geom2</i></p>
---	---

## Siehe auch

[CG\\_Extrude](#), [ST\\_AsX3D](#), [CG\\_3DIntersection](#) [CG\\_3DDifference](#)

### 8.3.16 ST\_AlphaShape

`ST_AlphaShape` — Berechnet eine Alpha-Form, die eine Geometrie umschließt

#### Synopsis

geometry **ST\_AlphaShape**(geometry geom, float alpha, boolean allow\_holes = false);

## Beschreibung

---



### Warning

`ST_AlphaShape` is deprecated as of 3.5.0. Use `CG_AlphaShape` instead.

---

Berechnet den **Alpha-Shape** der Punkte in einer Geometrie. Eine Alpha-Form ist eine (normalerweise) konkave polygonale Geometrie, die alle Scheitelpunkte der Eingabe enthält und deren Scheitelpunkte eine Teilmenge der Eingabe-Scheitelpunkte sind. Eine Alpha-Form passt sich besser an die Form der Eingabe an als die Form, die durch die **konvexe Hülle** erzeugt wird.

### 8.3.17 CG\_AlphaShape

`CG_AlphaShape` — Berechnet eine Alpha-Form, die eine Geometrie umschließt

#### Synopsis

```
geometry CG_AlphaShape(geometry geom, float alpha, boolean allow_holes = false);
```

#### Beschreibung

Berechnet den **Alpha-Shape** der Punkte in einer Geometrie. Eine Alpha-Form ist eine (normalerweise) konkave polygonale Geometrie, die alle Scheitelpunkte der Eingabe enthält und deren Scheitelpunkte eine Teilmenge der Eingabe-Scheitelpunkte sind. Eine Alpha-Form passt sich besser an die Form der Eingabe an als die Form, die durch die **konvexe Hülle** erzeugt wird.

Die "Passgenauigkeit" wird durch den Parameter `alpha` gesteuert, der Werte zwischen 0 und unendlich annehmen kann. Kleinere Alpha-Werte erzeugen konkavere Ergebnisse. Alpha-Werte, die größer als ein datenabhängiger Wert sind, erzeugen die konvexe Hülle der Eingabe.



#### Note

Nach der CGAL-Implementierung ist der Alpha-Wert das *Quadrat* des Radius der Scheibe, die im Alpha-Shape-Algorithmus verwendet wird, um die Delaunay-Triangulation der Eingabepunkte zu "erodieren". Siehe **CGAL Alpha-Shapes** für weitere Informationen. Dies unterscheidet sich von der ursprünglichen Definition von Alpha-Formen, die Alpha als den Radius der erodierenden Scheibe definiert.

---

Die berechnete Form enthält keine Löcher, es sei denn, das optionale Argument `allow_holes` wird als `true` angegeben.

Diese Funktion berechnet eine konkave Hülle einer Geometrie auf ähnliche Weise wie `ST_ConcaveHull`, verwendet aber CGAL und einen anderen Algorithmus.

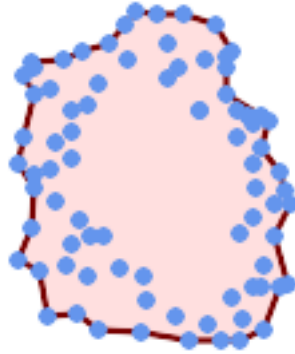
Availability: 3.5.0 - requires SFCGAL >= 1.4.1.



Diese Methode benötigt ein SFCGAL-Backend.

---

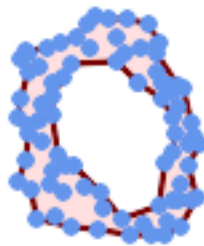
## Beispiele



Alpha-shape of a MultiPoint (same example As *CG\_OptimalAlphaShape*)

```
SELECT ST_AsText(CG_AlphaShape('MULTIPOINT((63 84),(76 88),(68 73),(53 18),(91 50),(81 70),
(88 29),(24 82),(32 51),(37 23),(27 54),(84 19),(75 87),(44 42),(77 67),(90 30) ←
,(36 61),(32 65),
(81 47),(88 58),(68 73),(49 95),(81 60),(87 50),
(78 16),(79 21),(30 22),(78 43),(26 85),(48 34),(35 35),(36 40),(31 79),(83 29) ←
,(27 84),(52 98),(72 95),(85 71),
(75 84),(75 77),(81 29),(77 73),(41 42),(83 72),(23 36),(89 53),(27 57),(57 97) ←
,(27 77),(39 88),(60 81),
(80 72),(54 32),(55 26),(62 22),(70 20),(76 27),(84 35),(87 42),(82 54),(83 64) ←
,(69 86),(60 90),(50 86),(43 80),(36 73),
(36 68),(40 75),(24 67),(23 60),(26 44),(28 33),(40 32),(43 19),(65 16),(73 16) ←
,(38 46),(31 59),(34 86),(45 90),(64 97)')::geometry,80.2));
```

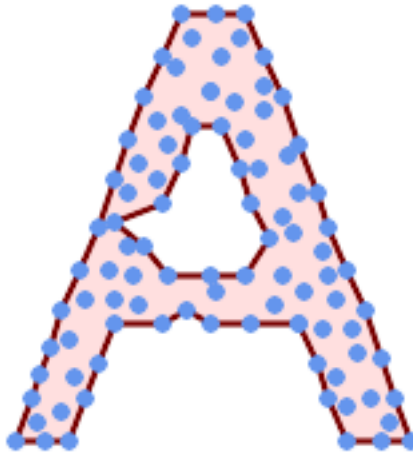
```
POLYGON((89 53,91 50,87 42,90 30,88 29,84 19,78 16,73 16,65 16,53 18,43 19,
37 23,30 22,28 33,23 36,26 44,27 54,23 60,24 67,27 77,
24 82,26 85,34 86,39 88,45 90,49 95,52 98,57 97,
64 97,72 95,76 88,75 84,83 72,85 71,88 58,89 53))
```



Alpha-shape of a MultiPoint, allowing holes (same example as *CG\_OptimalAlphaShape*)

```
SELECT ST_AsText(CG_AlphaShape('MULTIPOINT((63 84),(76 88),(68 73),(53 18),(91 50),(81 70) ←
, (88 29),(24 82),(32 51),(37 23),(27 54),(84 19),(75 87),(44 42),(77 67),(90 30),(36 61) ←
, (32 65),(81 47),(88 58),(68 73),(49 95),(81 60),(87 50),
(78 16),(79 21),(30 22),(78 43),(26 85),(48 34),(35 35),(36 40),(31 79),(83 29),(27 84) ←
, (52 98),(72 95),(85 71),
(75 84),(75 77),(81 29),(77 73),(41 42),(83 72),(23 36),(89 53),(27 57),(57 97),(27 77) ←
, (39 88),(60 81),
(80 72),(54 32),(55 26),(62 22),(70 20),(76 27),(84 35),(87 42),(82 54),(83 64),(69 86) ←
, (60 90),(50 86),(43 80),(36 73),
(36 68),(40 75),(24 67),(23 60),(26 44),(28 33),(40 32),(43 19),(65 16),(73 16),(38 46) ←
, (31 59),(34 86),(45 90),(64 97))'::geometry, 100.1,true))
```

```
POLYGON((89 53,91 50,87 42,90 30,84 19,78 16,73 16,65 16,53 18,43 19,30 22,28 33,23 36,
26 44,27 54,23 60,24 67,27 77,24 82,26 85,34 86,39 88,45 90,49 95,52 98,57 97,64 97,72 95,
76 88,75 84,83 72,85 71,88 58,89 53),(36 61,36 68,40 75,43 80,60 81,68 73,77 67,
81 60,82 54,81 47,78 43,76 27,62 22,54 32,44 42,38 46,36 61))
```



*Alpha-Form eines MultiPoint, die Löcher zulässt (gleiches Beispiel wie [ST\\_OptimalAlphaShape](#))*

```
SELECT ST_AsText(CG_AlphaShape(
'MULTIPOINT ((132 64), (114 64), (99 64), (81 64), (63 64), (57 49), (52 ←
36), (46 20), (37 20), (26 20), (32 36), (39 55), (43 69), (50 84), (57 ←
100), (63 118), (68 133), (74 149), (81 164), (88 180), (101 180), (112 ←
180), (119 164), (126 149), (132 131), (139 113), (143 100), (150 84), ←
(157 69), (163 51), (168 36), (174 20), (163 20), (150 20), (143 36), ←
(139 49), (132 64), (99 151), (92 138), (88 124), (81 109), (74 93), (70 ←
82), (83 82), (99 82), (112 82), (126 82), (121 96), (114 109), (110 ←
122), (103 138), (99 151), (34 27), (43 31), (48 44), (46 58), (52 73), ←
(63 73), (61 84), (72 71), (90 69), (101 76), (123 71), (141 62), (166 ←
27), (150 33), (159 36), (146 44), (154 53), (152 62), (146 73), (134 ←
76), (143 82), (141 91), (130 98), (126 104), (132 113), (128 127), (117 ←
122), (112 133), (119 144), (108 147), (119 153), (110 171), (103 164), ←
(92 171), (86 160), (88 142), (79 140), (72 124), (83 131), (79 118), ←
(68 113), (63 102), (68 93), (35 45))'::geometry,102.2, true));
```

```
POLYGON((26 20,32 36,35 45,39 55,43 69,50 84,57 100,63 118,68 133,74 149,81 164,88 180,
101 180,112 180,119 164,126 149,132 131,139 113,143 100,150 84,157 69,163 ←
51,168 36,
174 20,163 20,150 20,143 36,139 49,132 64,114 64,99 64,90 69,81 64,63 64,57 ←
49,52 36,46 20,37 20,26 20),
```

```
(74 93,81 109,88 124,92 138,103 138,110 122,114 109,121 96,112 82,99 82,83 82,74 93) ←
```

## Siehe auch

[ST\\_ConcaveHull](#), [CG\\_OptimalAlphaShape](#)

### 8.3.18 CG\_ApproxConvexPartition

CG\_ApproxConvexPartition — Berechnet die approximale konvexe Partition der Polygeometrie

#### Synopsis

```
geometry CG_ApproxConvexPartition(geometry geom);
```

#### Beschreibung

Berechnet die approximale konvexe Partition der Polygeometrie (unter Verwendung einer Triangulation).

---

#### Note

Eine Partition eines Polygons P ist eine Menge von Polygonen, bei der sich die Innenräume der Polygone nicht schneiden und die Vereinigung der Polygone gleich dem Innenraum des ursprünglichen Polygons P ist. Die Funktionen CG\_ApproxConvexPartition und CG\_GreeneApproxConvexPartition erzeugen annähernd optimale konvexe Partitionen. Beide Funktionen erzeugen konvexe Zerlegungen, indem sie das Polygon zunächst in einfachere Polygone zerlegen; CG\_ApproxConvexPartition verwendet eine Triangulation und CG\_GreeneApproxConvexPartition eine monotone Partition. Diese beiden Funktionen garantieren, dass sie nicht mehr als das Vierfache der optimalen Anzahl konvexer Teile erzeugen, unterscheiden sich aber in ihrer Laufzeitkomplexität. Obwohl der auf Triangulation basierende Approximationsalgorithmus oft zu weniger konvexen Teilen führt, ist dies nicht immer der Fall.

---

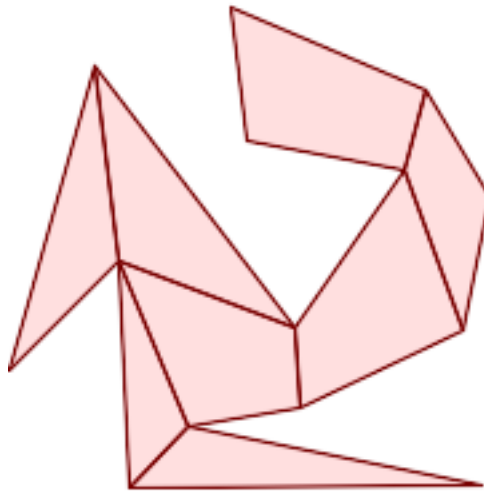
Verfügbarkeit: 3.5.0 - erfordert SFCGAL >= 1.5.0.

Erfordert SFCGAL >= 1.5.0



Diese Methode benötigt ein SFCGAL-Backend.

## Beispiele



*Approximale konvexe Partition (gleiches Beispiel wie [CG\\_YMonotonePartition](#), [CG\\_GreeneApproxConvexPartition](#) und [CG\\_OptimalConvexPartition](#))*

```
SELECT ST_AsText(CG_ApproxConvexPartition('POLYGON((156 150,83 181,89 131,148 120,107 61,32 ←
159,0 45,41 86,45 1,177 2,67 24,109 31,170 60,180 110,156 150))'::geometry));
```

```
GEOMETRYCOLLECTION(POLYGON((156 150,83 181,89 131,148 120,156 150)),POLYGON((32 159,0 45,41 ←
86,32 159)),POLYGON((107 61,32 159,41 86,107 61)),POLYGON((45 1,177 2,67 24,45 1)), ←
POLYGON((41 86,45 1,67 24,41 86)),POLYGON((107 61,41 86,67 24,109 31,107 61)),POLYGON ←
((148 120,107 61,109 31,170 60,148 120)),POLYGON((156 150,148 120,170 60,180 110,156 ←
150)))
```

## Siehe auch

[CG\\_YMonotonePartition](#), [CG\\_GreeneApproxConvexPartition](#), [CG\\_OptimalConvexPartition](#)

### 8.3.19 ST\_ApproximateMedialAxis

`ST_ApproximateMedialAxis` — Berechnet die konvexe Hülle einer Geometrie.

#### Synopsis

```
geometry ST_ApproximateMedialAxis(geometry geom);
```

#### Beschreibung



#### Warning

`ST_ApproximateMedialAxis` is deprecated as of 3.5.0. Use [CG\\_ApproximateMedialAxis](#) instead.

Return an approximate medial axis for the areal input based on its straight skeleton. Uses an SFCGAL specific API when built against a capable version (1.2.0+). Otherwise the function is just a wrapper around `CG_StraightSkeleton` (slower case).

Verfügbarkeit: 2.2.0



Diese Methode benötigt ein SFCGAL-Backend.



Diese Funktion unterstützt 3d und lässt den Z-Index nicht fallen.



Diese Funktion unterstützt polyedrische Flächen.



Diese Funktion unterstützt Dreiecke und dreieckige unregelmäßige Netzoberflächen (TIN).

### 8.3.20 `CG_ApproximateMedialAxis`

`CG_ApproximateMedialAxis` — Berechnet die konvexe Hülle einer Geometrie.

#### Synopsis

```
geometry CG_ApproximateMedialAxis(geometry geom);
```

#### Beschreibung

Return an approximate medial axis for the areal input based on its straight skeleton. Uses an SFCGAL specific API when built against a capable version (1.2.0+). Otherwise the function is just a wrapper around `CG_StraightSkeleton` (slower case).

Verfügbarkeit: 3.5.0



Diese Methode benötigt ein SFCGAL-Backend.



Diese Funktion unterstützt 3d und lässt den Z-Index nicht fallen.



Diese Funktion unterstützt polyedrische Flächen.

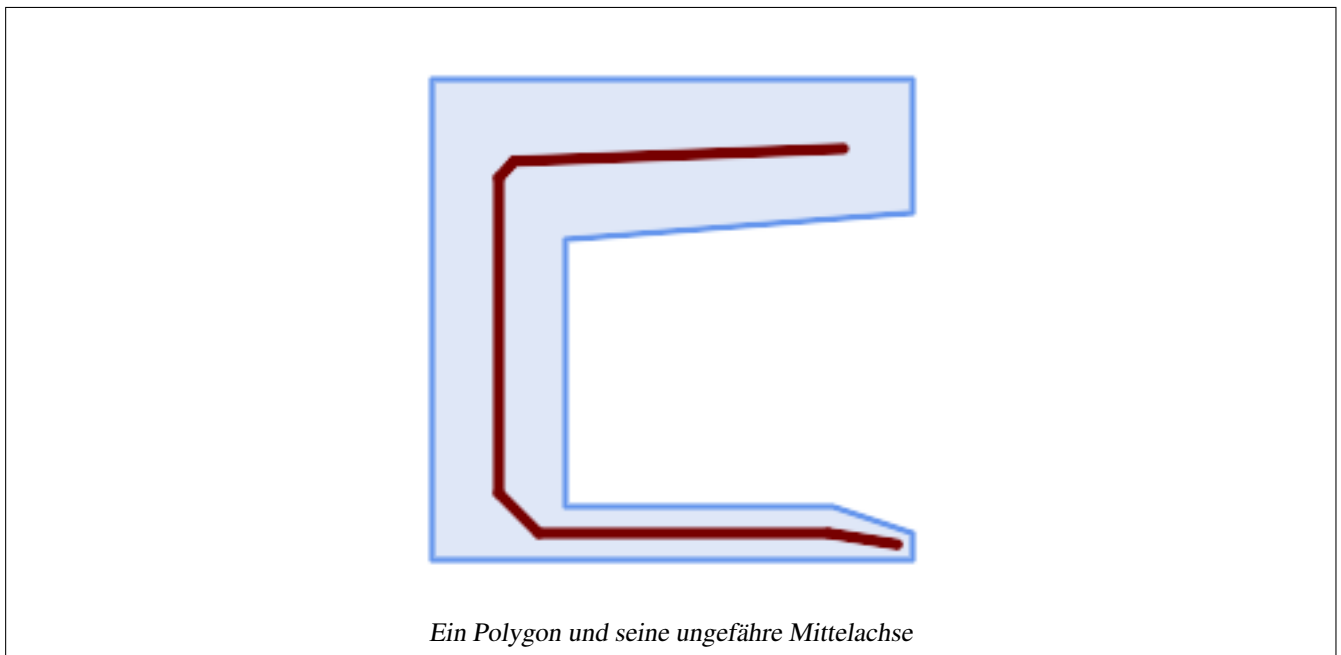


Diese Funktion unterstützt Dreiecke und dreieckige unregelmäßige Netzoberflächen (TIN).

#### Beispiele

```
SELECT CG_ApproximateMedialAxis(ST_GeomFromText('POLYGON (( 190 190, 10 190, 10 10, 190 10, ←  
190 20, 160 30, 60 30, 60 130, 190 140, 190 190 ))'));
```





Siehe auch

[CG\\_StraightSkeleton](#)

### 8.3.21 ST\_ConstrainedDelaunayTriangles

`ST_ConstrainedDelaunayTriangles` — Gibt eine eingeschränkte Delaunay-Triangulation um die angegebene Eingabegeometrie zurück.

Synopsis

```
geometry ST_ConstrainedDelaunayTriangles(geometry g1);
```

Beschreibung



**Warning**

`ST_ConstrainedDelaunayTriangles` is deprecated as of 3.5.0. Use [CG\\_ConstrainedDelaunayTriangles](#) instead.

Gibt eine **Constrained Delaunay Triangulation** um die Eckpunkte der Eingabegeometrie zurück. Die Ausgabe ist ein TIN.



Diese Methode benötigt ein SFCGAL-Backend.

Verfügbarkeit: 2.1.0



Diese Funktion unterstützt 3d und lässt den Z-Index nicht fallen.

### 8.3.22 CG\_ConstrainedDelaunayTriangles

`CG_ConstrainedDelaunayTriangles` — Gibt eine eingeschränkte Delaunay-Triangulation um die angegebene Eingabegeometrie zurück.

## Synopsis

```
geometry CG_ConstrainedDelaunayTriangles(geometry g1);
```

## Beschreibung

Gibt eine **Constrained Delaunay Triangulation** um die Eckpunkte der Eingabegeometrie zurück. Die Ausgabe ist ein TIN.



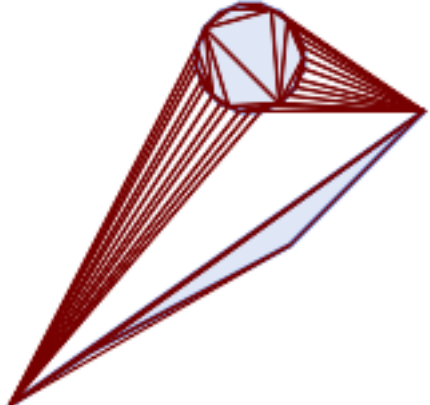
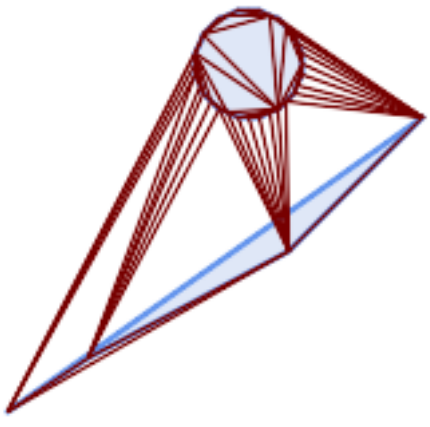
Diese Methode benötigt ein SFCGAL-Backend.

Verfügbarkeit: 2.1.0



Diese Funktion unterstützt 3d und lässt den Z-Index nicht fallen.

## Beispiele

 <p><i>CG_ConstrainedDelaunayTriangles of 2 polygons</i></p> <pre>select CG_ConstrainedDelaunayTriangles(   ST_Union(     POLYGON((175 150, 20 40, 50 60, 125 100, 175 150)),     ST_Buffer('POINT(110 170)::geometry', 20)   ) );</pre>	 <p><i>ST_DelaunayTriangles von 2 Polygonen. Die Kanten des Dreiecks schneiden die Polygongrenzen.</i></p> <pre>select ST_DelaunayTriangles(   ST_Union(     POLYGON((175 150, 20 40, 50 60, 125 100, 175 150)),     ST_Buffer('POINT(110 170)::geometry', 20)   ) );</pre>
--	--

## Siehe auch

[ST\\_DelaunayTriangles](#), [ST\\_TriangulatePolygon](#), [CG\\_Tessellate](#), [ST\\_ConcaveHull](#), [ST\\_Dump](#)

### 8.3.23 ST\_Extrude

ST\_Extrude — Extrudieren einer Oberfläche in ein zugehöriges Volumen

#### Synopsis

geometry **ST\_Extrude**(geometry geom, float x, float y, float z);

#### Beschreibung



#### Warning

**ST\_Extrude** is deprecated as of 3.5.0. Use **CG\_Extrude** instead.

---

Verfügbarkeit: 2.1.0



Diese Methode benötigt ein SFCGAL-Backend.



Diese Funktion unterstützt 3d und lässt den Z-Index nicht fallen.



Diese Funktion unterstützt polyedrische Flächen.



Diese Funktion unterstützt Dreiecke und dreieckige unregelmäßige Netzoberflächen (TIN).

### 8.3.24 CG\_Extrude

CG\_Extrude — Extrudieren einer Oberfläche in ein zugehöriges Volumen

#### Synopsis

geometry **CG\_Extrude**(geometry geom, float x, float y, float z);

#### Beschreibung

Verfügbarkeit: 3.5.0



Diese Methode benötigt ein SFCGAL-Backend.



Diese Funktion unterstützt 3d und lässt den Z-Index nicht fallen.



Diese Funktion unterstützt polyedrische Flächen.




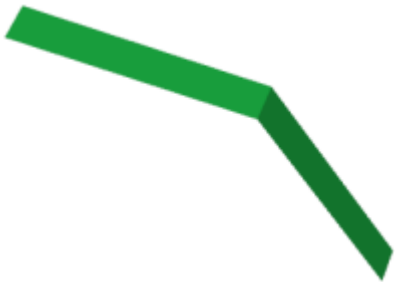


Diese Funktion unterstützt Dreiecke und dreieckige unregelmäßige Netzoberflächen (TIN).

#### Beispiele

Die 3D-Bilder wurden mit PostGIS **ST\_AsX3D** erzeugt und in HTML mit **X3Dom HTML Javascript Rendering Library** gerendert.

---

<pre>SELECT ST_Buffer(ST_GeomFromText('POINT (100 90)'),                     50, 'quad_segs=2'),0,0,30);</pre>  <p><i>Ursprüngliches Achteck aus Pufferpunkt gebildet</i></p>	<pre>CG_Extrude(ST_Buffer(ST_GeomFromText('POINT (100 90)'),                     50, 'quad_segs=2'),0,0,30);</pre>  <p><i>Ein Sechseck, das um 30 Einheiten entlang Z extrudiert wird, ergibt eine PolyhedralSurfaceZ</i></p>
<pre>SELECT ST_GeomFromText('LINESTRING(50 50, 100 90, 95 150)')</pre>  <p><i>Ursprüngliche Polygone</i></p>	<pre>SELECT CG_Extrude(ST_GeomFromText('LINESTRING(50 50, 100 90, 95 150)'))</pre>  <p><i>LineString Extrudiert entlang Z erzeugt eine PolyhedralSurfaceZ</i></p>

**Siehe auch**

[ST\\_AsX3D](#), [CG\\_ExtrudeStraightSkeleton](#)

### 8.3.25 CG\_ExtrudeStraightSkeleton

CG\_ExtrudeStraightSkeleton — Gerade Skelett-Extrusion

#### Synopsis

```
geometry CG_ExtrudeStraightSkeleton(geometry geom, float roof_height, float body_height = 0);
```

#### Beschreibung

Berechnet eine Extrusion mit einer maximalen Höhe der Polygoneometrie.

#### Note



Perhaps the first (historically) use-case of straight skeletons: given a polygonal roof, the straight skeleton directly gives the layout of each tent. If each skeleton edge is lifted from the plane a height equal to its offset distance, the resulting roof is "correct" in that water will always fall down to the contour edges (the roof's border), regardless of where it falls on the roof. The function computes this extrusion aka "roof" on a polygon. If the argument `body_height > 0`, so the polygon is extruded like with `CG_Extrude(polygon, 0, 0, body_height)`. The result is an union of these polyhedralsurfaces.

Verfügbarkeit: 3.5.0 - erfordert SFCGAL >= 1.5.0.

Erfordert SFCGAL >= 1.5.0



Diese Methode benötigt ein SFCGAL-Backend.

#### Beispiele

```
SELECT ST_AsText(CG_ExtrudeStraightSkeleton('POLYGON (( 0 0, 5 0, 5 5, 4 5, 4 4, 0 4, 0 0 ) ←
, (1 1, 1 2,2 2, 2 1, 1 1))', 3.0, 2.0));
```

```
POLYHEDRALSURFACE Z (((0 0 0,0 4 0,4 4 0,4 5 0,5 5 0,5 0 0,0 0 0), (1 1 0,2 1 0,2 2 0,1 2 ←
0,1 1 0)), ((0 0 0,0 0 2,0 4 2,0 4 0,0 0 0), ((0 4 0,0 4 2,4 4 2,4 4 0,0 4 0)), ((4 4 0,4 ←
4 2,4 5 2,4 5 0,4 4 0)), ((4 5 0,4 5 2,5 5 2,5 5 0,4 5 0)), ((5 5 0,5 5 2,5 0 2,5 0 0,5 5 ←
0)), ((5 0 0,5 0 2,0 0 2,0 0 0,5 0 0)), ((1 1 0,1 1 2,2 1 2,2 1 0,1 1 0)), ((2 1 0,2 1 2,2 ←
2 2,2 2 0,2 1 0)), ((2 2 0,2 2 2,1 2 2,1 2 0,2 2 0)), ((1 2 0,1 2 2,1 1 2,1 1 0,1 2 0)) ←
, ((4 5 2,5 5 2,4 4 2,4 5 2)), ((2 1 2,5 0 2,0 0 2,2 1 2)), ((5 5 2,5 0 2,4 4 2,5 5 2)), ((2 ←
1 2,0 0 2,1 1 2,2 1 2)), ((1 2 2,1 1 2,0 0 2,1 2 2)), ((0 4 2,2 2 2,1 2 2,0 4 2)), ((0 4 ←
2,1 2 2,0 0 2,0 4 2)), ((4 4 2,5 0 2,2 2 4 4 2)), ((4 4 2,2 2 2,0 4 2,4 4 2)), ((2 2 2,5 ←
0 2,2 1 2,2 2 2)), ((0.5 2.5 2.5,0 0 2,0.5 0.5 2.5,0.5 2.5 2.5)), ((1 3 3,0 4 2,0.5 2.5 ←
2.5,1 3 3)), ((0.5 2.5 2.5,0 4 2,0 0 2,0.5 2.5 2.5)), ((2.5 0.5 2.5,5 0 2,3.5 1.5 3.5,2.5 ←
0.5 2.5)), ((0 0 2,5 0 2,2.5 0.5 2.5,0 0 2)), ((0.5 0.5 2.5,0 0 2,2.5 0.5 2.5,0.5 0.5 2.5) ←
), ((4.5 3.5 2.5,5 5 2,4.5 4.5 2.5,4.5 3.5 2.5)), ((3.5 2.5 3.5,3.5 1.5 3.5,4.5 3.5 ←
2.5,3.5 2.5 3.5)), ((4.5 3.5 2.5,5 0 2,5 5 2,4.5 3.5 2.5)), ((3.5 1.5 3.5,5 0 2,4.5 3.5 ←
2.5,3.5 1.5 3.5)), ((5 5 2,4 5 2,4.5 4.5 2.5,5 5 2)), ((4.5 4.5 2.5,4 4 2,4.5 3.5 2.5,4.5 ←
4.5 2.5)), ((4.5 4.5 2.5,4 5 2,4 4 2,4.5 4.5 2.5)), ((3 3 3,0 4 2,1 3 3,3 3 3)), ((3.5 2.5 ←
3.5,4.5 3.5 2.5,3 3 3,3.5 2.5 3.5)), ((3 3 3,4 4 2,0 4 2,3 3 3)), ((4.5 3.5 2.5,4 4 2,3 3 ←
3,4.5 3.5 2.5)), ((2 1 2,1 1 2,0.5 0.5 2.5,2 1 2)), ((2.5 0.5 2.5,2 1 2,0.5 0.5 2.5,2.5 ←
0.5 2.5)), ((1 1 2,1 2 2,0.5 2.5 2.5,1 1 2)), ((0.5 0.5 2.5,1 1 2,0.5 2.5 2.5,0.5 0.5 2.5) ←
), ((1 3 3,2 2 2,3 3 3,1 3 3)), ((0.5 2.5 2.5,1 2 2,1 3 3,0.5 2.5 2.5)), ((1 3 3,1 2 2,2 2 ←
2,1 3 3)), ((2 2 2,2 1 2,2.5 0.5 2.5,2 2 2)), ((3.5 2.5 3.5,3 3 3,3.5 1.5 3.5,3.5 2.5 3.5) ←
), ((3.5 1.5 3.5,2 2 2,2.5 0.5 2.5,3.5 1.5 3.5)), ((3 3 3,2 2 2,3.5 1.5 3.5,3 3 3)))
```

#### Siehe auch

[ST\\_Extrude](#), [CG\\_StraightSkeleton](#)

### 8.3.26 CG\_GreeneApproxConvexPartition

CG\_GreeneApproxConvexPartition — Berechnet die approximale konvexe Partition der Polygeometrie

#### Synopsis

```
geometry CG_GreeneApproxConvexPartition(geometry geom);
```

#### Beschreibung

Berechnet approximale monotone konvexe Partition der Polygeometrie.

#### Note

Eine Partition eines Polygons P ist eine Menge von Polygonen, bei der sich die Innenräume der Polygone nicht schneiden und die Vereinigung der Polygone gleich dem Innenraum des ursprünglichen Polygons P ist. Die Funktionen CG\_ApproxConvexPartition und CG\_GreeneApproxConvexPartition erzeugen annähernd optimale konvexe Partitionen. Beide Funktionen erzeugen konvexe Zerlegungen, indem sie das Polygon zunächst in einfachere Polygone zerlegen; CG\_ApproxConvexPartition verwendet eine Triangulation und CG\_GreeneApproxConvexPartition eine monotone Partition. Diese beiden Funktionen garantieren, dass sie nicht mehr als das Vierfache der optimalen Anzahl konvexer Teile erzeugen, unterscheiden sich aber in ihrer Laufzeitkomplexität. Obwohl der auf Triangulation basierende Approximationsalgorithmus oft zu weniger konvexen Teilen führt, ist dies nicht immer der Fall.

Note!

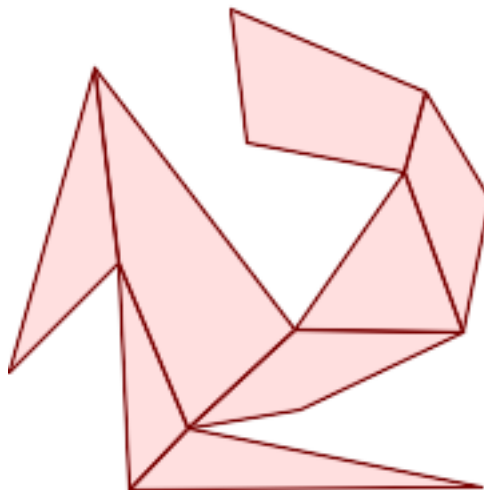
Verfügbarkeit: 3.5.0 - erfordert SFCGAL >= 1.5.0.

Erfordert SFCGAL >= 1.5.0



Diese Methode benötigt ein SFCGAL-Backend.

#### Beispiele



*Greene Approximal Convex Partition (gleiches Beispiel wie [CG\\_YMonotonePartition](#), [CG\\_ApproxConvexPartition](#) und [CG\\_OptimalConvexPartition](#))*

```
SELECT ST_AsText(CG_GreeneApproxConvexPartition('POLYGON((156 150,83 181,89 131,148 120,107 ←
61,32 159,0 45,41 86,45 1,177 2,67 24,109 31,170 60,180 110,156 150))'::geometry));
```

```
GEOMETRYCOLLECTION(POLYGON((32 159,0 45,41 86,32 159)),POLYGON((45 1,177 2,67 24,45 1)), ←  
POLYGON((67 24,109 31,170 60,107 61,67 24)),POLYGON((41 86,45 1,67 24,41 86)),POLYGON ←  
((107 61,32 159,41 86,67 24,107 61)),POLYGON((148 120,107 61,170 60,148 120)),POLYGON ←  
((148 120,170 60,180 110,156 150,148 120)),POLYGON((156 150,83 181,89 131,148 120,156 ←  
150)))
```

### Siehe auch

[CG\\_YMonotonePartition](#), [CG\\_ApproxConvexPartition](#), [CG\\_OptimalConvexPartition](#)

## 8.3.27 ST\_MinkowskiSum

ST\_MinkowskiSum — Führt die Minkowski-Summe aus

### Synopsis

```
geometry ST_MinkowskiSum(geometry geom1, geometry geom2);
```

### Beschreibung



#### Warning

ST\_MinkowskiSum is deprecated as of 3.5.0. Use [CG\\_MinkowskiSum](#) instead.

Diese Funktion führt eine 2D-Minkowski-Summe eines Punktes, einer Linie oder eines Polygons mit einem Polygon durch.

Eine Minkowski-Summe zweier Geometrien A und B ist die Menge aller Punkte, die die Summe aller Punkte in A und B sind. Minkowski-Summen werden häufig in der Bewegungsplanung und im computergestützten Design verwendet. Mehr Details auf [Wikipedia Minkowski Addition](#).

Der erste Parameter kann eine beliebige 2D-Geometrie sein (Punkt, Linienstring, Polygon). Wird eine 3D-Geometrie übergeben, so wird sie in 2D konvertiert, indem Z auf 0 gesetzt wird, was zu möglichen Ungültigkeitsfällen führen kann. Der zweite Parameter muss ein 2D-Polygon sein.

Die Implementierung verwendet [CGAL 2D Minkowskisum](#).

Verfügbarkeit: 2.1.0



Diese Methode benötigt ein SFCGAL-Backend.

## 8.3.28 CG\_MinkowskiSum

CG\_MinkowskiSum — Führt die Minkowski-Summe aus

### Synopsis

```
geometry CG_MinkowskiSum(geometry geom1, geometry geom2);
```

## Beschreibung

Diese Funktion führt eine 2D-Minkowski-Summe eines Punktes, einer Linie oder eines Polygons mit einem Polygon durch.

Eine Minkowski-Summe zweier Geometrien A und B ist die Menge aller Punkte, die die Summe aller Punkte in A und B sind. Minkowski-Summen werden häufig in der Bewegungsplanung und im computergestützten Design verwendet. Mehr Details auf [Wikipedia Minkowski Addition](#).

Der erste Parameter kann eine beliebige 2D-Geometrie sein (Punkt, Linienstring, Polygon). Wird eine 3D-Geometrie übergeben, so wird sie in 2D konvertiert, indem Z auf 0 gesetzt wird, was zu möglichen Ungültigkeitsfällen führen kann. Der zweite Parameter muss ein 2D-Polygon sein.

Die Implementierung verwendet [CGAL 2D Minkowskisum](#).

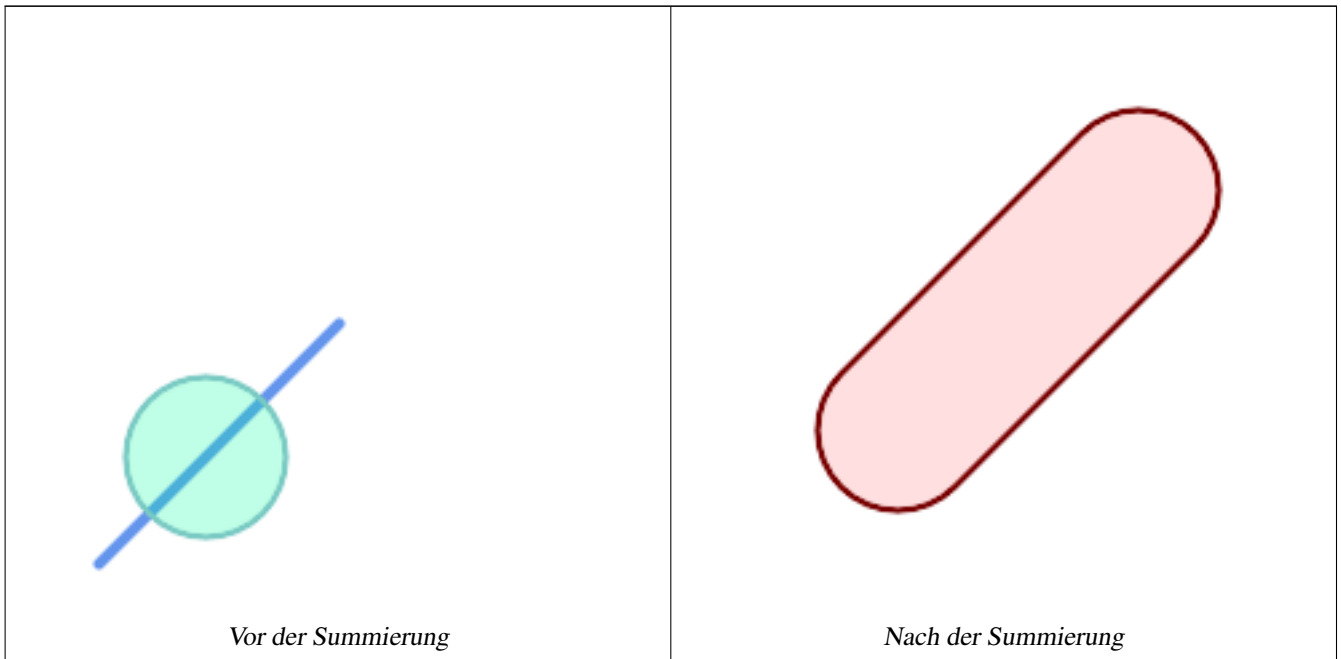
Verfügbarkeit: 3.5.0



Diese Methode benötigt ein SFCGAL-Backend.

## Beispiele

Minkowski-Summe aus Linienzug und Kreispolygon, wobei der Linienzug den Kreis durchschneidet



```
SELECT CG_MinkowskiSum(line, circle)
FROM (SELECT
  ST_MakeLine(ST_Point(10, 10),ST_Point(100, 100)) As line,
  ST_Buffer(ST_GeomFromText('POINT(50 50)'), 30) As circle) As foo;

-- wkt --
MULTIPOLYGON(((30 59.9999999999999,30.5764415879031 ↵
  54.1472903395161,32.2836140246614 48.5194970290472,35.0559116309237 ↵
  43.3328930094119,38.7867965644036 38.7867965644035,43.332893009412 ↵
  35.0559116309236,48.5194970290474 32.2836140246614,54.1472903395162 ↵
  30.5764415879031,60.0000000000001 30,65.8527096604839 ↵
  30.5764415879031,71.4805029709527 32.2836140246614,76.6671069905881 ↵
  35.0559116309237,81.2132034355964 38.7867965644036,171.213203435596 ↵
  128.786796564404,174.944088369076 133.332893009412,177.716385975339 ↵
  138.519497029047,179.423558412097 144.147290339516,180 150,179.423558412097 ↵
```

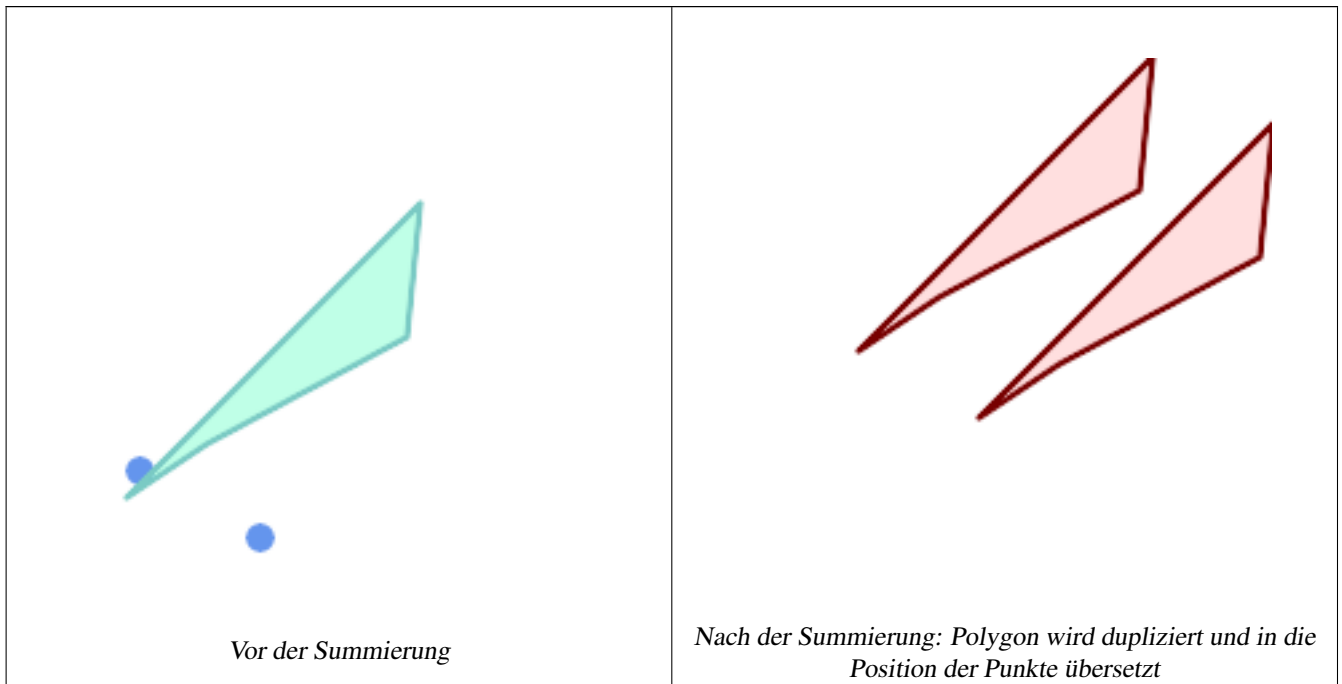


```

155.852709660484,177.716385975339 161.480502970953,174.944088369076 ←
166.667106990588,171.213203435596 171.213203435596,166.667106990588 ←
174.944088369076,
161.480502970953 177.716385975339,155.852709660484 179.423558412097,150 ←
180,144.147290339516 179.423558412097,138.519497029047 ←
177.716385975339,133.332893009412 174.944088369076,128.786796564403 ←
171.213203435596,38.7867965644035 81.2132034355963,35.0559116309236 ←
76.667106990588,32.2836140246614 71.4805029709526,30.5764415879031 ←
65.8527096604838,30 59.9999999999999))

```

### Minkowski-Summe eines Polygons und eines Mehrpunktes



```

SELECT CG_MinkowskiSum(mp, poly)
FROM (SELECT 'MULTIPOINT(25 50,70 25)::geometry As mp,
'POLYGON((130 150, 20 40, 50 60, 125 100, 130 150))::geometry As poly
) As foo

-- wkt --
MULTIPOLYGON(
((70 115,100 135,175 175,225 225,70 115)),
((120 65,150 85,225 125,275 175,120 65))
)

```

### 8.3.29 ST\_OptimalAlphaShape

**ST\_OptimalAlphaShape** — Berechnet eine Alpha-Form, die eine Geometrie umschließt, unter Verwendung eines "optimalen" Alpha-Wertes.

#### Synopsis

geometry **ST\_OptimalAlphaShape**(geometry geom, boolean allow\_holes = false, integer nb\_components = 1);

## Beschreibung

---



### Warning

`ST_OptimalAlphaShape` is deprecated as of 3.5.0. Use `CG_OptimalAlphaShape` instead.

---

Berechnet die "optimale" Alpha-Form der Punkte in einer Geometrie. Die Alpha-Form wird mit einem Wert von  $\alpha$  berechnet, der so gewählt wird, dass:

1. die Anzahl der Polygonelemente ist gleich oder kleiner als `nb_components` (Standardwert: 1)
2. alle Eingabepunkte sind in der Form enthalten

Das Ergebnis wird keine Löcher enthalten, es sei denn, das optionale Argument `allow_holes` wird als `true` angegeben.

Verfügbarkeit: 3.3.0 - erfordert SFCGAL  $\geq$  1.4.1.



Diese Methode benötigt ein SFCGAL-Backend.

### 8.3.30 CG\_OptimalAlphaShape

`CG_OptimalAlphaShape` — Berechnet eine Alpha-Form, die eine Geometrie umschließt, unter Verwendung eines "optimalen" Alpha-Wertes.

#### Synopsis

```
geometry CG_OptimalAlphaShape(geometry geom, boolean allow_holes = false, integer nb_components = 1);
```

#### Beschreibung

Berechnet die "optimale" Alpha-Form der Punkte in einer Geometrie. Die Alpha-Form wird mit einem Wert von  $\alpha$  berechnet, der so gewählt wird, dass:

1. die Anzahl der Polygonelemente ist gleich oder kleiner als `nb_components` (Standardwert: 1)
2. alle Eingabepunkte sind in der Form enthalten

Das Ergebnis wird keine Löcher enthalten, es sei denn, das optionale Argument `allow_holes` wird als `true` angegeben.

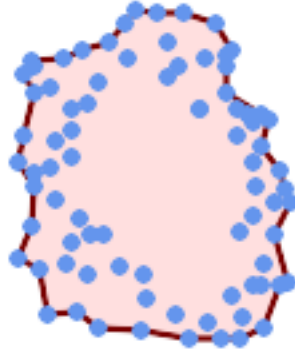
Availability: 3.5.0 - requires SFCGAL  $\geq$  1.4.1.



Diese Methode benötigt ein SFCGAL-Backend.

---

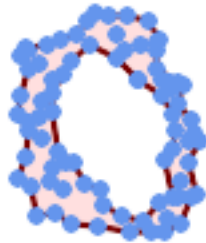
## Beispiele



*Optimal alpha-shape of a MultiPoint (same example as [CG\\_AlphaShape](#))*

```
SELECT ST_AsText(CG_OptimalAlphaShape('MULTIPOINT((63 84),(76 88),(68 73),(53 18),(91 50) ←
, (81 70),
(88 29),(24 82),(32 51),(37 23),(27 54),(84 19),(75 87),(44 42),(77 67),(90 ←
30),(36 61),(32 65),
(81 47),(88 58),(68 73),(49 95),(81 60),(87 50),
(78 16),(79 21),(30 22),(78 43),(26 85),(48 34),(35 35),(36 40),(31 79),(83 ←
29),(27 84),(52 98),(72 95),(85 71),
(75 84),(75 77),(81 29),(77 73),(41 42),(83 72),(23 36),(89 53),(27 57),(57 ←
97),(27 77),(39 88),(60 81),
(80 72),(54 32),(55 26),(62 22),(70 20),(76 27),(84 35),(87 42),(82 54),(83 ←
64),(69 86),(60 90),(50 86),(43 80),(36 73),
(36 68),(40 75),(24 67),(23 60),(26 44),(28 33),(40 32),(43 19),(65 16),(73 ←
16),(38 46),(31 59),(34 86),(45 90),(64 97)')::geometry));
```

```
POLYGON((89 53,91 50,87 42,90 30,88 29,84 19,78 16,73 16,65 16,53 18,43 19,37 23,30 22,28 ←
33,23 36,
26 44,27 54,23 60,24 67,27 77,24 82,26 85,34 86,39 88,45 90,49 95,52 98,57 ←
97,64 97,72 95,76 88,75 84,75 77,83 72,85 71,83 64,88 58,89 53))
```



Optimal alpha-shape of a MultiPoint, allowing holes (same example as [CG\\_AlphaShape](#))

```
SELECT ST_AsText(CG_OptimalAlphaShape('MULTIPOINT((63 84),(76 88),(68 73),(53 18),(91 50) ←
, (81 70),(88 29),(24 82),(32 51),(37 23),(27 54),(84 19),(75 87),(44 42),(77 67),(90 30) ←
, (36 61),(32 65),(81 47),(88 58),(68 73),(49 95),(81 60),(87 50),
(78 16),(79 21),(30 22),(78 43),(26 85),(48 34),(35 35),(36 40),(31 79),(83 29),(27 ←
84),(52 98),(72 95),(85 71),
(75 84),(75 77),(81 29),(77 73),(41 42),(83 72),(23 36),(89 53),(27 57),(57 97),(27 ←
77),(39 88),(60 81),
(80 72),(54 32),(55 26),(62 22),(70 20),(76 27),(84 35),(87 42),(82 54),(83 64),(69 ←
86),(60 90),(50 86),(43 80),(36 73),
(36 68),(40 75),(24 67),(23 60),(26 44),(28 33),(40 32),(43 19),(65 16),(73 16),(38 ←
46),(31 59),(34 86),(45 90),(64 97)')::geometry, allow_holes => true));
```

```
POLYGON((89 53,91 50,87 42,90 30,88 29,84 19,78 16,73 16,65 16,53 18,43 19,37 23,30 22,28 ←
33,23 36,26 44,27 54,23 60,24 67,27 77,24 82,26 85,34 86,39 88,45 90,49 95,52 98,57 ←
97,64 97,72 95,76 88,75 84,75 77,83 72,85 71,83 64,88 58,89 53),(36 61,36 68,40 75,43 ←
80,50 86,60 81,68 73,77 67,81 60,82 54,81 47,78 43,81 29,76 27,70 20,62 22,55 26,54 ←
32,48 34,44 42,38 46,36 61))
```

#### Siehe auch

[ST\\_ConcaveHull](#), [CG\\_AlphaShape](#)

### 8.3.31 CG\_OptimalConvexPartition

CG\_OptimalConvexPartition — Berechnet eine optimale konvexe Partition der Polygoneometrie

#### Synopsis

```
geometry CG_OptimalConvexPartition(geometry geom);
```

#### Beschreibung

Berechnet eine optimale konvexe Partition der Polygoneometrie.

**Note**

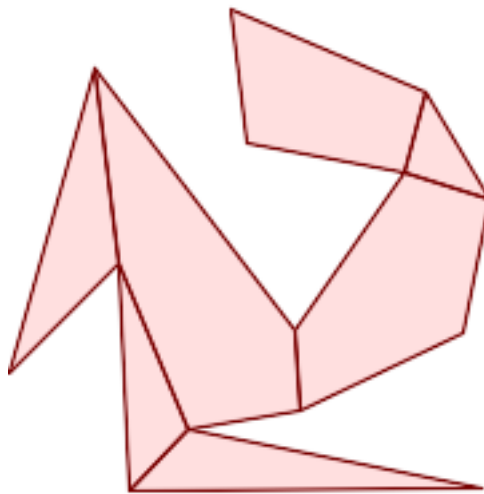
Eine Partition eines Polygons P ist eine Menge von Polygonen, die so beschaffen ist, dass sich die Innenräume der Polygone nicht schneiden und die Vereinigung der Polygone gleich dem Innenraum des ursprünglichen Polygons P ist. `CG_OptimalConvexPartition` erzeugt eine Partition, die hinsichtlich der Anzahl der Teile optimal ist.

Verfügbarkeit: 3.5.0 - erfordert SFCGAL >= 1.5.0.

Erfordert SFCGAL >= 1.5.0



Diese Methode benötigt ein SFCGAL-Backend.

**Beispiele**

*Optimale konvexe Teilung (gleiches Beispiel wie [CG\\_YMonotonePartition](#), [CG\\_ApproxConvexPartition](#) und [CG\\_GreeneApproxConvexPartition](#))*

```
SELECT ST_AsText(CG_OptimalConvexPartition('POLYGON((156 150,83 181,89 131,148 120,107 ←
61,32 159,0 45,41 86,45 1,177 2,67 24,109 31,170 60,180 110,156 150))'::geometry));
```

```
GEOMETRYCOLLECTION(POLYGON((156 150,83 181,89 131,148 120,156 150)),POLYGON((32 159,0 45,41 ←
86,32 159)),POLYGON((45 1,177 2,67 24,45 1)),POLYGON((41 86,45 1,67 24,41 86)),POLYGON ←
((107 61,32 159,41 86,67 24,109 31,107 61)),POLYGON((148 120,107 61,109 31,170 60,180 ←
110,148 120)),POLYGON((156 150,148 120,180 110,156 150)))
```

**Siehe auch**

[CG\\_YMonotonePartition](#), [CG\\_ApproxConvexPartition](#), [CG\\_GreeneApproxConvexPartition](#)

**8.3.32 CG\_StraightSkeleton**

`CG_StraightSkeleton` — Berechnet die konvexe Hülle einer Geometrie.

**Synopsis**

```
geometry CG_StraightSkeleton(geometry geom, boolean use_distance_as_m = false);
```

## Beschreibung

Verfügbarkeit: 3.5.0

Requires SFCGAL >= 1.3.8 for option use\_distance\_as\_m

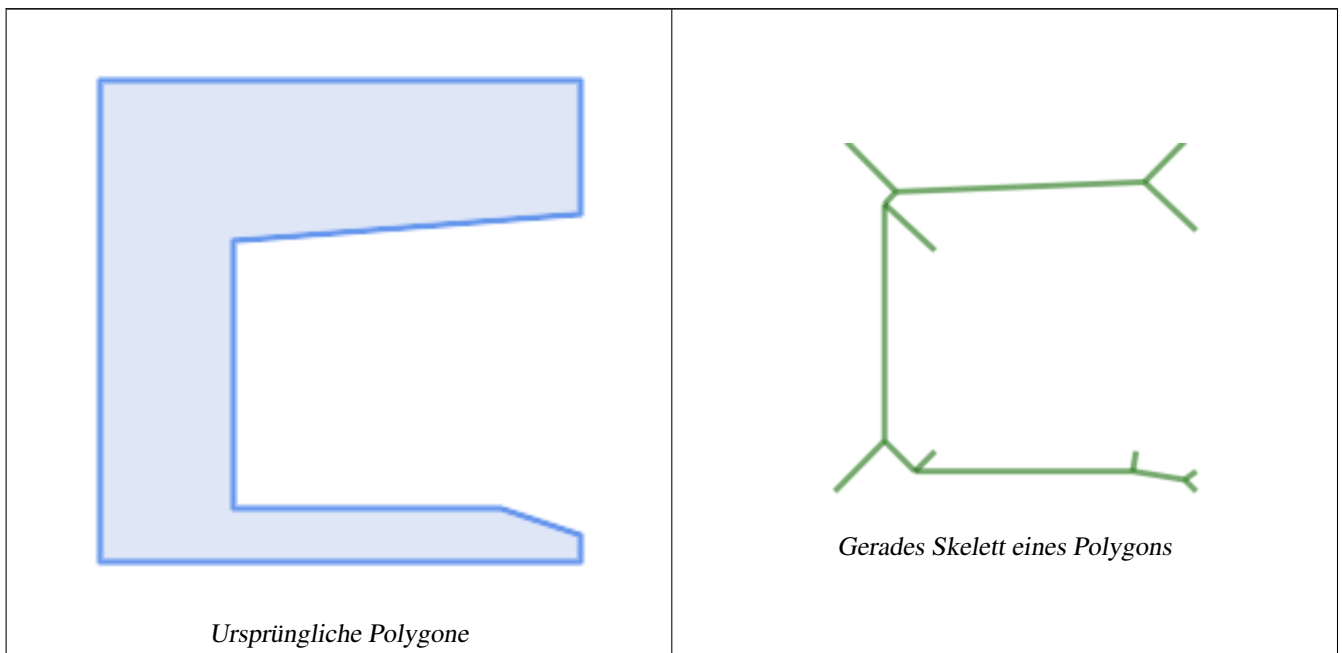
- ✔ Diese Methode benötigt ein SFCGAL-Backend.
- ✔ Diese Funktion unterstützt 3d und lässt den Z-Index nicht fallen.
- ✔ Diese Funktion unterstützt polyedrische Flächen.
- ✔ Diese Funktion unterstützt Dreiecke und dreieckige unregelmäßige Netzoberflächen (TIN).

## Beispiele

```
SELECT CG_StraightSkeleton(ST_GeomFromText('POLYGON (( 190 190, 10 190, 10 10, 190 10, 190 ←
20, 160 30, 60 30, 60 130, 190 140, 190 190 ))'));
```

```
ST_AsText(CG_StraightSkeleton('POLYGON((0 0,1 0,1 1,0 1,0 0))', true);
```

```
MULTILINESTRING M ((0 0 0,0.5 0.5 0.5), (1 0 0,0.5 0.5 0.5), (1 1 0,0.5 0.5 0.5), (0 1 0,0.5 ←
0.5 0.5))
```



## Siehe auch

[CG\\_ExtrudeStraightSkeleton](#)

### 8.3.33 ST\_StraightSkeleton

ST\_StraightSkeleton — Berechnet die konvexe Hülle einer Geometrie.

## Synopsis

geometry **ST\_StraightSkeleton**(geometry geom);

## Beschreibung



### Warning

**ST\_StraightSkeleton** is deprecated as of 3.5.0. Use **CG\_StraightSkeleton** instead.

Verfügbarkeit: 2.1.0



Diese Methode benötigt ein SFCGAL-Backend.



Diese Funktion unterstützt 3d und lässt den Z-Index nicht fallen.



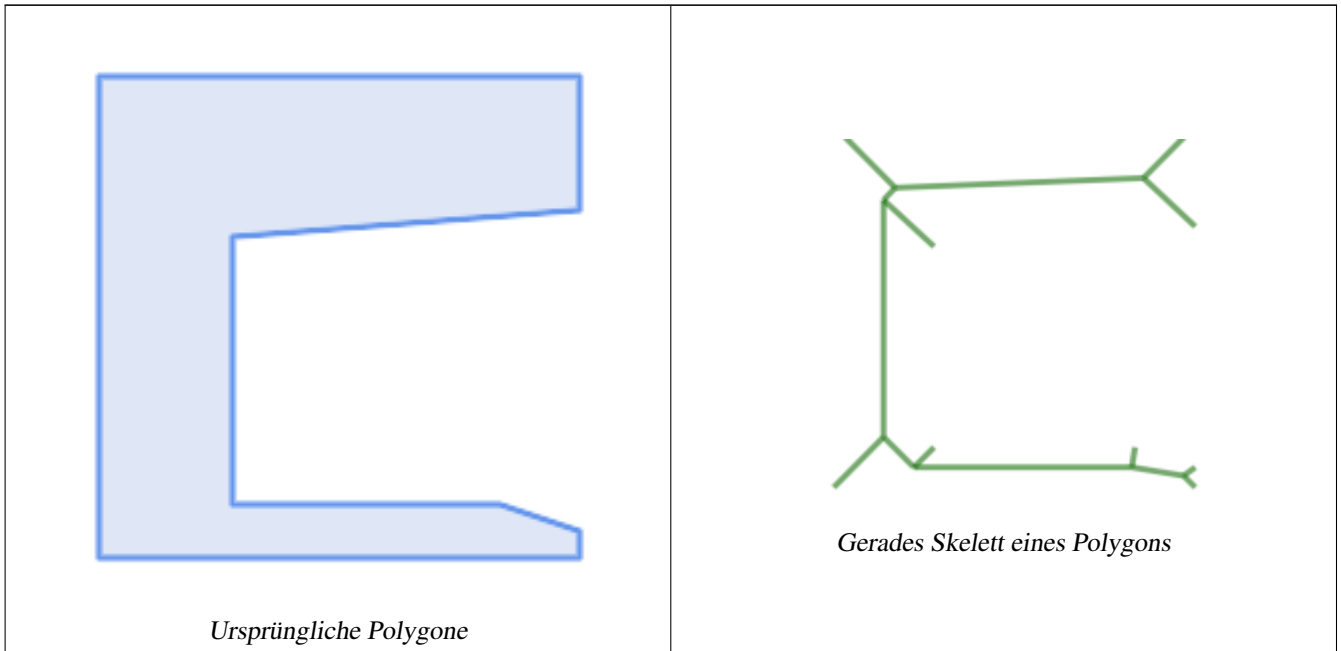
Diese Funktion unterstützt polyedrische Flächen.



Diese Funktion unterstützt Dreiecke und dreieckige unregelmäßige Netzoberflächen (TIN).

## Beispiele

```
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 ))'));
```



## Siehe auch

[CG\\_ExtrudeStraightSkeleton](#)

### 8.3.34 ST\_Tesselate

ST\_Tesselate — Führt eine Oberflächentesselierung eines Polygons oder einer Polyederfläche durch und gibt diese als TIN oder Sammlung von TINs zurück

#### Synopsis

```
geometry ST_Tesselate(geometry geom);
```

#### Beschreibung

**Warning**

ST\_Tesselate is deprecated as of 3.5.0. Use CG\_Tesselate instead.

---

Nimmt als Eingabe eine Oberfläche wie MULTI(POLYGON) oder POLYHEDRALSURFACE und gibt eine TIN-Darstellung über den Prozess der Tesselierung mit Dreiecken zurück.

**Note**

ST\_TriangulatePolygon funktioniert ähnlich wie diese Funktion, mit dem Unterschied, dass sie eine Geometriesammlung von Polygonen anstelle eines TINs zurückgibt und auch nur mit 2D-Geometrien funktioniert.

---

Verfügbarkeit: 2.1.0



Diese Methode benötigt ein SFCGAL-Backend.



Diese Funktion unterstützt 3d und lässt den Z-Index nicht fallen.



Diese Funktion unterstützt polyedrische Flächen.



Diese Funktion unterstützt Dreiecke und dreieckige unregelmäßige Netzoberflächen (TIN).

### 8.3.35 CG\_Tesselate

CG\_Tesselate — Führt eine Oberflächentesselierung eines Polygons oder einer Polyederfläche durch und gibt diese als TIN oder Sammlung von TINs zurück

#### Synopsis

```
geometry CG_Tesselate(geometry geom);
```

#### Beschreibung

Nimmt als Eingabe eine Oberfläche wie MULTI(POLYGON) oder POLYHEDRALSURFACE und gibt eine TIN-Darstellung über den Prozess der Tesselierung mit Dreiecken zurück.


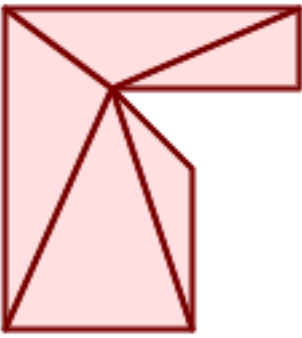
**Note**

ST\_TriangulatePolygon funktioniert ähnlich wie diese Funktion, mit dem Unterschied, dass sie eine Geometriesammlung von Polygonen anstelle eines TINs zurückgibt und auch nur mit 2D-Geometrien funktioniert.

---





<pre>SELECT 'POLYGON (( 10 190, 10 70, 80 70, ↵       80 130, 50 160, 120 160, 120 190, 10 190 )</pre>  <p style="text-align: center;"><i>Ursprüngliche Polygone</i></p>	<pre>SELECT       CG_Tesselate(' ↵       POLYGON (( 10 190, 10 70, 80 70, 80 130, 50 160, ↵       ;       ST_AsText Ausgabe       geometry;       TIN(((80 130, 50 160, 80 70, 80 130)), ((50 ↵       160, 10 190, 10 70, 50 160)),       ((80 70, 50 160, 10 70, 80 ↵       70)), ((120 160, 120 190, 50 160, 120 160)),       ((120 190, 10 190, 50 ↵       160, 120 190)))</pre>  <p style="text-align: center;"><i>Mosaikförmiges Polygon</i></p>
---	--

**Siehe auch**

[CG\\_ConstrainedDelaunayTriangles](#), [ST\\_DelaunayTriangles](#), [ST\\_TriangulatePolygon](#)

**8.3.36 CG\_Triangulate**

CG\_Triangulate — Triangulates a polygonal geometry

**Synopsis**

geometry **CG\_Triangulate**( geometry geom );

**Beschreibung**

Triangulates a polygonal geometry.  
 Performed by the SFCGAL module



**Note**

NOTE: this function returns a geometry representing the triangulated result.

Verfügbarkeit: 3.5.0



Diese Methode benötigt ein SFCGAL-Backend.

### Beispiele mit dem geometrischen Datentyp

```
SELECT CG_Triangulate('POLYGON((0.0 0.0,1.0 0.0,1.0 1.0,0.0 1.0,0.0 0.0), (0.2 0.2,0.2 0.8,0.8 0.8 0.8,0.8 0.2,0.2 0.2))');
      cg_triangulate
      -----
      TIN(((0.8 0.2,0.2 0.2,1 0,0.8 0.2)),((0.2 0.2,0 0,1 0,0.2 0.2)),((1 1,0.8 0.8,0.8 0.2,1 1)),((0 1,0 0,0.2 0.2,0 1)),((0 1,0.2 0.8,1 1,0 1)),((0 1,0.2 0.2,0.2 0.8,0 1)),((0.2 0.8,0.8 0.8,1 1,0.2 0.8)),((0.2 0.8,0.2 0.2,0.8 0.2,0.2 0.8)),((1 1,0.8 0.2,1 0,1 1)),((0.8 0.8,0.2 0.8,0.8 0.2,0.8 0.8)))

(1 row)
```

### Siehe auch

[CG\\_ConstrainedDelaunayTriangles](#), [ST\\_DelaunayTriangles](#), [ST\\_TriangulatePolygon](#)

## 8.3.37 CG\_Visibility

CG\_Visibility — Berechnen eines Sichtbarkeitspolygons aus einem Punkt oder einem Segment in einer Polygoneometrie

### Synopsis

```
geometry CG_Visibility(geometry polygon, geometry point);
geometry CG_Visibility(geometry polygon, geometry pointA, geometry pointB);
```

### Beschreibung

Verfügbarkeit: 3.5.0 - erfordert SFCGAL >= 1.5.0.

Erfordert SFCGAL >= 1.5.0



Diese Methode benötigt ein SFCGAL-Backend.



Diese Funktion unterstützt 3d und lässt den Z-Index nicht fallen.



Diese Funktion unterstützt polyedrische Flächen.



Diese Funktion unterstützt Dreiecke und dreieckige unregelmäßige Netzoberflächen (TIN).

### Beispiele

```
SELECT CG_Visibility('POLYGON((23.5 23.5,23.5 173.5,173.5 173.5,173.5 23.5,23.5 23.5), (108 98,108 36,156 37,155 99,108 98), (107 157.5,107 106.5,135 107.5,133 127.5,143.5 127.5,143.5 108.5,153.5 109.5,151.5 166,107 157.5), (41 95.5,41 35,100.5 36,98.5 68,78.5 68,77.5 96.5,41 95.5), (39 150,40 104,97.5 106.5,95.5 152,39 150))'::geometry, 'POINT(91 87)'::geometry);
```

```
SELECT CG_Visibility('POLYGON((23.5 23.5,23.5 173.5,173.5 173.5,173.5 23.5,23.5 23.5),(108 ←
98,108 36,156 37,155 99,108 98),(107 157.5,107 106.5,135 107.5,133 127.5,143.5 ←
127.5,143.5 108.5,153.5 109.5,151.5 166,107 157.5),(41 95.5,41 35,100.5 36,98.5 68,78.5 ←
68,77.5 96.5,41 95.5),(39 150,40 104,97.5 106.5,95.5 152,39 150))'::geometry, 'POINT(78.5 ←
68)'::geometry, 'POINT(98.5 68)'::geometry);
```



### 8.3.38 CG\_YMonotonePartition

CG\_YMonotonePartition — Berechnet die y-monotone Partition der Polygoneometrie


#### Synopsis

geometry **CG\_YMonotonePartition**(geometry geom);

#### Beschreibung

Berechnet die y-monotone Partition der Polygoneometrie.

---

**Note**  
 Eine Partition eines Polygons P ist eine Menge von Polygonen, die so beschaffen ist, dass sich die Innenräume der Polygone nicht schneiden und die Vereinigung der Polygone gleich dem Innenraum des ursprünglichen Polygons P ist. Ein y-monotones Polygon ist ein Polygon, dessen Scheitelpunkte  $v_1, \dots, v_n$  in zwei Ketten  $v_1, \dots, v_k$  und  $v_k, \dots, v_n, v_1$  unterteilt werden können, so dass jede horizontale Linie eine der beiden Ketten höchstens einmal schneidet. Dieser Algorithmus garantiert keine Begrenzung der Anzahl der erzeugten Polygone in Bezug auf die optimale Anzahl.

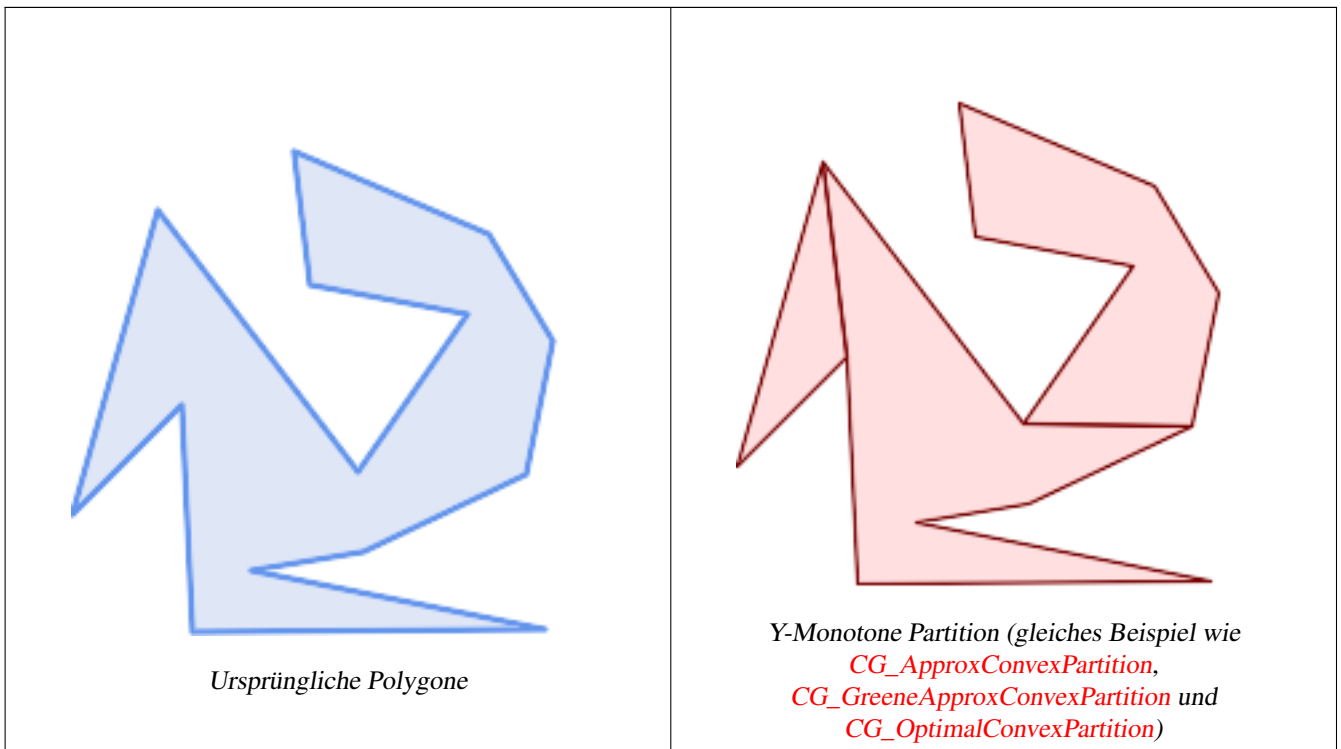
---

Verfügbarkeit: 3.5.0 - erfordert SFCGAL >= 1.5.0.

Erfordert SFCGAL >= 1.5.0

 Diese Methode benötigt ein SFCGAL-Backend.

#### Beispiele



```
SELECT ST_AsText(CG_YMonotonePartition('POLYGON((156 150,83 181,89 131,148 120,107 61,32 ↵
159,0 45,41 86,45 1,177 2,67 24,109 31,170 60,180 110,156 150))'::geometry));
```

```
GEOMETRYCOLLECTION(POLYGON((32 159,0 45,41 86,32 159)),POLYGON((107 61,32 159,41 86,45 ↵
1,177 2,67 24,109 31,170 60,107 61)),POLYGON((156 150,83 181,89 131,148 120,107 61,170 ↵
60,180 110,156 150)))
```

### Siehe auch

[CG\\_ApproxConvexPartition](#), [CG\\_GreeneApproxConvexPartition](#), [CG\\_OptimalConvexPartition](#)

## Chapter 9

# Topologie

Die topologischen Datentypen und Funktionen von PostGIS werden für die Verwaltung von topologischen Objekten wie Maschen, Kanten und Knoten verwendet.

Sandro Santilli's Vortrag auf der Tagung "PostGIS Day Paris 2011" liefert eine gute Übersicht über die PostGIS Topologie und deren Perspektiven [Topology with PostGIS 2.0 slide deck](#).

Vincent Picavet gibt in [PostGIS Topology PGConf EU 2012](#) einen guten Überblick darüber, was Topologie ist, wie sie verwendet wird und welche FOSS4G-Tools sie unterstützen.

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.

## 9.1 Topologische Datentypen

### 9.1.1 `getfaceedges_returntype`

`getfaceedges_returntype` — Ein zusammengesetzter Typ, der aus einer Sequenznummer und einer Randnummer besteht.

#### Beschreibung

Ein zusammengesetzter Typ, der aus einer Sequenznummer und einer Kantenummer besteht. Dies ist der Rückgabetyt für die Funktionen `ST_GetFaceEdges` und `GetNodeEdges`.

1. `sequence` ist eine Ganzzahl: Sie verweist auf eine Topologie, die in der Tabelle `topology.topology` definiert ist. In dieser Tabelle sind das Schema, das die Topologie enthält, und die SRID verzeichnet.
  2. `edge` ist eine Ganzzahl: Der Identifikator einer Kante.
-

## 9.1.2 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](#)

## 9.1.3 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 mismatch, 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](#)

## 9.2 Topologische Domänen

### 9.2.1 TopoElement

TopoElement — Ein Feld mit 2 Ganzzahlen, welches in der Regel für die Auffindung einer Komponente einer TopoGeometry dient.

**Beschreibung**

Ein Feld mit 2 Ganzzahlen, welches einen Bestandteil einer einfachen oder hierarchischen **TopoGeometry** abbildet.

Im Falle einer einfachen TopoGeometry ist das erste Element des Feldes der Identifikator einer topologischen Elementarstruktur und das zweite Element der Typ (1:Knoten, 2:Kante, 3:Masche). Im Falle einer hierarchischen TopoGeometry ist das erste Element des Feldes der Identifikator der Kind-TopoGeometry und das zweite Element der Identifikator des Layers.

**Note**

Bei jeder gegebenen hierarchischen TopoGeometry werden alle Kindklassen der TopoGeometry vom selben Kindlayer abgeleitet, so wie dies in dem Datensatz von "topology.layer" für den Layer der TopoGeometry definiert ist.

**Beispiele**

```
SELECT te[1] AS id, te[2] AS type FROM
( SELECT ARRAY[1,2]::topology.topoelement AS te ) f;
 id | type
-----+-----
  1 |    2
```

```
SELECT ARRAY[1,2]::topology.topoelement;
 te
-----
{1,2}
```

```
--Example of what happens when you try to case a 3 element array to topoelement
-- NOTE: topoement has to be a 2 element array so fails dimension check
SELECT ARRAY[1,2,3]::topology.topoelement;
ERROR:  value for domain topology.topoelement violates check constraint "dimensions"
```

**Siehe auch**

[GetTopoGeomElements](#), [TopoElementArray](#), [TopoGeometry](#), [TopoGeom\\_addElement](#), [TopoGeom\\_remElement](#)

### 9.2.2 TopoElementArray

TopoElementArray — Ein Feld mit TopoElement Objekten.



## Beschreibung

Ein Feld mit 1 oder mehreren TopoElement Objekten; wird hauptsächlich verwendet, um Bestandteile einer TopoGeometry heranzureichen.

## Beispiele

```
SELECT '{{1,2},{4,3}}'::topology.topoelementarray As tea;
   tea
-----
{{1,2},{4,3}}

-- more verbose equivalent --
SELECT ARRAY[ARRAY[1,2], ARRAY[4,3]]::topology.topoelementarray As tea;

   tea
-----
{{1,2},{4,3}}

--using the array agg function packaged with topology --
SELECT topology.TopoElementArray_Agg(ARRAY[e,t]) As tea
   FROM generate_series(1,4) As e CROSS JOIN generate_series(1,3) As t;
   tea
-----
{{1,1},{1,2},{1,3},{2,1},{2,2},{2,3},{3,1},{3,2},{3,3},{4,1},{4,2},{4,3}}
```

```
SELECT '{{1,2,4},{3,4,5}}'::topology.topoelementarray As tea;
ERROR:  value for domain topology.topoelementarray violates check constraint "dimensions"
```

## Siehe auch

[TopoElement](#), [GetTopoGeomElementArray](#), [TopoElementArray\\_Agg](#)

## 9.3 Verwaltung von Topologie und TopoGeometry

### 9.3.1 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: PUNKT, MULTIPUNKT, LINIE, MULTILINIE, POLYGON, MULTIPOLYGON, SAMMLUNG

Verfügbarkeit: 1.1

### Beispiele

```
-- Note for this example we created our new table in the ma_topo schema
-- though we could have created it in a different schema -- in which case topology_name and ←
  schema_name would be different
CREATE SCHEMA ma;
CREATE TABLE ma.parcels(gid serial, parcel_id varchar(20) PRIMARY KEY, address text);
SELECT topology.AddTopoGeometryColumn('ma_topo', 'ma', 'parcels', 'topo', 'POLYGON');
```

```
CREATE SCHEMA ri;
CREATE TABLE ri.roads(gid serial PRIMARY KEY, road_name text);
SELECT topology.AddTopoGeometryColumn('ri_topo', 'ri', 'roads', 'topo', 'LINE');
```

### Siehe auch

[DropTopoGeometryColumn](#), [toTopoGeom](#), [CreateTopology](#), [CreateTopoGeom](#)

## 9.3.2 RenameTopoGeometryColumn

`RenameTopoGeometryColumn` — Benennt eine topogeometrische Spalte um

### Synopsis

`topology.layer` **RenameTopoGeometryColumn**(regclass layer\_table, name feature\_column, name new\_name);

### Beschreibung

Diese Funktion ändert den Namen einer bestehenden TopoGeometry-Spalte und stellt sicher, dass die Metadateninformationen entsprechend aktualisiert werden.

Verfügbarkeit: 3.4.0

### Beispiele

```
SELECT topology.RenameTopoGeometryColumn('public.parcels', 'topogeom', 'tgeom');
```

### Siehe auch

[AddTopoGeometryColumn](#), [RenameTopology](#)

---

### 9.3.3 DropTopology

DropTopology — Bitte mit Vorsicht verwenden: Löscht ein topologisches Schema und dessen Referenz in der Tabelle topology.topology, sowie die Referenzen zu den Tabellen in diesem Schema aus der Tabelle geometry\_columns.

#### Synopsis

```
integer DropTopology(varchar topology_schema_name);
```

#### Beschreibung

Löscht ein topologisches Schema und dessen Referenz in der Tabelle topology.topology, sowie die Referenzen zu den Tabellen in diesem Schema aus der Tabelle geometry\_columns. Diese Funktion sollte MIT VORSICHT BENUTZT werden, da damit unabsichtlich Daten zerstört werden können. Falls das Schema nicht existiert, werden nur die Referenzeinträge des bezeichneten Schemas gelöscht.

Verfügbarkeit: 1.1

#### Beispiele

Löscht das Schema "ma\_topo" kaskadierend und entfernt alle Referenzen in topology.topology und in geometry\_columns.

```
SELECT topology.DropTopology('ma_topo');
```

#### Siehe auch

[DropTopoGeometryColumn](#)

### 9.3.4 RenameTopology

RenameTopology — Benennt eine Topologie um

#### Synopsis

```
varchar RenameTopology(varchar old_name, varchar new_name);
```

#### Beschreibung

Benennt ein Topologieschema um und aktualisiert seinen Metadatensatz in der Tabelle topology.topology.

Verfügbarkeit: 3.4.0

#### Beispiele

Umbenennen einer Topologie von topo\_stage in topo\_prod.

```
SELECT topology.RenameTopology('topo_stage', 'topo_prod');
```

#### Siehe auch

[CopyTopology](#), [RenameTopoGeometryColumn](#)

---

### 9.3.5 DropTopoGeometryColumn

`DropTopoGeometryColumn` — Entfernt ein `TopoGeometry`-Attribut aus der Tabelle mit der Bezeichnung `table_name` im Schema `schema_name` und entfernt die Registrierung der Attribute aus der Tabelle "topology.layer".

#### Synopsis

```
text DropTopoGeometryColumn(varchar schema_name, varchar table_name, varchar column_name);
```

#### Beschreibung

Entfernt ein `TopoGeometry`-Attribut aus der Tabelle mit der Bezeichnung `table_name` im Schema `schema_name` und entfernt die Registrierung der Attribute aus der Tabelle "topology.layer". Gibt eine Zusammenfassung des Löschstaus aus. ANMERKUNG: vor dem Löschen werden alle Werte auf `NULL` gesetzt, um die Überprüfung der referenziellen Integrität zu umgehen.

Verfügbarkeit: 1.1

#### Beispiele

```
SELECT topology.DropTopoGeometryColumn('ma_topo', 'parcel_topo', 'topo');
```

#### Siehe auch

[AddTopoGeometryColumn](#)

### 9.3.6 Populate\_Topology\_Layer

`Populate_Topology_Layer` — Fügt fehlende Einträge zu der Tabelle `topology.layer` hinzu, indem Metadaten aus den topologischen Tabellen ausgelesen werden.

#### Synopsis

```
setof record Populate_Topology_Layer();
```

#### Beschreibung

Trägt fehlende Einträge in der Tabelle `topology.layer` ein, indem die topologischen Constraints und Tabellen inspiziert werden. Diese Funktion ist nach der Wiederherstellung von Schemata mit topologischen Daten sinnvoll, um Einträge im Topologie-Katalog zu reparieren.

Wirft eine Liste der erzeugten Einträge aus. Die zurückgegebenen Attribute sind `schema_name`, `table_name` und `feature_colu`

Verfügbarkeit: 2.3.0

## Beispiele

```
SELECT CreateTopology('strk_topo');
CREATE SCHEMA strk;
CREATE TABLE strk.parcels(gid serial, parcel_id varchar(20) PRIMARY KEY, address text);
SELECT topology.AddTopoGeometryColumn('strk_topo', 'strk', 'parcels', 'topo', 'POLYGON');
-- this will return no records because this feature is already registered
SELECT *
  FROM topology.Populate_Topology_Layer();

-- let's rebuild
TRUNCATE TABLE topology.layer;

SELECT *
  FROM topology.Populate_Topology_Layer();

SELECT topology_id,layer_id, schema_name As sn, table_name As tn, feature_column As fc
FROM topology.layer;
```

```
schema_name | table_name | feature_column
-----+-----+-----
strk        | parcels    | topo
(1 row)

topology_id | layer_id | sn | tn | fc
-----+-----+-----+-----+-----
          2 |         2 | strk | parcels | topo
(1 row)
```

## Siehe auch

[AddTopoGeometryColumn](#)

### 9.3.7 TopologySummary

**TopologySummary** — Nimmt den Namen einer Topologie und liefert eine Zusammenfassung der Gesamtsummen der Typen und Objekte in der Topologie.

#### Synopsis

```
text TopologySummary(varchar topology_schema_name);
```

#### Beschreibung

Nimmt den Namen einer Topologie und liefert eine Zusammenfassung der Gesamtsummen der Typen und Objekte in der Topologie.

Verfügbarkeit: 2.0.0

#### Beispiele

```
SELECT topology.topologysummary('city_data');
           topologysummary
-----
Topology city_data (329), SRID 4326, precision: 0
22 nodes, 24 edges, 10 faces, 29 topogeoms in 5 layers
Layer 1, type Polygonal (3), 9 topogeoms
  Deploy: features.land_parcels.feature
Layer 2, type Puntal (1), 8 topogeoms
  Deploy: features.traffic_signs.feature
Layer 3, type Lineal (2), 8 topogeoms
  Deploy: features.city_streets.feature
Layer 4, type Polygonal (3), 3 topogeoms
  Hierarchy level 1, child layer 1
  Deploy: features.big_parcels.feature
Layer 5, type Puntal (1), 1 topogeoms
  Hierarchy level 1, child layer 2
  Deploy: features.big_signs.feature
```

**Siehe auch**[Topology\\_Load\\_Tiger](#)**9.3.8 ValidateTopology**

ValidateTopology — Liefert eine Menge `validatetopology_returntype` Objekte, die Probleme mit der Topologie beschreiben.

**Synopsis**

```
setof validatetopology_returntype ValidateTopology(varchar toponame, geometry bbox);
```

**Beschreibung**

Gibt eine Reihe von `validatetopology_returntype` Objekten zurück, die Probleme mit der Topologie aufzeigen, wobei die Prüfung optional auf den durch den Parameter `bbox` angegebenen Bereich beschränkt werden kann.

Die Liste der möglichen Fehler, ihre Bedeutung und die zurückgegebenen IDs sind unten aufgeführt:

<b>Error</b>	<b>id1</b>	<b>id2</b>	<b>Bedeutung</b>
übereinstimmende Knotenpunkte	Kennung des ersten Knotens.	Kennung des zweiten Knotens.	Zwei Knoten haben die gleiche Geometrie.
edge crosses node	Kennung der Kante.	Bezeichner des Knotens.	Eine Kante hat einen Knoten in ihrem Inneren. Siehe <a href="#">ST_Relate</a> .
invalid edge	Kennung der Kante.		Eine Kantengeometrie ist ungültig. Siehe <a href="#">ST_IsValid</a> .
edge not simple	Kennung der Kante.		Eine Kantengeometrie hat Selbstschnittpunkte. Siehe <a href="#">ST_IsSimple</a> .
edge crosses edge	Kennung der ersten Kante.	Kennung der zweiten Kante.	Zwei Kanten haben einen inneren Schnittpunkt. Siehe <a href="#">ST_Relate</a> .

<b>Error</b>	<b>id1</b>	<b>id2</b>	<b>Bedeutung</b>
edge start node geometry mismatch	Kennung der Kante.	Kennung des angegebenen Startknotens.	Die Geometrie des Knotens, der als Startknoten für eine Kante angegeben ist, stimmt nicht mit dem ersten Punkt der Kantengeometrie überein. Siehe <a href="#">ST_StartPoint</a> .
edge end node geometry mismatch	Kennung der Kante.	Kennung des angegebenen Endknotens.	Die Geometrie des Knotens, der als Endknoten für eine Kante angegeben ist, stimmt nicht mit dem letzten Punkt der Kantengeometrie überein. Siehe <a href="#">ST_EndPoint</a> .
face without edges	Kennung des verwaisten Gesichts.		Keine Kante meldet eine vorhandene Fläche auf einer ihrer Seiten ( <code>left_face</code> , <code>right_face</code> ).
face has no rings	Kennung der teilweise definierten Fläche.		Kanten, die an ihren Seiten eine Fläche aufweisen, bilden keinen Ring.
Gesicht hat falsches mbr	Kennung des Gesichts mit falschem mbr-Cache.		Das minimale Begrenzungsrechteck einer Fläche stimmt nicht mit dem minimalen Begrenzungsrahmen der Sammlung von Kanten überein, die die Fläche an ihren Seiten darstellen.
Loch nicht im beworbenen Gesicht	Signierter Bezeichner einer Kante, der den Ring identifiziert. Siehe <a href="#">GetRingEdges</a> .		Ein Ring von Kanten, der eine Fläche auf seiner Außenseite meldet, ist in einer anderen Fläche enthalten.
nicht-isolierter Knoten hat nicht-enthaltende_Fläche	Kennung des schlecht definierten Knotens.		Ein Knoten, der als an der Grenze einer oder mehrerer Kanten liegend gemeldet wird, zeigt eine enthaltende Fläche an.
isolierter Knoten hat containing_face	Kennung des schlecht definierten Knotens.		Einem Knoten, der nicht als auf der Grenze einer Kante liegend gemeldet wird, fehlt der Hinweis auf eine enthaltende Fläche.
isolierter Knoten hat falsche containing_face	Kennung des falsch dargestellten Knotens.		Ein Knoten, der nicht als auf dem Rand einer Kante liegend gemeldet wird, weist auf eine enthaltende Fläche hin, die nicht die eigentliche Fläche ist, die ihn enthält. Siehe <a href="#">GetFaceContainingPoint</a> .
ungültig next_right_edge	Kennung der falsch dargestellten Kante.	Vorzeichenlose ID der Kante, die als nächste rechte Kante angezeigt werden soll.	Die Kante, die als nächste Kante angegeben ist, wenn man auf der rechten Seite einer Kante geht, ist falsch.

Error	id1	id2	Bedeutung
ungültig nächster_linker_Rand	Kennung der falsch dargestellten Kante.	Signierte ID der Kante, die als nächste linke Kante angezeigt werden soll.	Die Kante, die als nächste Kante angegeben ist, wenn man auf der linken Seite einer Kante geht, ist falsch.
gemischte Gesichtsauszeichnung im Ring	Signierter Bezeichner einer Kante, der den Ring identifiziert. Siehe <a href="#">GetRingEdges</a> .		Kanten in einem Ring weisen auf kollidierende Flächen auf der gehenden Seite hin. Dies wird auch als "Seitenlagekonflikt" bezeichnet.
nicht geschlossener Ring	Signierter Bezeichner einer Kante, der den Ring identifiziert. Siehe <a href="#">GetRingEdges</a> .		Ein Ring von Kanten, der durch die Attribute next_left_edge/next_right_edge gebildet wird, beginnt und endet an verschiedenen Knoten.
Gesicht hat mehrere Schalen	Kennung der umstrittenen Fläche.	Signierter Bezeichner einer Kante, der den Ring identifiziert. Siehe <a href="#">GetRingEdges</a> .	Mehr als ein Kranz von Kanten weist auf dieselbe Fläche im Inneren hin.

Verfügbarkeit: 1.0.0

Erweiterung: 2.0.0 effizientere Ermittlung sich überkreuzender Kanten. Falsch positive Fehlmeldungen von früheren Versionen fixiert.

Änderung: 2.2.0 Bei 'edge crosses node' wurden die Werte für id1 und id2 vertauscht, um mit der Fehlerbeschreibung konsistent zu sein.

Geändert: 3.2.0 fügte den optionalen bbox-Parameter hinzu und führte Prüfungen der Flächenbeschriftung und der Kantenverknüpfung durch.

### Beispiele

```
SELECT * FROM topology.ValidateTopology('ma_topo');
      error      | id1 | id2
-----+-----+-----
face without edges | 1 |
```

### Siehe auch

[validatetopology\\_returntype](#), [Topology\\_Load\\_Tiger](#)

## 9.3.9 ValidateTopologyRelation

ValidateTopologyRelation — Gibt Informationen über ungültige Topologiebeziehungssätze zurück

### Synopsis

```
setof record ValidateTopologyRelation(varchar toponame);
```



## Beschreibung

Gibt eine Reihe von Datensätzen mit Informationen über Ungültigkeiten in der Beziehungstabelle der Topologie zurück.

Verfügbarkeit: 3.2.0

## Siehe auch

[ValidateTopology](#)

## 9.3.10 FindTopology

FindTopology — Gibt einen Topologie-Datensatz mit anderen Mitteln zurück.

## Synopsis

```
topology FindTopology(TopoGeometry topogeom);
topology FindTopology(regclass layerTable, name layerColumn);
topology FindTopology(name layerSchema, name layerTable, name layerColumn);
topology FindTopology(text topoName);
topology FindTopology(int id);
```

## Beschreibung

Nimmt einen Topologiebezeichner oder den Bezeichner eines topologiebezogenen Objekts und gibt einen topology.topology-Datensatz zurück.

Verfügbarkeit: 3.2.0

## Beispiele

```
SELECT name(findTopology('features.land_parcels', 'feature'));
   name
-----
city_data
(1 row)
```

## Siehe auch

[FindLayer](#)

## 9.3.11 FindLayer

FindLayer — Gibt einen topology.layer-Datensatz mit anderen Mitteln zurück.

## Synopsis

```
topology.layer FindLayer(TopoGeometry tg);
topology.layer FindLayer(regclass layer_table, name feature_column);
topology.layer FindLayer(name schema_name, name table_name, name feature_column);
topology.layer FindLayer(integer topology_id, integer layer_id);
```

## Beschreibung

Nimmt einen Layer-Identifikator oder den Identifikator eines topologiebezogenen Objekts und gibt einen `topology.layer`-Datensatz zurück.

Verfügbarkeit: 3.2.0

## Beispiele

```
SELECT layer_id(findLayer('features.land_parcels', 'feature'));
 layer_id
-----
         1
(1 row)
```

## Siehe auch

[FindTopology](#)

## 9.4 Verwaltung der Topologie-Statistiken

Das Hinzufügen von Elementen zu einer Topologie löst viele Datenbankabfragen aus, um bestehende Kanten zu finden, die geteilt werden, Knoten hinzuzufügen und Kanten zu aktualisieren, die mit dem neuen Liniennetz verknötet werden. Aus diesem Grund ist es sinnvoll, dass die Statistiken über die Daten in den Topologietabellen aktuell sind.

Die PostGIS-Topologiebevölkerungs- und -bearbeitungsfunktionen aktualisieren die Statistiken nicht automatisch, da eine Aktualisierung der Statistiken nach jeder einzelnen Änderung in einer Topologie zu viel Aufwand wäre.



### Note

Die von Autovacuum aktualisierten Statistiken sind für Transaktionen, die vor Beendigung des Autovacuum-Prozesses gestartet wurden, NICHT sichtbar, so dass lang laufende Transaktionen selbst ANALYZE ausführen müssen, um aktualisierte Statistiken zu verwenden.

## 9.5 Topologie Konstruktoren

### 9.5.1 CreateTopology

`CreateTopology` — Erstellt ein neues Topologie-Schema und trägt es in die Tabelle `topology.topology` ein.

#### 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

Erstellt ein neues Topologie-Schema mit dem Namen `topology_name` und registriert es in der Tabelle `topology.topology`. Topologien müssen eindeutig benannt sein. Die Topologietabellen (`edge_data`, `face`, `node`, und `relation` werden im Schema erstellt. Sie gibt die ID der Topologie zurück.

Das `srid` ist das **Raumbezugssystem** SRID für die Topologie.

Die Toleranz `prec` wird in den Einheiten des räumlichen Bezugssystems gemessen. Der Standardwert für die Toleranz ist 0.

`hasz` ist standardmäßig `false`, wenn nicht angegeben.

Sie ähnelt der SQL/MM **ST\_InitTopoGeo**, hat aber mehr Funktionen.

Verfügbarkeit: 1.1

Verbessert: 2.0 fügte die Signatur hinzu, die `hasZ` akzeptiert

## Beispiele

Erstellen Sie ein Topologieschema mit der Bezeichnung `ma_topo`, das Kanten und Knoten in Massachusetts State Plane-Metern (SRID = 26986) speichert. Die Toleranz beträgt 0,5 Meter, da das räumliche Bezugssystem meterbasiert ist.

```
SELECT topology.CreateTopology('ma_topo', 26986, 0.5);
```

Erstellen einer Topologie für Rhode Island mit der Bezeichnung `ri_topo` im Raumbezugssystem State Plane-feet (SRID = 3438)

```
SELECT topology.CreateTopology('ri_topo', 3438) AS topoid;
topoid
-----
2
```

## Siehe auch

Section [4.5](#), [ST\\_InitTopoGeo](#), [Topology\\_Load\\_Tiger](#)

## 9.5.2 CopyTopology

**CopyTopology** — Erstellt eine Kopie einer Topologie (Knoten, Kanten, Flächen, Ebenen und TopoGeometrien) in ein neues Schema

### Synopsis

integer **CopyTopology**(varchar existing\_topology\_name, varchar new\_name);

### Beschreibung

Erstellt eine neue Topologie mit dem Namen `new_name`, wobei SRID und Präzision von `existing_topology_name` kopiert werden. Die Knoten, Kanten und Flächen in `existing_topology_name` werden in die neue Topologie kopiert, ebenso wie die Layer und ihre zugehörigen TopoGeometrien.



#### Note

Die neuen Zeilen in der Tabelle `topology.layer` enthalten synthetische Werte für `schema_name`, `table_name` und `feature_column`. Dies liegt daran, dass die TopoGeometry-Objekte nur als Definition existieren und noch nicht in einer benutzerdefinierten Tabelle verfügbar sind.

Verfügbarkeit: 2.0.0

## Beispiele

Erstellen Sie ein Backup einer Topologie mit dem Namen `ma_topo`.

```
SELECT topology.CopyTopology('ma_topo', 'ma_topo_backup');
```

## Siehe auch

Section [4.5](#), [CreateTopology](#), [RenameTopology](#)

## 9.5.3 ST\_InitTopoGeo

`ST_InitTopoGeo` — Erstellt ein neues Topologie-Schema und trägt es in die Tabelle `topology.topology` ein.

### Synopsis

```
text ST_InitTopoGeo(varchar topology_schema_name);
```

### Beschreibung

Dies ist das SQL-MM-Äquivalent zu [CreateTopology](#). Es fehlen die Optionen für das räumliche Bezugssystem und die Toleranz. Es liefert eine Textbeschreibung der Topologierstellung anstelle der Topologie-ID.

Verfügbarkeit: 1.1



Diese Methode setzt die SQL/MM-Spezifikation um. SQL-MM 3 Topo-Geo and Topo-Net 3: Routine Details: X.3.17

## Beispiele

```
SELECT topology.ST_InitTopoGeo('topo_schema_to_create') AS topocreation;
           astopocreation
-----
Topology-Geometry 'topo_schema_to_create' (id:7) created.
```

## Siehe auch

[CreateTopology](#)

## 9.5.4 ST\_CreateTopoGeo

`ST_CreateTopoGeo` — Fügt eine Sammlung von Geometrien an eine leere Topologie an und gibt eine Bestätigungsmeldung aus.

### Synopsis

```
text ST_CreateTopoGeo(varchar atopology, geometry acollection);
```

## Beschreibung

Fügt eine Sammelgeometrie einer leeren Topologie hinzu und gibt eine Bestätigungsmeldung aus.

Nützlich um eine leere Topologie zu befüllen.

Verfügbarkeit: 2.0



Diese Methode setzt die SQL/MM-Spezifikation um. SQL-MM: Topo-Geo and Topo-Net 3: Routine Details -- X.3.18

## Beispiele

```
-- Populate topology --
SELECT topology.ST_CreateTopoGeo('ri_topo',
  ST_GeomFromText('MULTILINESTRING((384744 236928,384750 236923,384769 236911,384799 ←
    236895,384811 236890,384833 236884,
    384844 236882,384866 236881,384879 236883,384954 236898,385087 236932,385117 236938,
    385167 236938,385203 236941,385224 236946,385233 236950,385241 236956,385254 236971,
    385260 236979,385268 236999,385273 237018,385273 237037,385271 237047,385267 237057,
    385225 237125,385210 237144,385192 237161,385167 237192,385162 237202,385159 237214,
    385159 237227,385162 237241,385166 237256,385196 237324,385209 237345,385234 237375,
    385237 237383,385238 237399,385236 237407,385227 237419,385213 237430,385193 237439,
    385174 237451,385170 237455,385169 237460,385171 237475,385181 237503,385190 237521,
    385200 237533,385206 237538,385213 237541,385221 237542,385235 237540,385242 237541,
    385249 237544,385260 237555,385270 237570,385289 237584,385292 237589,385291 ←
    237596,385284 237630))',3438)
);

      st_createtopogeo
-----
Topology ri_topo populated

-- create tables and topo geometries --
CREATE TABLE ri.roads(gid serial PRIMARY KEY, road_name text);

SELECT topology.AddTopoGeometryColumn('ri_topo', 'ri', 'roads', 'topo', 'LINE');
```

## Siehe auch

[TopoGeo\\_LoadGeometry](#), [AddTopoGeometryColumn](#), [CreateTopology](#), [DropTopology](#)

### 9.5.5 TopoGeo\_AddPoint

**TopoGeo\_AddPoint** — Fügt einen Punkt, unter Berücksichtigung einer Toleranz, an eine bestehende Topologie an. Existierende Kanten werden eventuell aufgetrennt.

## Synopsis

```
integer TopoGeo_AddPoint(varchar atopology, geometry apoint, float8 tolerance);
```

## Beschreibung

Fügt einen Punkt an eine bestehende Topologie an und gibt dessen Identifikator zurück. Der vorgegebene Punkt wird von bestehenden Knoten oder Kanten, innerhalb einer gegebenen Toleranz, gefangen. Eine existierende Kante kann durch den gefangenen Punkt aufgetrennt werden.

Verfügbarkeit: 2.0.0

**Siehe auch**

[TopoGeo\\_AddLineString](#), [TopoGeo\\_AddPolygon](#), [TopoGeo\\_LoadGeometry](#), [AddNode](#), [CreateTopology](#)

### 9.5.6 TopoGeo\_AddLineString

`TopoGeo_AddLineString` — Adds a linestring to an existing topology using a tolerance and possibly splitting existing edges/faces.

**Synopsis**

SETOF integer **TopoGeo\_AddLineString**(varchar atopology, geometry aline, float8 tolerance);

**Beschreibung**

Adds a linestring to an existing topology and returns a set of signed edge identifiers forming it up (negative identifies mean the edge goes in the opposite direction of the input linestring). The given line will snap to existing nodes or edges within given tolerance. Existing edges and faces may be split by the line. New nodes and faces may be added.

**Note**

Die Aktualisierung von Statistiken über Topologien, die über diese Funktion geladen werden, ist Sache des Aufrufers, siehe [maintaining statistics during topology editing and population](#).

---

Verfügbarkeit: 2.0.0

Enhanced: 3.2.0 added support for returning signed identifier.

**Siehe auch**

[TopoGeo\\_AddPoint](#), [TopoGeo\\_AddPolygon](#), [TopoGeo\\_LoadGeometry](#), [AddEdge](#), [CreateTopology](#)

### 9.5.7 TopoGeo\_AddPolygon

`TopoGeo_AddPolygon` — Fügt ein Polygon, unter Berücksichtigung einer Toleranz, an eine bestehende Topologie an. Existierende Kanten/Maschen werden eventuell aufgetrennt. Gibt den Identifikator der Masche zurück.

**Synopsis**

SETOF integer **TopoGeo\_AddPolygon**(varchar atopology, geometry apoly, float8 tolerance);

**Beschreibung**

Fügt ein Polygon zu einer existierenden Topologie hinzu und gibt die Identifikatoren der Maschen aus, aus denen es gebildet ist. Die Begrenzung des gegebenen Polygons wird innerhalb der angegebenen Toleranz an bestehenden Knoten und Kanten gefangen. Gegebenenfalls werden existierende Kanten und Maschen an der Begrenzung des neuen Polygons geteilt.

**Note**

Die Aktualisierung von Statistiken über Topologien, die über diese Funktion geladen werden, ist Sache des Aufrufers, siehe [maintaining statistics during topology editing and population](#).

---

Verfügbarkeit: 2.0.0

---

**Siehe auch**

[TopoGeo\\_AddPoint](#), [TopoGeo\\_AddLineString](#), [TopoGeo\\_LoadGeometry](#), [AddFace](#), [CreateTopology](#)

## 9.5.8 TopoGeo\_LoadGeometry

TopoGeo\_LoadGeometry — Load a geometry into an existing topology, snapping and splitting as needed.

**Synopsis**

```
void TopoGeo_LoadGeometry(vchar atopology, geometry ageom, float8 tolerance);
```

**Beschreibung**

Loads a geometry into an existing topology. The given geometry will snap to existing nodes or edges within given tolerance. Existing edges and faces may be split as a consequence of the load.

**Note**

Die Aktualisierung von Statistiken über Topologien, die über diese Funktion geladen werden, ist Sache des Aufrufers, siehe [maintaining statistics during topology editing and population](#).

Verfügbarkeit: 3.5.0

**Siehe auch**

[TopoGeo\\_AddPoint](#), [TopoGeo\\_AddLineString](#), [TopoGeo\\_AddPolygon](#), [CreateTopology](#)

## 9.6 Topologie Editoren

### 9.6.1 ST\_AddIsoNode

ST\_AddIsoNode — Fügt einen isolierten Knoten zu einer Masche in einer Topologie hinzu und gibt die "nodeid" des neuen Knotens aus. Falls die Masche NULL ist, wird der Knoten dennoch erstellt.

**Synopsis**

```
integer ST_AddIsoNode(vchar 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: 1.1



Diese Methode setzt die SQL/MM-Spezifikation um. SQL-MM: Topo-Net Routines: X+1.3.1

## Beispiele

### Siehe auch

[AddNode](#), [CreateTopology](#), [DropTopology](#), [ST\\_Intersects](#)

## 9.6.2 ST\_AddIsoEdge

`ST_AddIsoEdge` — Fügt eine isolierte Kante, die durch die Geometrie `alinestring` festgelegt wird zu einer Topologie hinzu, indem zwei bestehende isolierte Knoten `anode` und `anothernode` verbunden werden. Gibt die "edgeid" der neuen Kante aus.

### Synopsis

```
integer ST_AddIsoEdge(varchar atopology, integer anode, integer anothernode, geometry alinestring);
```

### Beschreibung

Fügt eine isolierte Kante, die durch die Geometrie `alinestring` festgelegt wird zu einer Topologie hinzu, indem zwei bestehende isolierte Knoten `anode` und `anothernode` verbunden werden. Gibt die "edgeid" der neuen Kante aus.

Wenn das Koordinatenreferenzsystem (SRID) der Geometrie `alinestring` nicht mit dem der Topologie übereinstimmt, irgendein Eingabewert NULL ist, die Knoten in mehreren Maschen enthalten sind, oder die Knoten Anfangs- oder Endknoten einer bestehenden Kante darstellen, wird eine Fehlermeldung ausgegeben.

Wenn `alinestring` nicht innerhalb der Masche liegt zu der `anode` und `anothernode` gehören, dann wird eine Fehlermeldung ausgegeben.

Wenn `anode` und `anothernode` nicht Anfangs- und Endpunkt von `alinestring` sind, wird eine Fehlermeldung ausgegeben.

Verfügbarkeit: 1.1



Diese Methode setzt die SQL/MM-Spezifikation um. SQL-MM: Topo-Geo and Topo-Net 3: Routine Details: X.3.4

## Beispiele

### Siehe auch

[ST\\_AddIsoNode](#), [ST\\_IsSimple](#), [ST\\_Within](#)

## 9.6.3 ST\_AddEdgeNewFaces

`ST_AddEdgeNewFaces` — Fügt eine Kante hinzu. Falls dabei eine Masche aufgetrennt wird, so wird die ursprüngliche Masche gelöscht und durch zwei neue Maschen ersetzt.

### Synopsis

```
integer ST_AddEdgeNewFaces(varchar atopology, integer anode, integer anothernode, geometry acurve);
```



## Beschreibung

Fügt eine Kante hinzu. Falls dabei eine Masche aufgetrennt wird, so wird die ursprüngliche Masche gelöscht und durch zwei neue Maschen ersetzt. Gibt die ID der hinzugefügten Kante aus.

Führt ein entsprechendes Update auf alle verbundenen Kanten und Beziehungen durch.

Wenn irgendwelche Übergabewerte NULL sind, die angegebenen Knoten unbekannt sind (müssen bereits in der `node` Tabelle des Schemas "topology" existieren), `acurve` kein `LINestring` ist, oder `anode` und `anothernode` nicht die Anfangs- und Endpunkte von `acurve` sind, dann wird eine Fehlermeldung ausgegeben.

Wenn das Koordinatenreferenzsystem (SRID) der Geometrie `acurve` nicht mit jener der Topologie übereinstimmt, wird eine Fehlermeldung ausgegeben.

Verfügbarkeit: 2.0



Diese Methode setzt die SQL/MM-Spezifikation um. SQL-MM: Topo-Geo and Topo-Net 3: Routine Details: X.3.12

## Beispiele

### Siehe auch

[ST\\_RemEdgeNewFace](#)

[ST\\_AddEdgeModFace](#)

## 9.6.4 ST\_AddEdgeModFace

`ST_AddEdgeModFace` — Fügt eine Kante hinzu. Falls dabei eine Masche aufgetrennt wird, so wird die ursprüngliche Masche angepasst und eine weitere Masche hinzugefügt.

### Synopsis

integer `ST_AddEdgeModFace`(varchar atopology, integer anode, integer anothernode, geometry acurve);

## Beschreibung

Fügt eine Kante hinzu. Falls dabei eine Masche aufgetrennt wird, so wird die ursprüngliche Masche angepasst und eine weitere Masche hinzugefügt.



### Note

Wenn möglich, wird die neue Masche auf der linken Seite der neuen Kante erstellt. Dies ist jedoch nicht möglich, wenn die Masche auf der linken Seite die (unbegrenzte) Grundmenge der Maschen darstellt.

Gibt die id der hinzugefügten Kante zurück.

Führt ein entsprechendes Update auf alle verbundenen Kanten und Beziehungen durch.

Wenn irgendwelche Übergabewerte NULL sind, die angegebenen Knoten unbekannt sind (müssen bereits in der `node` Tabelle des Schemas "topology" existieren), `acurve` kein `LINestring` ist, oder `anode` und `anothernode` nicht die Anfangs- und Endpunkte von `acurve` sind, dann wird eine Fehlermeldung ausgegeben.

Wenn das Koordinatenreferenzsystem (SRID) der Geometrie `acurve` nicht mit jener der Topologie übereinstimmt, wird eine Fehlermeldung ausgegeben.

Verfügbarkeit: 2.0



Diese Methode setzt die SQL/MM-Spezifikation um. SQL-MM: Topo-Geo and Topo-Net 3: Routine Details: X.3.13

## Beispiele

### Siehe auch

[ST\\_RemEdgeModFace](#)

[ST\\_AddEdgeNewFaces](#)

## 9.6.5 ST\_RemEdgeNewFace

`ST_RemEdgeNewFace` — Entfernt eine Kante. Falls die gelöschte Kante zwei Maschen voneinander getrennt hat, werden die ursprünglichen Maschen gelöscht und durch einer neuen Masche ersetzt.

### Synopsis

```
integer ST_RemEdgeNewFace(varchar atopology, integer anedge);
```

### Beschreibung

Entfernt eine Kante. Falls die gelöschte Kante zwei Maschen voneinander getrennt hat, werden die ursprünglichen Maschen gelöscht und durch einer neuen Masche ersetzt.

Gibt die ID der neu erzeugten Masche aus; oder NULL wenn keine Masche erstellt wurde. Es wird keine neue Masche erstellt, wenn die gelöschte Kante defekt oder isoliert ist, oder wenn sie die Begrenzung der Maschengrundmenge darstellt (wodurch die Grundmenge womöglich von der anderen Seite in die Masche fließen könnte).

Führt ein entsprechendes Update auf alle verbundenen Kanten und Beziehungen durch.

Wenn eine Kante an der Definition einer bestehenden TopoGeometry beteiligt ist, kann sie nicht gelöscht werden. Wenn irgendeine TopoGeometry nur durch eine der beiden Maschen definiert ist (und nicht auch durch die andere), können die beiden Maschen nicht "geheilt" werden.

Wenn irgendwelche Übergabewerte NULL sind, die angegebene Kante unbekannt ist (muss bereits in der `edge` Tabelle des Schemas "topology" existieren), oder die Bezeichnung der Topologie ungültig ist, dann wird eine Fehlermeldung ausgegeben.

Verfügbarkeit: 2.0



Diese Methode setzt die SQL/MM-Spezifikation um. SQL-MM: Topo-Geo and Topo-Net 3: Routine Details: X.3.14

## Beispiele

### Siehe auch

[ST\\_RemEdgeModFace](#)

[ST\\_AddEdgeNewFaces](#)

## 9.6.6 ST\_RemEdgeModFace

`ST_RemEdgeModFace` — Entfernt eine Kante, und wenn die Kante zwei Flächen trennt, wird eine Fläche gelöscht und die andere Fläche so verändert, dass sie den Raum beider Flächen abdeckt.

### Synopsis

```
integer ST_RemEdgeModFace(varchar atopology, integer anedge);
```

## Beschreibung

Entfernt eine Kante, und wenn die entfernte Kante zwei Flächen trennt, wird eine Fläche gelöscht und die andere Fläche so verändert, dass sie den Raum beider Flächen abdeckt. Bevorzugt wird die Fläche auf der rechten Seite beibehalten, um mit `ST_AddEdgeModFace` konsistent zu sein. Gibt die ID der Fläche zurück, die erhalten bleibt.

Führt ein entsprechendes Update auf alle verbundenen Kanten und Beziehungen durch.

Wenn eine Kante an der Definition einer bestehenden TopoGeometry beteiligt ist, kann sie nicht gelöscht werden. Wenn irgendeine TopoGeometry nur durch eine der beiden Maschen definiert ist (und nicht auch durch die andere), können die beiden Maschen nicht "geheilt" werden.

Wenn irgendwelche Übergabewerte NULL sind, die angegebene Kante unbekannt ist (muss bereits in der `edge` Tabelle des Schemas "topology" existieren), oder die Bezeichnung der Topologie ungültig ist, dann wird eine Fehlermeldung ausgegeben.

Verfügbarkeit: 2.0



Diese Methode setzt die SQL/MM-Spezifikation um. SQL-MM: Topo-Geo and Topo-Net 3: Routine Details: X.3.15

## Beispiele

### Siehe auch

[ST\\_AddEdgeModFace](#)

[ST\\_RemEdgeNewFace](#)

## 9.6.7 ST\_ChangeEdgeGeom

`ST_ChangeEdgeGeom` — Ändert die geometrische Form einer Kante, ohne sich auf die topologische Struktur auszuwirken.

### Synopsis

```
text ST_ChangeEdgeGeom(varchar atopology, integer anedge, geometry acurve);
```

### Beschreibung

Ändert die geometrische Form der Kante, ohne sich auf topologische Struktur auszuwirken.

Wenn eines der Argumente null ist, die angegebene Kante nicht in der Tabelle `edge` des Topologie-Schemas existiert, die `acurve` keine `LINestring` ist oder die Änderung die zugrunde liegende Topologie verändern würde, wird ein Fehler ausgelöst.

Wenn das Koordinatenreferenzsystem (SRID) der Geometrie `acurve` nicht mit jener der Topologie übereinstimmt, wird eine Fehlermeldung ausgegeben.

Wenn die neue `acurve` nicht "simple" ist, wird eine Fehlermeldung ausgegeben.

Wenn beim Verschieben der Kante von der alten auf die neue Position ein Hindernis auftritt, wird eine Fehlermeldung ausgegeben.

Verfügbarkeit: 1.1.0

Erweiterung: 2.0.0 Erzwingung topologischer Konsistenz hinzugefügt



Diese Methode setzt die SQL/MM-Spezifikation um. SQL-MM: Topo-Geo and Topo-Net 3: Routine Details X.3.6

## Beispiele

```
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](#)

## 9.6.8 ST\_ModEdgeSplit

**ST\_ModEdgeSplit** — Trennt eine Kante auf, indem ein neuer Knoten entlang einer bestehenden Kante erstellt wird. Ändert die ursprüngliche Kante und fügt eine neue Kante hinzu.

### Synopsis

integer **ST\_ModEdgeSplit**(varchar atopology, integer anedge, geometry apoint);

### Beschreibung

Trennt eine Kante auf, indem ein neuer Knoten entlang einer bestehenden Kante erstellt wird. Ändert die ursprüngliche Kante und fügt eine neue Kante hinzu. Alle bestehenden Kanten und Beziehungen werden entsprechend aktualisiert. Gibt den Identifikator des neu hinzugefügten Knotens aus.

Verfügbarkeit: 1.1

Änderung: 2.0 - In Vorgängerversionen fälschlicherweise als `ST_ModEdgesSplit` bezeichnet



Diese Methode setzt die SQL/MM-Spezifikation um. SQL-MM: Topo-Geo and Topo-Net 3: Routine Details: X.3.9

## Beispiele

```
-- Add an edge --
SELECT topology.AddEdge('ma_topo', ST_GeomFromText('LINESTRING(227592 893910, 227600
    893910)', 26986) ) As edgeid;

-- edgeid-
3

-- Split the edge --
SELECT topology.ST_ModEdgeSplit('ma_topo', 3, ST_SetSRID(ST_Point(227594,893910),26986) ) ←
    As node_id;
    node_id
-----
7
```

**Siehe auch**

[ST\\_NewEdgesSplit](#), [ST\\_ModEdgeHeal](#), [ST\\_NewEdgeHeal](#), [AddEdge](#)

**9.6.9 ST\_ModEdgeHeal**

`ST_ModEdgeHeal` — "Heilt" zwei Kanten, indem der verbindende Knoten gelöscht wird, die erste Kante modifiziert und die zweite Kante gelöscht wird. Gibt die ID des gelöschten Knoten zurück.

**Synopsis**

```
int ST_ModEdgeHeal(varchar atopology, integer anedge, integer anotheredge);
```

**Beschreibung**

"Heilt" zwei Kanten, indem der verbindende Knoten gelöscht wird, die erste Kante modifiziert und die zweite Kante gelöscht wird. Gibt die ID des gelöschten Knoten zurück. Aktualisiert alle verbundenen Kanten und Beziehungen dementsprechend.

Verfügbarkeit: 2.0



Diese Methode setzt die SQL/MM-Spezifikation um. SQL-MM: Topo-Geo and Topo-Net 3: Routine Details: X.3.9

**Siehe auch**

[ST\\_ModEdgeSplit](#) [ST\\_NewEdgesSplit](#)

**9.6.10 ST\_NewEdgeHeal**

`ST_NewEdgeHeal` — "Heilt" zwei Kanten, indem der verbindende Knoten und beide Kanten gelöscht werden. Die beiden Kanten werden durch eine Kante ersetzt, welche dieselbe 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



Diese Methode setzt die SQL/MM-Spezifikation um. SQL-MM: Topo-Geo and Topo-Net 3: Routine Details: X.3.9

**Siehe auch**

[ST\\_ModEdgeHeal](#) [ST\\_ModEdgeSplit](#) [ST\\_NewEdgesSplit](#)

---

### 9.6.11 ST\_MoveIsoNode

`ST_MoveIsoNode` — Verschiebt einen isolierten Knoten in einer Topologie von einer Stelle an eine andere. Falls die neue Geometrie `apoint` bereits als Knoten existiert, wird eine Fehlermeldung ausgegeben. Gibt eine Beschreibung der Verschiebung aus.

#### Synopsis

```
text ST_MoveIsoNode(varchar atopology, integer anode, geometry apoint);
```

#### Beschreibung

Verschiebt einen isolierten Knoten in einer Topologie von einer Stelle an eine andere. Falls die neue Geometrie `apoint` bereits als Knoten existiert, wird eine Fehlermeldung ausgegeben.

Wenn eines der Argumente null ist, der `apoint` kein Punkt ist, der vorhandene Knoten nicht isoliert ist (ein Start- oder Endpunkt einer vorhandenen Kante ist), die neue Knotenposition eine vorhandene Kante schneidet (sogar an den Endpunkten) oder die neue Position in einer anderen Fläche liegt (seit 3.2.0), wird eine Ausnahme ausgelöst.

Wenn das Koordinatenreferenzsystem (SRID) der Punktgeometrie nicht mit jener der Topologie übereinstimmt, wird eine Fehlermeldung ausgegeben.

Verfügbarkeit: 2.0.0

Verbessert: 3.2.0 stellt sicher, dass der Knoten nicht in ein anderes Gesicht verschoben werden kann



Diese Methode setzt die SQL/MM-Spezifikation um. SQL-MM: Topo-Net Routines: X.3.2

#### Beispiele

```
-- Add an isolated node with no face --
SELECT topology.ST_AddIsoNode('ma_topo',  NULL, ST_GeomFromText('POINT(227579 893916)',  ←
    26986) ) As nodeid;
  nodeid
-----
      7
-- Move the new node --
SELECT topology.ST_MoveIsoNode('ma_topo', 7,  ST_GeomFromText('POINT(227579.5 893916.5)',  ←
    26986) ) As descrip;
                descrip
-----
Isolated Node 7 moved to location 227579.5,893916.5
```

#### Siehe auch

[ST\\_AddIsoNode](#)

### 9.6.12 ST\_NewEdgesSplit

`ST_NewEdgesSplit` — Trennt eine Kante auf, indem ein neuer Knoten entlang einer bestehenden Kante erstellt, die ursprüngliche Kante gelöscht und durch zwei neue Kanten ersetzt wird. Gibt die ID des neu erstellten Knotens aus, der die neuen Kanten verbindet.

#### Synopsis

```
integer ST_NewEdgesSplit(varchar atopology, integer anedge, geometry apoint);
```

## Beschreibung

Trennt eine Kante mit der Kanten-ID `anedgeauf`, indem ein neuer Knoten mit der Punktlage `apoint` entlang der aktuellen Kante erstellt, die ursprüngliche Kante gelöscht und durch zwei neue Kanten ersetzt wird. Gibt die ID des neu erstellten Knotens aus, der die neuen Kanten verbindet. Aktualisiert alle verbundenen Kanten und Beziehungen dementsprechend.

Wenn das Koordinatenreferenzsystem (SRID) der Punktgeometrie nicht mit dem der Topologie übereinstimmt, `apoint` keine Punktgeometrie ist, der Punkt NULL ist, der Punkt bereits als Knoten existiert, die Kante mit einer bestehenden Kante nicht zusammenpasst, oder der Punkt nicht innerhalb der Kante liegt, dann wird eine Fehlermeldung ausgegeben.

Verfügbarkeit: 1.1



Diese Methode setzt die SQL/MM-Spezifikation um. SQL-MM: Topo-Net Routines: X.3.8

## Beispiele

```
-- Add an edge --
SELECT topology.AddEdge('ma_topo', ST_GeomFromText('LINESTRING(227575 893917,227592 893900) ←
', 26986) ) As edgeid;
-- result-
edgeid
-----
      2
-- Split the new edge --
SELECT topology.ST_NewEdgesSplit('ma_topo', 2, ST_GeomFromText('POINT(227578.5 893913.5)', ←
26986) ) As newnodeid;
newnodeid
-----
      6
```

## Siehe auch

[ST\\_ModEdgeSplit](#) [ST\\_ModEdgeHeal](#) [ST\\_NewEdgeHeal](#) [AddEdge](#)

### 9.6.13 ST\_RemoveIsoNode

`ST_RemoveIsoNode` — Löscht einen isolierten Knoten und gibt eine Beschreibung der getroffenen Maßnahmen aus. Falls der Knoten nicht isoliert ist (ist der Anfangs- oder der Endpunkt einer Kante), wird eine Fehlermeldung ausgegeben.

## Synopsis

text `ST_RemoveIsoNode`(varchar `atopology`, integer `anode`);

## Beschreibung

Löscht einen isolierten Knoten und gibt eine Beschreibung der getroffenen Maßnahmen aus. Falls der Knoten nicht isoliert ist (ist der Anfangs- oder der Endpunkt einer Kante), wird eine Fehlermeldung ausgegeben.

Verfügbarkeit: 1.1



Diese Methode setzt die SQL/MM-Spezifikation um. SQL-MM: Topo-Geo and Topo-Net 3: Routine Details: X+1.3.3

## Beispiele

```
-- Remove an isolated node with no face --
SELECT topology.ST_RemoveIsoNode('ma_topo', 7 ) As result;
           result
-----
Isolated node 7 removed
```

## Siehe auch

[ST\\_AddIsoNode](#)

### 9.6.14 ST\_RemoveIsoEdge

`ST_RemoveIsoEdge` — Löscht einen isolierten Knoten und gibt eine Beschreibung der getroffenen Maßnahmen aus. Falls der Knoten nicht isoliert ist, wird eine Fehlermeldung ausgegeben.

## Synopsis

text `ST_RemoveIsoEdge`(varchar atopology, integer anedge);

## Beschreibung

Löscht einen isolierten Knoten und gibt eine Beschreibung der getroffenen Maßnahmen aus. Falls der Knoten nicht isoliert ist, wird eine Fehlermeldung ausgegeben.

Verfügbarkeit: 1.1



Diese Methode setzt die SQL/MM-Spezifikation um. SQL-MM: Topo-Geo and Topo-Net 3: Routine Details: X+1.3.3

## Beispiele

```
-- Remove an isolated node with no face --
SELECT topology.ST_RemoveIsoNode('ma_topo', 7 ) As result;
           result
-----
Isolated node 7 removed
```

## Siehe auch

[ST\\_AddIsoNode](#)

## 9.7 Zugriffsfunktionen zur Topologie

### 9.7.1 GetEdgeByPoint

`GetEdgeByPoint` — Findet die edge-id einer Kante die einen gegebenen Punkt schneidet.

## Synopsis

integer `GetEdgeByPoint`(varchar atopology, geometry apoint, float8 toll);



## Beschreibung

Erfasst die ID einer Kante, die einen Punkt schneidet

Die Funktion gibt eine Ganzzahl (id-edge) für eine Topologie, einen POINT und eine Toleranz aus. Wenn tolerance = 0, dann muss der Punkt die Kante schneiden.

Wenn apoint keine Kante schneidet, wird 0 (Null) zurückgegeben.

Wenn use tolerance > 0 und es gibt mehr als eine Kante in der Nähe des Punktes dann eine Ausnahme ausgelöst wird.



### Note

Wenn tolerance = 0, wird von der Funktion ST\_Intersects angewendet, ansonsten ST\_DWithin.

Wird vom GEOS Modul ausgeführt

Verfügbarkeit: 2.0.0

## Beispiele

Die folgenden Beispiele benutzen die Kanten, die wir in [AddEdge](#) erzeugt haben

```
SELECT topology.GetEdgeByPoint('ma_topo',geom, 1) As 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;

-- get error --
ERROR:  Two or more edges found
```

## Siehe auch

[AddEdge](#), [GetNodeByPoint](#), [GetFaceByPoint](#)

### 9.7.2 GetFaceByPoint

GetFaceByPoint — Findet eine Fläche, die einen bestimmten Punkt schneidet.

#### Synopsis

integer **GetFaceByPoint**(varchar atopology, geometry apoint, float8 toll1);

## Beschreibung

Findet eine Fläche, die von einem Punkt referenziert wird, mit der angegebenen Toleranz.

Die Funktion sucht tatsächlich nach einer Fläche, die einen Kreis mit dem Punkt als Mittelpunkt und der Toleranz als Radius schneidet.

Wenn keine Fläche die angegebene Abfrageposition schneidet, wird 0 zurückgegeben (Universalfläche).

Wenn mehr als eine Fläche den Abfrageort schneidet, wird eine Ausnahme ausgelöst.

Verfügbarkeit: 2.0.0

Verbessert: 3.2.0 effizientere Implementierung und klarerer Vertrag, funktioniert nicht mehr mit ungültigen Topologien.

## Beispiele

```
SELECT topology.GetFaceByPoint('ma_topo',geom, 10) As with1mtol, topology.GetFaceByPoint(' ←
ma_topo',geom,0) As withnotol
FROM ST_GeomFromEWKT('POINT(234604.6 899382.0)') As geom;
```

```
with1mtol | withnotol
-----+-----
1 | 0
```

```
SELECT topology.GetFaceByPoint('ma_topo',geom, 1) As nearnode
FROM ST_GeomFromEWKT('POINT(227591.9 893900.4)') As geom;
```

```
-- get error --
ERROR: Two or more faces found
```

## Siehe auch

[GetFaceContainingPoint](#), [AddFace](#), [GetNodeByPoint](#), [GetEdgeByPoint](#)

### 9.7.3 GetFaceContainingPoint

`GetFaceContainingPoint` — Findet die Fläche, die einen Punkt enthält.

#### Synopsis

integer **GetFaceContainingPoint**(text atopolgy, geometry apoint);

#### Beschreibung

Gibt die ID der Fläche zurück, die einen Punkt enthält.

Eine Ausnahme wird ausgelöst, wenn der Punkt auf eine Flächenbegrenzung fällt.



#### Note

Die Funktion setzt eine gültige Topologie voraus, die Kantenverknüpfungen und Flächenbeschriftungen verwendet.

Verfügbarkeit: 3.2.0

**Siehe auch**[ST\\_GetFaceGeometry](#)**9.7.4 GetNodeByPoint**

GetNodeByPoint — Findet zu der Lage eines Punktes die node-id eines Knotens.

**Synopsis**

integer **GetNodeByPoint**(varchar atopology, geometry apoint, float8 toll);

**Beschreibung**

Erfasst zu der Lage eines Punktes die ID eines Knotens.

Diese Funktion gibt für eine Topologie, einen POINT und eine Toleranz eine Ganzzahl (id-node) aus. Tolerance = 0 bedeutet exakte Überschneidung, ansonsten wird der Knoten in einem bestimmten Abstand gesucht.

Wenn apoint keinen Knoten schneidet, wird 0 (Null) zurückgegeben.

Wenn use tolerance > 0 und es gibt mehr als einen Knoten in der Nähe des Punktes dann eine Ausnahme geworfen wird.

**Note**

Wenn tolerance = 0, wird von der Funktion ST\_Intersects angewendet, ansonsten ST\_DWithin.

Wird vom GEOS Modul ausgeführt

Verfügbarkeit: 2.0.0

**Beispiele**

Die folgenden Beispiele benutzen die Kanten, die wir in [AddEdge](#) erzeugt haben

```
SELECT topology.GetNodeByPoint('ma_topo',geom, 1) As nearnode
FROM ST_GeomFromEWKT('SRID=26986;POINT(227591.9 893900.4)') As geom;
nearnode
-----
2
```

```
SELECT topology.GetNodeByPoint('ma_topo',geom, 1000) As too_much_tolerance
FROM ST_GeomFromEWKT('SRID=26986;POINT(227591.9 893900.4)') As geom;

----get error--
ERROR: Two or more nodes found
```

**Siehe auch**[AddEdge](#), [GetEdgeByPoint](#), [GetFaceByPoint](#)**9.7.5 GetTopologyID**

GetTopologyID — Gibt für den Namen einer Topologie die ID der Topologie in der Tabelle "topology.topology" aus.

## Synopsis

integer **GetTopologyID**(varchar toponame);

## Beschreibung

Gibt für den Namen einer Topologie, die ID der Topologie in der Tabelle "topology.topology" aus.

Verfügbarkeit: 1.1

## Beispiele

```
SELECT topology.GetTopologyID('ma_topo') As topo_id;
topo_id
-----
      1
```

## Siehe auch

[CreateTopology](#), [DropTopology](#), [GetTopologyName](#), [GetTopologySRID](#)

## 9.7.6 GetTopologySRID

**GetTopologySRID** — Gibt für den Namen einer Topologie, die SRID der Topologie in der Tabelle "topology.topology" aus.

## Synopsis

integer **GetTopologyID**(varchar toponame);

## Beschreibung

Gibt für den Namen einer Topologie, die ID des Koordinatenreferenzsystems in der Tabelle "topology.topology" aus.

Verfügbarkeit: 2.0.0

## Beispiele

```
SELECT topology.GetTopologySRID('ma_topo') As SRID;
SRID
-----
 4326
```

## Siehe auch

[CreateTopology](#), [DropTopology](#), [GetTopologyName](#), [GetTopologyID](#)

## 9.7.7 GetTopologyName

**GetTopologyName** — Gibt für die ID der Topologie, den Namen der Topologie (Schema) zurück.

**Synopsis**

```
varchar GetTopologyName(integer topology_id);
```

**Beschreibung**

Gibt für die ID einer Topologie, den Namen (Schema) der Topologie in der Tabelle "topology.topology" aus.

Verfügbarkeit: 1.1

**Beispiele**

```
SELECT topology.GetTopologyName(1) As topo_name;
topo_name
-----
ma_topo
```

**Siehe auch**

[CreateTopology](#), [DropTopology](#), [GetTopologyID](#), [GetTopologySRID](#)

**9.7.8 ST\_GetFaceEdges**

ST\_GetFaceEdges — Gibt die Kanten, die *aface* begrenzen, sortiert aus.

**Synopsis**

```
getfaceedges_returntype ST_GetFaceEdges(varchar atopology, integer aface);
```

**Beschreibung**

Gibt die Kanten, die *aface* begrenzen, sortiert aus. Jede Ausgabe besteht aus einer Sequenz und einer "edgeid". Die 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



Diese Methode setzt die SQL/MM-Spezifikation um. SQL-MM 3 Topo-Geo and Topo-Net 3: Routine Details: X.3.5

**Beispiele**

```
-- Returns the edges bounding face 1
SELECT (topology.ST_GetFaceEdges('tt', 1)).*;
-- result --
sequence | edge
-----+-----
         1 |   -4
         2 |    5
         3 |    7
         4 |   -6
         5 |    1
         6 |    2
         7 |    3
(7 rows)
```

```
-- Returns the sequence, edge id
-- and geometry of the edges that bound face 1
-- If you just need geom and seq, can use ST_GetFaceGeometry
SELECT t.seq, t.edge, geom
FROM topology.ST_GetFaceEdges('tt',1) As t(seq,edge)
      INNER JOIN tt.edge AS e ON abs(t.edge) = e.edge_id;
```

**Siehe auch**

[GetRingEdges](#), [AddFace](#), [ST\\_GetFaceGeometry](#)

**9.7.9 ST\_GetFaceGeometry**

`ST_GetFaceGeometry` — Gibt für eine Topologie und eine bestimmte Maschen-ID das Polygon zurück.

**Synopsis**

geometry `ST_GetFaceGeometry`(varchar atopology, integer aface);

**Beschreibung**

Gibt für eine Topologie und eine bestimmte Maschen-ID das Polygon zurück. Erstellt das Polygon aus den Kanten, die die Masche aufbauen.

Verfügbarkeit: 1.1



Diese Methode setzt die SQL/MM-Spezifikation um. SQL-MM 3 Topo-Geo and Topo-Net 3: Routine Details: X.3.16

**Beispiele**

```
-- Returns the wkt of the polygon added with AddFace
SELECT ST_AsText(topology.ST_GetFaceGeometry('ma_topo', 1)) As facegeomwkt;
-- result --
          facegeomwkt
-----
POLYGON((234776.9 899563.7,234896.5 899456.7,234914 899436.4,234946.6 899356.9,
234872.5 899328.7,234891 899285.4,234992.5 899145,234890.6 899069,
234755.2 899255.4,234612.7 899379.4,234776.9 899563.7))
```

**Siehe auch**

[AddFace](#)

**9.7.10 GetRingEdges**

`GetRingEdges` — Gibt eine sortierte Liste von mit Vorzeichen versehenen Identifikatoren der Kanten zurück, die angetroffen werden, wenn man an der Seite der gegebenen Kante entlangwandert.

**Synopsis**

getfaceedges\_returntype `GetRingEdges`(varchar atopology, integer aring, integer max\_edges=null);

## Beschreibung

Gibt eine sortierte Liste von mit Vorzeichen versehenen Identifikatoren der Kanten zurück, die angetroffen werden, wenn man an der Seite der gegebenen Kante entlangwandert. Jede Ausgabe besteht aus einer Sequenz und einer mit einem Vorzeichen versehenen ID der Kante. Die Sequenz beginnt mit dem Wert 1.

Wenn Sie eine positive ID für die Kante übergeben, beginnt der Weg auf der linken Seite der entsprechenden Kante und folgt der Ausrichtung der Kante. Wenn Sie eine negative ID für die Kante übergeben, beginnt der Weg auf der rechten Seite der Kante und verläuft rückwärts.

Wenn `max_edges` nicht NULL ist, so beschränkt dieser Parameter die Anzahl der von dieser Funktion ausgegebenen Datensätze. Ist als Sicherheitsparameter für den Umgang mit möglicherweise invaliden Topologien gedacht.



### Note

Diese Funktion verwendet Metadaten um die Kanten eines Ringes zu verbinden.

---

Verfügbarkeit: 2.0.0

## Siehe auch

[ST\\_GetFaceEdges](#), [GetNodeEdges](#)

### 9.7.11 GetNodeEdges

`GetNodeEdges` — Gibt für einen Knoten die sortierte Menge der einfallenden Kanten aus.

## Synopsis

```
getfaceedges_returntype GetNodeEdges(varchar atopology, integer anode);
```

## Beschreibung

Gibt für einen Knoten die sortierte Menge der einfallenden Kanten aus. Jede Ausgabe besteht aus einer Sequenz und einer mit Vorzeichen versehenen ID für die Kante. Die Sequenz beginnt mit dem Wert 1. Eine Kante mit positiver ID beginnt an dem gegebenen Knoten. Eine negative Kante endet in dem gegebenen Knoten. Geschlossene Kanten kommen zweimal vor (mit beiden Vorzeichen). Die Sortierung geschieht von Norden ausgehend im Uhrzeigersinn.



### Note

Diese Funktion errechnet die Reihenfolge, anstatt sie aus den Metadaten abzuleiten und kann daher verwendet werden, um die Kanten eines Ringes zu verbinden.

---

Verfügbarkeit: 2.0

## Siehe auch

[getfaceedges\\_returntype](#), [GetRingEdges](#), [ST\\_Azimuth](#)

---

## 9.8 Topologie Verarbeitung

### 9.8.1 Polygonize

Polygonize — Findet und registriert alle Maschen, die durch die Kanten der Topologie festgelegt sind.

#### Synopsis

```
text Polygonize(varchar toponame);
```

#### Beschreibung

Registriert alle Maschen, die aus den Kanten der topologischen Elementarstrukturen erstellt werden können

Von der Zieltopologie wird angenommen, dass sie keine sich selbst überschneidenden Kanten enthält.



#### Note

Da bereits bekannte Maschen erkannt werden, kann Polygonize gefahrlos mehrere Male auf die selbe Topologie angewendet werden.



#### Note

Von dieser Funktion werden die Attribute "next\_left\_edge" und "next\_right\_edge" der Tabelle "edge" weder verwendet noch gesetzt.

Verfügbarkeit: 2.0.0

#### Siehe auch

[AddFace](#), [ST\\_Polygonize](#)

### 9.8.2 AddNode

AddNode — Fügt einen Knotenpunkt zu der Tabelle "node" in dem vorgegebenen topologischen Schema hinzu und gibt die "nodeid" des neuen Knotens aus. Falls der Punkt bereits als Knoten existiert, wird die vorhandene nodeid 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](#)

**9.8.3 AddEdge**

**AddEdge** — Fügt die Kante eines Linienzugs in der Tabelle "edge", und die zugehörigen Anfangs- und Endpunkte in die Knotenpunktabelle, des jeweiligen topologischen Schemas ein. Dabei wird die übergebene Linienzuggeometrie verwendet und die `edgeid` der neuen (oder bestehenden) Kante ausgegeben.

**Synopsis**

```
integer AddEdge(varchar toponame, geometry aline);
```

**Beschreibung**

Fügt eine Kante in der Tabelle "edge", und die zugehörigen Knoten in die Tabelle "node", des jeweiligen Schemas `toponame` ein. Dabei wird die übergebene Linienzuggeometrie verwendet und die `edgeid` des neuen oder des bestehenden Datensatzes ausgegeben. Die neu hinzugefügte Kante hat auf beiden Seiten die Masche für die Grundmenge/"Universum" und verweist auf sich selbst.

**Note**

Wenn die Geometrie `aline` eine bestehende Kante kreuzt, überlagert, beinhaltet oder in ihr enthalten ist, dann wird eine Fehlermeldung ausgegeben und die Kante wird nicht hinzugefügt.

**Note**

Die Geometrie von `aline` muss dieselbe SRID aufweisen wie die Topologie, ansonsten wird die Fehlermeldung "invalid spatial reference sys error" ausgegeben.

Wird vom GEOS Modul ausgeführt

**Warning**

`AddEdge` is deprecated as of 3.5.0. Use `TopoGeo_AddLineString` instead.

Verfügbarkeit: 2.0.0

**Beispiele**

```
SELECT topology.AddEdge('ma_topo', ST_GeomFromText('LINESTRING(227575.8 893917.2,227591.9 ↔
      893900.4)', 26986) ) As edgeid;
-- result-
edgeid
-----
1

SELECT topology.AddEdge('ma_topo', ST_GeomFromText('LINESTRING(227591.9 893900.4,227622.6 ↔
      893844.2,227641.6 893816.5,
      227704.5 893778.5)', 26986) ) As edgeid;
-- result --
edgeid
-----
2

SELECT topology.AddEdge('ma_topo', ST_GeomFromText('LINESTRING(227591.2 893900, 227591.9 ↔
      893900.4,
      227704.5 893778.5)', 26986) ) As edgeid;
-- gives error --
ERROR:  Edge intersects (not on endpoints) with existing edge 1
```

**Siehe auch**

[TopoGeo\\_AddLineString](#), [CreateTopology](#), [Section 4.5](#)

**9.8.4 AddFace**

`AddFace` — Registriert die Elementarstruktur einer Masche in einer Topologie und gibt den Identifikator der Masche aus.

**Synopsis**

```
integer AddFace(varchar toponame, geometry apolygon, boolean force_new=false);
```

**Beschreibung**

Registriert die Elementarstruktur einer Masche in einer Topologie und gibt den Identifikator der Masche aus.

Bei einer neu hinzugefügten Masche werden die Kanten die ihre Begrenzung bilden und jene die innerhalb der Masche liegen aktualisiert, damit diese die richtigen Werte in den Attributen "left\_face" und "right\_face" aufweisen. Isolierte Knoten innerhalb der Masche werden ebenfalls aktualisiert, damit das Attribut "containing\_face" die richtigen Werte aufweist.

**Note**

Von dieser Funktion werden die Attribute "next\_left\_edge" und "next\_right\_edge" der Tabelle "edge" weder verwendet noch gesetzt.

Es wird angenommen, dass die Zieltopologie valide ist (keine sich selbst überschneidenden Kanten enthält). Eine Fehlermeldung wird ausgegeben, wenn: Die Begrenzung des Polygons nicht vollständig durch bestehende Kanten festgelegt ist oder das Polygon eine bereits bestehende Masche überlappt.

Falls die Geometrie `apolygon` bereits als Masche existiert, dann: wenn `force_new` `FALSE` (der Standardwert) ist, dann wird die bestehende Masche zurückgegeben; wenn `force_new` `TRUE` ist, dann wird der neu registrierten Masche eine neue ID zugewiesen.

**Note**

Wenn eine bestehende Masche neu registriert wird (`force_new=true`), werden keine Maßnahmen durchgeführt um hängende Verweise auf eine bestehende Masche in den Tabellen "edge", "node" und "relation" zu bereinigen, noch wird der das Attribut MBR der bestehenden Masche 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**

```
-- first add the edges we use generate_series as an iterator (the below
-- will only work for polygons with < 10000 points because of our max in gs)
SELECT topology.AddEdge('ma_topo', ST_MakeLine(ST_PointN(geom,i), ST_PointN(geom, i + 1) )) ←
  As edgeid
  FROM (SELECT ST_NPoints(geom) AS npt, geom
        FROM
          (SELECT ST_Boundary(ST_GeomFromText('POLYGON((234896.5 899456.7,234914  ←
            899436.4,234946.6 899356.9,234872.5 899328.7,
            234891 899285.4,234992.5 899145, 234890.6 899069,234755.2 899255.4,
            234612.7 899379.4,234776.9 899563.7,234896.5 899456.7))', 26986) ) As geom
        ) As geoms) As facen CROSS JOIN generate_series(1,10000) As i
        WHERE i < npt;
-- result --
edgeid
-----
  3
  4
  5
  6
  7
  8
  9
 10
 11
 12
(10 rows)
-- then add the face -

SELECT topology.AddFace('ma_topo',
  ST_GeomFromText('POLYGON((234896.5 899456.7,234914 899436.4,234946.6 899356.9,234872.5  ←
    899328.7,
    234891 899285.4,234992.5 899145, 234890.6 899069,234755.2 899255.4,
    234612.7 899379.4,234776.9 899563.7,234896.5 899456.7))', 26986) ) As faceid;
```

```
-- result --
faceid
-----
1
```

### Siehe auch

[AddEdge](#), [CreateTopology](#), [Section 4.5](#)

## 9.8.5 ST\_Simplify

`ST_Simplify` — Gibt für eine `TopoGeometry` eine "vereinfachte" geometrische Version zurück. Verwendet den Douglas-Peucker Algorithmus.

### Synopsis

```
geometry ST_Simplify(TopoGeometry tg, float8 tolerance);
```

### Beschreibung

Gibt für eine `TopoGeometry` eine "vereinfachte" geometrische Version zurück. Wendet den Douglas-Peucker Algorithmus auf jede Kante an.



#### Note

Es kann vorkommen, dass die zurückgegebene Geometrie weder "simple" noch valide ist. Das Auftrennen der Kanten kann helfen, die Simplizität/Validität zu erhalten.

Wird vom GEOS Modul ausgeführt

Verfügbarkeit: 2.1.0

### Siehe auch

Geometrie [ST\\_Simplify](#), [ST\\_IsSimple](#), [ST\\_IsValid](#), [ST\\_ModEdgeSplit](#)

## 9.8.6 RemoveUnusedPrimitives

`RemoveUnusedPrimitives` — Entfernt Topologieprimitive, die zur Definition bestehender `TopoGeometry`-Objekte nicht benötigt werden.

### Synopsis

```
int RemoveUnusedPrimitives(text topology_name, geometry bbox);
```

## Beschreibung

Findet alle Primitive (Knoten, Kanten, Flächen), die nicht unbedingt zur Darstellung vorhandener TopoGeometry-Objekte benötigt werden, und entfernt sie, wobei die Gültigkeit der Topologie (Kantenverknüpfung, Flächenbeschriftung) und die Belegung des TopoGeometry-Raums erhalten bleiben.

Es werden keine neuen Primitivbezeichner erstellt, sondern die vorhandenen Primitive werden erweitert, um verschmolzene Flächen (beim Entfernen von Kanten) oder geheilte Kanten (beim Entfernen von Knoten) aufzunehmen.

Verfügbarkeit: 3.3.0

## Siehe auch

[ST\\_ModEdgeHeal](#), [ST\\_RemEdgeModFace](#)

## 9.9 TopoGeometry Konstruktoren

### 9.9.1 CreateTopoGeom

CreateTopoGeom — Erzeugt ein neues topologisch geometrisches Objekt aus einem Feld mit topologischen Elementen - tg\_type: 1:[multi]point, 2:[multi]line, 3:[multi]poly, 4:collection

## Synopsis

```
topogeometry CreateTopoGeom(varchar toponame, integer tg_type, integer layer_id, topoelementarray tg_objs);
topogeometry CreateTopoGeom(varchar toponame, integer tg_type, integer layer_id);
```

## Beschreibung

Erstellt ein TopoGeometry Objekt für den Layer, der über die `layer_id` angegeben wird, und registriert es in der Tabelle "relation" in dem Schema `toponame`.

`tg_type` ist eine Ganzzahl: 1:[multi]point (punktförmig), 2:[multi]line (geradlinig), 3:[multi]poly (flächenhaft), 4:collection. `layer_id` ist der Identifikator des Layers in der Tabelle "topology.layer".

Punktförmige Layer werden aus Knoten gebildet, linienförmige Layer aus Kanten, flächige Layer aus Maschen und Kollektionen können aus einer Mischung von Knoten, Kanten und Maschen gebildet werden.

Wird das Feld mit den Komponenten weggelassen, wird ein leeres TopoGeometrie Objekt erstellt.

Verfügbarkeit: 1.1

## Beispiele: Aus bestehenden Kanten bilden

Erstellt eine TopoGeometry im Schema "ri\_topo" für den Layer 2 (ri\_roads), vom Datentyp (2) LINE, für die erste Kante (die wir unter `ST_CreateTopoGeo` geladen haben).

```
INSERT INTO ri.ri_roads(road_name, topo) VALUES('Unknown', topology.CreateTopoGeom('ri_topo ←
',2,2, '{{1,2}}'::topology.topoelementarray);
```

### Beispiele: Eine flächige Geometrie in die vermutete TopoGeometry konvertieren

Angenommen wir wollen eine Geometrie aus einer Kollektion von Maschen bilden. Wir haben zum Beispiel die Tabelle "blockgroups" und wollen die TopoGeometry von jeder "blockgroup" wissen. Falls unsere Daten perfekt ausgerichtet sind, können wir folgendes ausführen:

```
-- create our topo geometry column --
SELECT topology.AddTopoGeometryColumn(
    'topo_boston',
    'boston', 'blockgroups', 'topo', 'POLYGON');

-- addtopogeometrycolumn --
1

-- update our column assuming
-- everything is perfectly aligned with our edges
UPDATE boston.blockgroups AS bg
    SET topo = topology.CreateTopoGeom('topo_boston'
    ,3,1
    , foo.bfaces)
FROM (SELECT b.gid, topology.TopoElementArray_Agg(ARRAY[f.face_id,3]) As bfaces
    FROM boston.blockgroups As b
    INNER JOIN topo_boston.face As f ON b.geom && f.mbr
    WHERE ST_Covers(b.geom, topology.ST_GetFaceGeometry('topo_boston', f.face_id))
    GROUP BY b.gid) As foo
WHERE foo.gid = bg.gid;
```

```
--the world is rarely perfect allow for some error
--count the face if 50% of it falls
-- within what we think is our blockgroup boundary
UPDATE boston.blockgroups AS bg
    SET topo = topology.CreateTopoGeom('topo_boston'
    ,3,1
    , foo.bfaces)
FROM (SELECT b.gid, topology.TopoElementArray_Agg(ARRAY[f.face_id,3]) As bfaces
    FROM boston.blockgroups As b
    INNER JOIN topo_boston.face As f ON b.geom && f.mbr
    WHERE ST_Covers(b.geom, topology.ST_GetFaceGeometry('topo_boston', f.face_id))
    OR
    ( ST_Intersects(b.geom, topology.ST_GetFaceGeometry('topo_boston', f.face_id))
    AND ST_Area(ST_Intersection(b.geom, topology.ST_GetFaceGeometry('topo_boston', ←
    f.face_id) ) ) >
    ST_Area(topology.ST_GetFaceGeometry('topo_boston', f.face_id))*0.5
    )
    GROUP BY b.gid) As foo
WHERE foo.gid = bg.gid;

-- and if we wanted to convert our topogeometry back
-- to a denormalized geometry aligned with our faces and edges
-- cast the topo to a geometry
-- The really cool thing is my new geometries
-- are now aligned with my tiger street centerlines
UPDATE boston.blockgroups SET new_geom = topo::geometry;
```

### Siehe auch

[AddTopoGeometryColumn](#), [toTopoGeom](#) [ST\\_CreateTopoGeo](#), [ST\\_GetFaceGeometry](#), [TopoElementArray](#), [TopoElementArray\\_Agg](#)

## 9.9.2 toTopoGeom

toTopoGeom — Wandelt eine einfache Geometrie in eine TopoGeometry um.

### Synopsis

```
topogeometry toTopoGeom(geometry geom, varchar toponame, integer layer_id, float8 tolerance);
topogeometry toTopoGeom(geometry geom, topogeometry topogeom, float8 tolerance);
```

### Beschreibung

Wandelt eine einfache Geometrie in eine **TopoGeometry** um.

Die topologischen Elementarstrukturen, die benötigt werden um die Übergabegeometrie darzustellen, werden der zugrunde liegenden Topologie hinzugefügt. Dabei können bestehende Strukturen aufgetrennt werden, die dann mit der ausgegebenen TopoGeometry in der Tabelle `relation` zusammengeführt werden.

Bestehende Objekte einer TopoGeometry (mit der möglichen Ausnahme von `topogeom`, falls angegeben) behalten ihre geometrische Gestalt.

Wenn `tolerance` angegeben ist, wird diese zum Fangen der Eingabegeometrie an bestehenden Elementarstrukturen verwendet.

Bei der ersten Form wird eine neue TopoGeometry für den Layer (`layer_id`) einer Topologie (`toponame`) erstellt.

Bei der zweiten Form werden die aus der Konvertierung 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

```
-- do this if you don't have a topology setup already
-- creates topology not allowing any tolerance
SELECT topology.CreateTopology('topo_boston_test', 2249);
-- create a new table
CREATE TABLE nei_topo(gid serial primary key, nei varchar(30));
--add a topogeometry column to it
SELECT topology.AddTopoGeometryColumn('topo_boston_test', 'public', 'nei_topo', 'topo', ' ←
MULTIPOLYGON') As new_layer_id;
new_layer_id
-----
1

--use new layer id in populating the new topogeometry column
-- we add the topogeoms to the new layer with 0 tolerance
INSERT INTO nei_topo(nei, topo)
SELECT nei, topology.toTopoGeom(geom, 'topo_boston_test', 1)
FROM neighborhoods
WHERE gid BETWEEN 1 and 15;

--use to verify what has happened --
SELECT * FROM
    topology.TopologySummary('topo_boston_test');

-- summary--
Topology topo_boston_test (5), SRID 2249, precision 0
```

```
61 nodes, 87 edges, 35 faces, 15 topogeoms in 1 layers
Layer 1, type Polygonal (3), 15 topogeoms
Deploy: public.nei_topo.topo
```

```
-- Shrink all TopoGeometry polygons by 10 meters
UPDATE nei_topo SET topo = ST_Buffer(clearTopoGeom(topo), -10);

-- Get the no-one-lands left by the above operation
-- I think GRASS calls this "polygon0 layer"
SELECT ST_GetFaceGeometry('topo_boston_test', f.face_id)
  FROM topo_boston_test.face f
 WHERE f.face_id
 > 0 -- don't consider the universe face
 AND NOT EXISTS ( -- check that no TopoGeometry references the face
   SELECT * FROM topo_boston_test.relation
   WHERE layer_id = 1 AND element_id = f.face_id
 );
```

### Siehe auch

[CreateTopology](#), [AddTopoGeometryColumn](#), [CreateTopoGeom](#), [TopologySummary](#), [clearTopoGeom](#)

### 9.9.3 TopoElementArray\_Agg

TopoElementArray\_Agg — Gibt für eine Menge an element\_id, type Feldern (topoelements) ein topoelementarray zurück.

#### Synopsis

topoelementarray **TopoElementArray\_Agg**(topoelement set tefield);

#### Beschreibung

Verwendet um ein **TopoElementArray** aus einer Menge an **TopoElement** zu erstellen.

Verfügbarkeit: 2.0.0

#### Beispiele

```
SELECT topology.TopoElementArray_Agg(ARRAY[e,t]) As tea
  FROM generate_series(1,3) As e CROSS JOIN generate_series(1,4) As t;
  tea
-----
{{1,1},{1,2},{1,3},{1,4},{2,1},{2,2},{2,3},{2,4},{3,1},{3,2},{3,3},{3,4}}
```

### Siehe auch

[TopoElement](#), [TopoElementArray](#)

### 9.9.4 TopoElement

TopoElement — Konvertiert eine Topogeometrie in ein Topoelement.



## Synopsis

topoelement **TopoElement**(topogeometry topo);

## Beschreibung

Konvertiert eine **TopoGeometry** in eine **TopoElement**.

Verfügbarkeit: 3.4.0

## Beispiele

Dies ist ein in sich selbst vollkommen abgeschlossener Arbeitsablauf

```
-- do this if you don't have a topology setup already
-- Creates topology not allowing any tolerance
SELECT TopoElement(topo)
FROM neighborhoods;
```

```
-- using as cast
SELECT topology.TopoElementArray_Agg(topo::topoelement)
FROM neighborhoods
GROUP BY city;
```

## Siehe auch

[TopoElementArray\\_Agg](#), [TopoGeometry](#), [TopoElement](#)

## 9.10 TopoGeometry Editoren

### 9.10.1 clearTopoGeom

clearTopoGeom — Löscht den Inhalt einer TopoGeometry.

## Synopsis

topogeometry **clearTopoGeom**(topogeometry topogeom);

## Beschreibung

Löscht den Inhalt einer **TopoGeometry** und wandelt sie in eine leere um. Am nützlichsten in Verbindung mit **toTopoGeom**, um die geometrische Gestalt bestehende Objekte und alle abhängigen Objekte in höheren hierarchischen Ebenen zu ersetzen.

Verfügbarkeit: 2.1

## Beispiele

```
-- Shrink all TopoGeometry polygons by 10 meters
UPDATE nei_topo SET topo = ST_Buffer(clearTopoGeom(topo), -10);
```

**Siehe auch**

[toTopoGeom](#)

### 9.10.2 TopoGeom\_addElement

TopoGeom\_addElement — Fügt ein Element zu der Definition einer TopoGeometry hinzu.

**Synopsis**

```
topogeometry TopoGeom_addElement(topogeometry tg, topoelement el);
```

**Beschreibung**

Fügt ein **TopoElement** zur Definition eines TopoGeometry-Objekts hinzu. Wenn das Element bereits Teil der Definition ist, führt dies zu keinem Fehler.

Verfügbarkeit: 2.3

**Beispiele**

```
-- Add edge 5 to TopoGeometry tg
UPDATE mylayer SET tg = TopoGeom_addElement(tg, '{5,2}');
```

**Siehe auch**

[TopoGeom\\_remElement](#), [CreateTopoGeom](#)

### 9.10.3 TopoGeom\_remElement

TopoGeom\_remElement — Entfernt ein Element aus der Definition einer TopoGeometry.

**Synopsis**

```
topogeometry TopoGeom_remElement(topogeometry tg, topoelement el);
```

**Beschreibung**

Entfernt ein **TopoElement** aus der Ausgestaltung des TopoGeometry Objekts.

Verfügbarkeit: 2.3

**Beispiele**

```
-- Remove face 43 from TopoGeometry tg
UPDATE mylayer SET tg = TopoGeom_remElement(tg, '{43,3}');
```

**Siehe auch**

[TopoGeom\\_addElement](#), [CreateTopoGeom](#)

---

### 9.10.4 TopoGeom\_addTopoGeom

TopoGeom\_addTopoGeom — Fügt Element einer TopoGeometry zur Definition einer anderen TopoGeometry hinzu.

#### Synopsis

```
topogeometry TopoGeom_addTopoGeom(topogeometry tgt, topogeometry src);
```

#### Beschreibung

Fügt die Elemente einer **TopoGeometry** zur Definition einer anderen TopoGeometry hinzu, wobei der zwischengespeicherte Typ (type-Attribut) gegebenenfalls in eine Sammlung geändert wird, um alle Elemente des Quellobjekts aufzunehmen.

Die beiden TopoGeometry-Objekte müssen für die *gleiche* Topologie definiert sein und, falls sie hierarchisch definiert sind, aus Elementen derselben untergeordneten Ebene bestehen.

Verfügbarkeit: 3.2

#### Beispiele

```
-- Set an "overall" TopoGeometry value to be composed by all
-- elements of specific TopoGeometry values
UPDATE mylayer SET tg_overall = TopoGeom_addTopoGeom(
    TopoGeom_addTopoGeom(
        clearTopoGeom(tg_overall),
        tg_specific1
    ),
    tg_specific2
);
```

#### Siehe auch

[TopoGeom\\_addElement](#), [clearTopoGeom](#), [CreateTopoGeom](#)

### 9.10.5 toTopoGeom

toTopoGeom — Fügt eine Geometrie zu einer bestehenden TopoGeometry hinzu.

#### Beschreibung

Siehe [toTopoGeom](#).

## 9.11 TopoGeometry Accessors

### 9.11.1 GetTopoGeomElementArray

GetTopoGeomElementArray — Gibt ein `topoelementarray` (ein Feld von `topoelements`) zurück, das die topologischen Elemente und den Datentyp der gegebenen TopoGeometry (die Elementarstrukturen) enthält.

## Synopsis

```
topoelementarray GetTopoGeomElementArray(varchar toponame, integer layer_id, integer tg_id);  
topoelementarray GetTopoGeomElementArray(topogeometry tg);
```

## Beschreibung

Gibt ein **TopoElementArray** zurück, das die topologischen Elemente und den Datentyp der gegebenen TopoGeometry (die Elementarstrukturen) enthält. Dies ist ähnlich dem **GetTopoGeomElements**, ausser dass die Elemente als Feld statt als Datensatz ausgegeben werden.

tg\_id steht für die ID des TopoGeometry Objekts der Topologie eines Layers, der durch die layer\_id der Tabelle "topology.layer" angegeben wird.

Verfügbarkeit: 1.1

## Beispiele

### Siehe auch

[GetTopoGeomElements](#), [TopoElementArray](#)

## 9.11.2 GetTopoGeomElements

**GetTopoGeomElements** — Gibt für eine TopoGeometry (Elementarstrukturen) einen Satz an topoelement Objekten zurück, welche die topologische element\_id und den element\_type beinhalten.

## Synopsis

```
setof topoelement GetTopoGeomElements(varchar toponame, integer layer_id, integer tg_id);  
setof topoelement GetTopoGeomElements(topogeometry tg);
```

## Beschreibung

Gibt eine Menge von element\_id,element\_type (topoelements) zurück, die den primitiven Topologieelementen **TopoElement** (1: Knoten, 2: Kanten, 3: Flächen) entsprechen, aus denen ein bestimmtes topogeometrisches Objekt im Schema toponame besteht.

tg\_id steht für die ID des TopoGeometry Objekts der Topologie eines Layers, der durch die layer\_id der Tabelle "topology.layer" angegeben wird.

Verfügbarkeit: 2.0.0

## Beispiele

### Siehe auch

[GetTopoGeomElementArray](#), [TopoElement](#), [TopoGeom\\_addElement](#), [TopoGeom\\_remElement](#)

## 9.11.3 ST\_SRID

**ST\_SRID** — Gibt den räumlichen Referenzbezeichner für eine Topogeometrie zurück.

## Synopsis

integer **ST\_SRID**(topogeometry tg);

## Beschreibung

Liefert die Kennung des Raumbezugs für die ST\_Geometrie, wie in der Tabelle spatial\_ref\_sys definiert. Section [4.5](#)



### Note

Die Tabelle spatial\_ref\_sys ist eine Tabelle, die alle PostGIS bekannten räumlichen Bezugssysteme katalogisiert und für Transformationen von einem räumlichen Bezugssystem in ein anderes verwendet wird. Daher ist es wichtig, dass Sie die richtige Kennung für das räumliche Bezugssystem haben, wenn Sie Ihre Geometrien transformieren möchten.

Verfügbarkeit: 3.2.0



Diese Methode setzt die SQL/MM-Spezifikation um. SQL-MM 3: 14.1.5

## Beispiele

```
SELECT ST_SRID(ST_GeomFromText('POINT(-71.1043 42.315)', 4326));
--result
4326
```

## Siehe auch

Section [4.5](#), [ST\\_SetSRID](#), [ST\\_Transform](#), [ST\\_SRID](#)

## 9.12 TopoGeometry Ausgabe

### 9.12.1 AsGML

AsGML — Gibt die GML-Darstellung einer TopoGeometry zurück.

## Synopsis

```
text AsGML(topogeometry tg);
text AsGML(topogeometry tg, text namespace_in);
text AsGML(topogeometry tg, regclass visitedTable);
text AsGML(topogeometry tg, regclass visitedTable, text namespace);
text AsGML(topogeometry tg, text namespace_in, integer precision, integer options);
text AsGML(topogeometry tg, text namespace_in, integer precision, integer options, regclass visitedTable);
text AsGML(topogeometry tg, text namespace_in, integer precision, integer options, regclass visitedTable, text idprefix);
text AsGML(topogeometry tg, text namespace_in, integer precision, integer options, regclass visitedTable, text idprefix, int gmlversion);
```

## Beschreibung

Gibt die GML-Darstellung einer TopoGeometry im GML3-Format aus. Wenn kein `nsprefix_in` angegeben ist, dann wird `gml` verwendet. Übergeben sie eine leere Zeichenfolge für "nsprefix" um keinen bestimmten Namensraum festzulegen. Wenn die Parameter "precision" (Standardwert: 15) und "options" (Standardwert: 1) angegeben sind, werden diese unangetastet an den zugrunde liegenden Aufruf von `ST_AsGML` übergeben.

Wenn der Parameter `visitedTable` angegeben ist, dann wird dieser verwendet um die bereits besuchten Knoten und Kanten über Querverweise (`xlink:xref`) zu verfolgen, anstatt Definitionen zu vervielfältigen. Die Tabelle muss (zumindest) zwei Interfelder enthalten: 'element\_type' und 'element\_id'. Für den Aufruf muss der Anwender sowohl Lese- als auch Schreibrechte auf die Tabelle besitzen. Um die maximale Rechenleistung zu erreichen, sollte ein Index für die Attribute `element_type` und `element_id` - in dieser Reihenfolge - festgelegt werden. Dieser Index wird automatisch erstellt, wenn auf die Attribute ein Unique Constraint gelegt wird. Beispiel:

```
CREATE TABLE visited (
  element_type integer, element_id integer,
  unique(element_type, element_id)
);
```

Wird der Parameter `idprefix` angegeben, so wird dieser den Identifikatoren der Tags von Kanten und Knoten vorangestellt.

Wird der Parameter `gmlver` angegeben, so wird dieser an 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:directedEdge>
  <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](#)

## 9.12.2 AsTopoJSON

AsTopoJSON — Gibt die TopoJSON-Darstellung einer TopoGeometry zurück.

### Synopsis

```
text AsTopoJSON(topogeometry tg, regclass edgeMapTable);
```

### Beschreibung

Gibt eine TopoGeometry in der TopoJSON-Darstellung zurück. Wenn `edgeMapTable` nicht NULL ist, wird diese als Lookup/Speicher für die Abbildung der Identifikatoren der Kanten auf die Indizes der Kreisbögen verwendet. Dadurch wird ein kompaktes Feld "arcs" im endgültigen Dokument ermöglicht.

Wenn die Tabelle angegeben ist, wird die Existenz der Attribute "arc\_id" vom Datentyp "serial" und "edge\_id" vom Typ "integer" vorausgesetzt; da der Code die Tabelle nach der "edge\_id" abfragt, sollte ein Index für dieses Attribut erstellt werden.



#### Note

Die Kreisbögen in der TopoJSON Ausgabe sind von 0 weg indiziert, während sie in der Tabelle "edgeMapTable" 1-basiert sind.

Ein vollständiges TopoJson Dokument benötigt zusätzlich zu den von dieser Funktion ausgegebenen Schnipseln, die tatsächlichen Bögen und einige Header. 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 ←
": {'

-- objects
UNION ALL SELECT '' || feature_name || ': ' || AsTopoJSON(feature, 'edgemap')
FROM features.big_parcel WHERE feature_name = 'P3P4';

-- arcs
WITH edges AS (
  SELECT m.arc_id, e.geom FROM edgemap m, city_data.edge e
  WHERE e.edge_id = m.edge_id
), points AS (
  SELECT arc_id, (st_dumppoints(geom)).* FROM edges
), compare AS (
  SELECT p2.arc_id,
     CASE WHEN p1.path IS NULL THEN p2.geom
          ELSE ST_Translate(p2.geom, -ST_X(p1.geom), -ST_Y(p1.geom))
  END AS geom
```



```

FROM points p2 LEFT OUTER JOIN points p1
ON ( p1.arc_id = p2.arc_id AND p2.path[1] = p1.path[1]+1 )
ORDER BY arc_id, p2.path
), 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]]
]]}

```

## 9.13 Räumliche Beziehungen einer Topologie

### 9.13.1 Equals

Equals — Gibt TRUE zurück, wenn zwei TopoGeometry Objekte aus denselben topologischen Elementarstrukturen bestehen.

#### Synopsis

```
boolean Equals(topogeometry tg1, topogeometry tg2);
```

#### Beschreibung

Gibt TRUE zurück, wenn zwei TopoGeometry Objekte aus denselben topologischen Elementarstrukturen: Maschen, Kanten, Knoten, bestehen.



#### Note

Diese Funktion unterstützt keine TopoGeometry aus einer Sammelgeometrie. Es kann auch keine TopoGeometry Objekte unterschiedlicher Topologien vergleichen

Verfügbarkeit: 1.1.0



Diese Funktion unterstützt 3d und lässt den Z-Index nicht fallen.

## Beispiele

### Siehe auch

[GetTopoGeomElements](#), [ST\\_Equals](#)

## 9.13.2 Intersects

**Intersects** — Gibt TRUE zurück, wenn sich kein beliebiges Paar von Elemtarstrukturen zweier TopoGeometry Objekte überschneidet.

### Synopsis

```
boolean Intersects(topogeometry tg1, topogeometry tg2);
```

### Beschreibung

Gibt TRUE zurück, wenn sich kein beliebiges Paar von Elemtarstrukturen zweier TopoGeometry Objekte überschneidet.



#### Note

Diese Funktion wird nicht für Topogeometrien unterstützt, die Geometriesammlungen sind. Sie kann auch keine Topogeometrien aus verschiedenen Topologien vergleichen. Außerdem wird sie derzeit nicht für hierarchische Topogeometrien (Topogeometrien, die aus anderen Topogeometrien bestehen) unterstützt.

Verfügbarkeit: 1.1.0



Diese Funktion unterstützt 3d und lässt den Z-Index nicht fallen.

## Beispiele

### Siehe auch

[ST\\_Intersects](#)

## 9.14 Importieren und Exportieren von Topologien

Nachdem Sie Topologien und eventuell zugehörige topologische Ebenen erstellt haben, möchten Sie diese vielleicht in ein dateibasiertes Format exportieren, um sie zu sichern oder in eine andere Datenbank zu übertragen.

Die Verwendung der Standard-Dump/Restore-Tools von PostgreSQL ist problematisch, da Topologien aus einer Reihe von Tabellen (4 für Primitive, eine beliebige Anzahl für Layer) und Datensätzen in Metadatentabellen (`topology.topology` und `topology.layer`) bestehen. Außerdem sind die Topologiebezeichner nicht datenbankübergreifend einheitlich, so dass die Parameter Ihrer Topologie bei der Wiederherstellung geändert werden müssen.

Um das Exportieren/Wiederherstellen von Topologien zu vereinfachen, werden zwei ausführbare Dateien bereitgestellt: `pgtopo_export` und `pgtopo_import`. Beispiel für die Verwendung:

```
pgtopo_export dev_db topo1 | pgtopo_import topo1 | psql staging_db
```

### 9.14.1 Verwendung des Topologie-Exporters

Das Skript `pgtopo_export` nimmt den Namen einer Datenbank und eine Topologie und gibt eine Dump-Datei aus, die zum Importieren der Topologie (und der zugehörigen Ebenen) in eine neue Datenbank verwendet werden kann.

Standardmäßig schreibt `pgtopo_export` die Speicherauszugsdatei in die Standardausgabe, so dass sie an `pgtopo_import` weitergeleitet oder in eine Datei umgeleitet werden kann (Ablehnung des Schreibens ins Terminal). Sie können optional einen Ausgabedateinamen mit der Befehlszeilenoption `-f` angeben.

Standardmäßig enthält `pgtopo_export` einen Speicherauszug aller für die angegebene Topologie definierten Ebenen. Dies können mehr Daten sein, als Sie benötigen, oder sie können nicht funktionieren (falls Ihre Ebenentabellen komplexe Abhängigkeiten haben). In diesem Fall können Sie mit dem Schalter `--skip-layers` verlangen, dass die Ebenen übersprungen werden, und diese separat behandeln.

Der Aufruf von `pgtopo_export` mit dem Schalter `--help` (oder kurz `-h`) gibt immer einen kurzen Verwendungsstring aus.

Das Format der Dump-Datei ist ein komprimiertes tar-Archiv eines `pgtopo_export` Verzeichnisses, das mindestens eine `pgtopo_dump_version` Datei mit Informationen zur Formatversion enthält. Ab Version 1 enthält das Verzeichnis tabulartrennte CSV-Dateien mit den Daten der Topologie-Primitivtabellen (`node`, `edge_data`, `face`, `relation`), den damit verbundenen Topologie- und Ebenendatensätzen und (sofern nicht `--skip-layers` angegeben ist) einem PostgreSQL-Dump im benutzerdefinierten Format der Tabellen, die als Ebenen der angegebenen Topologie gemeldet werden.

### 9.14.2 Verwendung des Topologie-Importers

Das Skript `pgtopo_import` nimmt einen Topologie-Dump im Format `pgtopo_export` und einen Namen für die zu erstellende Topologie und gibt ein SQL-Skript aus, das die Topologie und die zugehörigen Ebenen rekonstruiert.

Die generierte SQL-Datei enthält Anweisungen, die eine Topologie mit dem angegebenen Namen erstellen, Primitivdaten in sie laden, alle Topologieebenen wiederherstellen und registrieren, indem sie alle TopoGeometry-Werte ordnungsgemäß mit der richtigen Topologie verknüpfen.

Standardmäßig liest `pgtopo_import` den Speicherauszug von der Standardeingabe, so dass er in Verbindung mit `pgtopo_export` in einer Pipeline verwendet werden kann. Sie können optional einen Eingabedateinamen mit der Befehlszeilenoption `-f` angeben.

Standardmäßig enthält `pgtopo_import` in der SQL-Ausgabedatei den Code zur Wiederherstellung aller im Dump gefundenen Ebenen.

Dies kann unerwünscht sein oder nicht funktionieren, wenn Ihre Zieldatenbank bereits Tabellen mit dem gleichen Namen wie die im Dump hat. In diesem Fall können Sie das Überspringen der Ebenen mit dem Schalter `--skip-layers` anfordern und diese separat (oder später) bearbeiten.

SQL, um nur Ebenen zu laden und mit einer benannten Topologie zu verknüpfen, kann mit dem Schalter `--only-layers` erzeugt werden. Dies kann nützlich sein, um Ebenen zu laden, NACHDEM die Benennungskonflikte gelöst wurden, oder um Ebenen mit einer anderen Topologie zu verknüpfen (z. B. einer räumlich vereinfachten Version der Ausgangstopologie).

## Chapter 10

# Rasterdatenverwaltung, -abfrage und Anwendungen

### 10.1 Laden und Erstellen von Rastertabellen

In den häufigsten Anwendungsfällen werden Sie einen PostGIS-Raster durch das Laden einer bestehenden Rasterdatei, mit Hilfe des Rasterladers `raster2pgsql`, erstellen.

#### 10.1.1 Verwendung von `raster2pgsql` zum Laden von Rastern

Das `raster2pgsql` ist ein Rasterladeprogramm, das GDAL-unterstützte Rasterformate in eine SQL-Datei lädt, die für das Laden in eine PostGIS-Rastertabelle geeignet ist. Es ist in der Lage, Ordner mit Rasterdateien zu laden und Übersichten über Raster zu erstellen.

Da `raster2pgsql` in den meisten Fällen als Teil von PostGIS kompiliert wird (es sei denn, Sie kompilieren Ihre eigene GDAL-Bibliothek), werden die von der ausführbaren Datei unterstützten Rastertypen dieselben sein wie die in der GDAL-Abhängigkeitsbibliothek kompilierten. Um eine Liste der Rastertypen zu erhalten, die Ihr spezielles `raster2pgsql` unterstützt, verwenden Sie den Schalter `-G`.



#### 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.

##### 10.1.1.1 Beispiel für die Verwendung

Eine Beispielsitzung, wo mit dem Lader eine Eingabedatei erstellt und stückchenweise als 100x100 Kacheln hochgeladen wird, könnte so aussehen:

```
# -s use srid 4326
# -I create spatial index
# -C use standard raster constraints
# -M vacuum analyze after load
# *.tif load all these files
# -F include a filename column in the raster table
# -t tile the output 100x100
# public.demelevation load into this table
raster2pgsql -s 4326 -I -C -M -F -t 100x100 *.tif public.demelevation
> elev.sql
```

```
# -d connect to this database
# -f read this file after connecting
psql -d gisdb -f elev.sql
```

**Note**

Wenn Sie das Schema nicht als Teil des Namens der Zieltabelle angeben, wird die Tabelle im Standardschema der Datenbank oder des Benutzers, mit dem Sie sich verbinden, erstellt.

Durch die Verwendung von UNIX-Pipes kann die Konvertierung und der Upload in einem Schritt vollzogen werden:

```
raster2pgsql -s 4326 -I -C -M *.tif -F -t 100x100 public.demelevation | psql -d gisdb
```

Luftbildkacheln in "Massachusetts State Plane Meters" in das Schema `aerial` laden. Einen vollständigen View und Übersichtstabellen mit Faktor 2 und 4 erstellen. Verwendet den Modus "copy" für das Insert (keine dazwischengeschaltete Datei, sondern direkt in die Datenbank). Die Option `-e` bedingt, dass nicht alles innerhalb einer Transaktion abläuft (nützlich, wenn Sie sofort Daten sehen wollen, ohne zu warten). Die Raster werden in 128x128 Pixel große Kacheln zerlegt und Constraints auf die Raster gesetzt. Verwendet den Modus "copy" anstelle eines Tabellen-Inserts. (`-F`) Erzeugt das Attribut "filename", welches die Bezeichnung der Ausgangsdateien enthält, aus denen die Rasterkacheln ausgeschnitten wurden.

```
raster2pgsql -I -C -e -Y -F -s 26986 -t 128x128 -l 2,4 bostonaerials2008/*.jpg aerials. ↔
  boston | psql -U postgres -d gisdb -h localhost -p 5432
```

```
--get a list of raster types supported:
raster2pgsql -G
```

Der `-G` Befehl gibt eine ähnliche Liste wie die Folgende aus

```
Available GDAL raster formats:
  Virtual Raster
  GeoTIFF
  National Imagery Transmission Format
  Raster Product Format TOC format
  ECRG TOC format
  Erdas Imagine Images (.img)
  CEOS SAR Image
  CEOS Image
  ...
  Arc/Info Export E00 GRID
  ZMap Plus Grid
  NOAA NGS Geoid Height Grids
```

### 10.1.1.2 raster2pgsql-Optionen

`-?` Zeigt die Hilfe an, auch dann, wenn keine Argumente übergeben werden.

`-G` Gibt die unterstützten Rasterformate aus.

**(claldp) Dies sind sich gegenseitig ausschließende Optionen:**

- `-c` Eine neue Tabelle anlegen und mit Raster(n) befüllen, *this is the default mode*
- `-a` Raster zu einer bestehende Tabelle hinzufügen.
- `-d` Tabelle löschen, eine Neu erzeugen und mit einem oder mehreren Raster befüllen
- `-p` Beim vorbereitenden Modus wird 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**

- 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 Behält leere Kacheln bei und überspringt die Überprüfung der NODATA-Werte für jedes Rasterband. Beachten Sie, dass Sie bei der Überprüfung Zeit sparen, aber am Ende viel mehr Junk-Zeilen in Ihrer Datenbank haben können, die nicht als leere Kacheln markiert sind.
  - T tablespace Bestimmt den Tablespace für die neue Tabelle. Beachten Sie bitte, dass Indizes (einschließlich des Primärschlüssels) weiterhin den standardmäßigen Tablespace nutzen, solange nicht die -X Flag benutzt wird.
  - X tablespace Bestimmt den Tablespace für den neuen Index der Tabelle. Dieser gilt sowohl für den Primärschlüssel als auch für den räumlichen Index, falls die -I Flag gesetzt ist.
  - Y max\_rows\_per\_copy=50 Verwenden Sie Kopieranweisungen anstelle von Einfügeanweisungen. Optional können Sie `max_rows_per_copy` angeben; Standardwert 50, wenn nicht angegeben.
- 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.

## 10.1.2 Erzeugung von Rastern mit den PostGIS Rasterfunktionen

Oftmals werden Sie die Raster und die Rastertabellen direkt in der Datenbank erzeugen wollen. Dafür existieren eine Unmenge an Funktionen. Dies verlangt im Allgemeinen die folgende Schritte.

1. Erstellung einer Tabelle mit einer Rasterspalte für die neuen Rasterdatensätze:

```
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.

## 10.1.3 Verwendung von "out db"-Wolkenrastern

Das Tool `raster2pgsql` verwendet GDAL, um auf Rasterdaten zuzugreifen, und kann eine wichtige GDAL-Funktion nutzen: die Fähigkeit, von Rastern zu lesen, die **entfernt** in Cloud-"Objektspeichern" (z. B. AWS S3, Google Cloud Storage) gespeichert sind.

Eine effiziente Nutzung von in der Cloud gespeicherten Rastern erfordert die Verwendung eines "cloudoptimierten" Formats. Das bekannteste und am weitesten verbreitete Format ist das **"cloud optimized GeoTIFF"**. Die Verwendung eines nicht cloud-optimierten Formats wie JPEG oder TIFF ohne Kacheln führt zu einer sehr schlechten Leistung, da das System jedes Mal das gesamte Raster herunterladen muss, wenn es auf eine Teilmenge zugreifen will.

Laden Sie zunächst Ihr Raster in den Cloud-Speicher Ihrer Wahl. Sobald es geladen ist, haben Sie eine URI, mit der Sie darauf zugreifen können, entweder eine "http"-URI oder manchmal eine für den Dienst spezifische URI. (z. B. "s3://bucket/object"). Um auf nicht-öffentliche Buckets zuzugreifen, müssen Sie GDAL-Konfigurationsoptionen angeben, um Ihre Verbindung zu authentifizieren. Beachten Sie, dass dieser Befehl aus dem Cloud-Raster liest und in die Datenbank schreibt.

```
AWS_ACCESS_KEY_ID=xxxxxxxxxxxxxxxxxxxxxx \
AWS_SECRET_ACCESS_KEY=xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx \
raster2pgsql \
-s 990000 \
-t 256x256 \
-I \
-R \
```

```
/vsis3/your.bucket.com/your_file.tif \
your_table \
| psql your_db
```

Sobald die Tabelle geladen ist, müssen Sie der Datenbank die Berechtigung geben, von entfernten Rastern zu lesen, indem Sie zwei Berechtigungen festlegen: `postgis.enable_outdb_rasters` und `postgis.gdal_enabled_drivers`.

```
SET postgis.enable_outdb_rasters = true;
SET postgis.gdal_enabled_drivers TO 'ENABLE_ALL';
```

Um die Änderungen dauerhaft zu machen, legen Sie sie direkt in Ihrer Datenbank fest. Sie müssen die Verbindung erneut herstellen, damit die neuen Einstellungen wirksam werden.

```
ALTER DATABASE your_db SET postgis.enable_outdb_rasters = true;
ALTER DATABASE your_db SET postgis.gdal_enabled_drivers TO 'ENABLE_ALL';
```

Für nicht-öffentliche Raster müssen Sie möglicherweise Zugriffsschlüssel bereitstellen, um aus den Cloud-Rastern zu lesen. Dieselben Schlüssel, die Sie für den Aufruf `raster2pgsql` verwendet haben, können für die Verwendung innerhalb der Datenbank mit der Konfiguration `postgis.gdal_vsi_options` festgelegt werden. Beachten Sie, dass mehrere Optionen gesetzt werden können, indem die Schlüssel=Wert-Paare durch Leerzeichen getrennt werden.

```
SET postgis.gdal_vsi_options = 'AWS_ACCESS_KEY_ID=xxxxxxxxxxxxxxxxxxxxxxxxx
AWS_SECRET_ACCESS_KEY=xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx';
```

Sobald Sie die Daten geladen und die Berechtigungen festgelegt haben, können Sie mit der Rastertabelle wie mit jeder anderen Rastertabelle interagieren und dieselben Funktionen verwenden. Die Datenbank kümmert sich um die Verbindung zu den Cloud-Daten, wenn sie Pixeldaten lesen muss.

## 10.2 Raster 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.

### 10.2.1 Rasterspalten Katalog

`raster_columns` ist ein Katalog mit allen Rasterspalten Ihrer Datenbanktabellen. Es handelt sich dabei um einen View, der die Constraints auf die Tabellen ausnutzt, um so immer konsistent mit dem aktuellen Stand der Datenbank zu bleiben; sogar dann, wenn Sie den Raster aus einem Backup oder einer anderen Datenbank wiederherstellen. Der `raster_columns` Katalog beinhaltet die folgenden Spalten.

Falls Sie Ihre Tabellen nicht mit dem Loader erstellt haben, oder vergessen haben, die `-C` Option während des Ladens anzugeben, können Sie die Constraints auch anschließend erzwingen, indem Sie `AddRasterConstraints` verwenden, wodurch der `raster_columns` Katalog die Information über Ihre Rasterkacheln, wie üblich abspeichert.

- `r_table_catalog` Die Datenbank, in der sich die Tabelle befindet. Greift immer auf die aktuelle Datenbank zu.
- `r_table_schema` Das Datenbankschema in dem sich die Rastertabelle befindet.
- `r_table_name` Rastertabelle



- `r_raster_column` Die Spalte, in der Tabelle `r_table_name`, die den Datentyp Raster aufweist. In PostGIS gibt es nichts, was Sie daran hindert, mehrere Rasterspalten in einer Tabelle zu haben. Somit ist es möglich auf unterschiedliche Raster(spalten) in einer einzigen Rastertabelle zuzugreifen.
- `srid` Der Identifikator für das Koordinatensystem in dem der Raster vorliegt. Sollte in Section 4.5 eingetragen sein.
- `scale_x` Der Skalierungsfaktor zwischen den Koordinaten der Vektoren und den Pixeln. Dieser steht nur dann zur Verfügung, wenn alle Kacheln der Rasterspalte denselben `scale_x` aufweisen und dieser Constraint auch gesetzt ist. Siehe [ST\\_ScaleX](#) für genauere Angaben.
- `scale_y` Der Skalierungsfaktor zwischen den Koordinaten der Vektoren und den Pixeln. Dieser steht nur dann zur Verfügung, wenn alle Kacheln der Rasterspalte denselben `scale_y` aufweisen und der Constraint `scale_y` auch gesetzt ist. Siehe [ST\\_ScaleY](#) für genauere Angaben.
- `blocksize_x` Die Breite (Anzahl der waagrechten Zellen) einer Rasterkachel. Siehe [ST\\_Width](#) für weitere Details.
- `blocksize_y` Die Höhe (Anzahl der senkrechten Zellen) einer Rasterkachel. Siehe [ST\\_Height](#) für weitere Details.
- `same_alignment` Eine boolesche Variable, die TRUE ist, wenn alle Rasterkacheln dieselbe Ausrichtung haben. Siehe [ST\\_SameAlignment](#) für genauere Angaben.
- `regular_blocking` Wenn auf die Rasterspalte die Constraints für die räumliche Eindeutigkeit und für die Coveragekachel gesetzt sind, ist der Wert TRUE, ansonsten FALSE..
- `num_bands` Die Anzahl der Bänder, die jede Kachel des Rasters aufweist. `ST_NumBands` gibt die gleiche Information aus. [ST\\_NumBands](#)
- `pixel_types` Ein Feld das den Pixeltyp für die Bänder festlegt. Die Anzahl der Elemente in diesem Feld entspricht der Anzahl der Rasterbänder. Die "pixel\_types" sind unter [ST\\_BandPixelType](#) definiert.
- `nodata_values` Ein Feld mit Double Precision Zahlen, welche den `nodata_value` für jedes Band festlegen. Die Anzahl der Elemente in diesem Feld entspricht der Anzahl der Rasterbänder. Diese Zahlen legen den Pixelwert für jedes Rasterband fest, der bei den meisten Operationen ignoriert 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.

## 10.2.2 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 `-1` Switch beim Laden des Rasters angeben, werden diese gemeinsam mit der Rasterhaupttabelle erstellt; sie können aber auch händisch über [AddOverviewConstraints](#) erstellt werden.

Übersichtstabellen enthalten dieselben Constraints wie andere Rastertabellen und zusätzliche informative Constraints, spezifisch für die Übersichten.



### Note

Die Information in `raster_overviews` befindet sich nicht in `raster_columns`. Falls Sie die Information der Übersichtstabelle und der `raster_columns` zusammen benötigen, können Sie einen Join auf `raster_overviews` und `raster_columns` ausführen, um die gesamte Information zu erhalten.

Die zwei Hauptgründe für Übersichtsraster sind:

1. Eine niedrig aufgelöste Darstellung der Basistabellen; wird im Allgemeinen zum schnellen Hinauszoomen verwendet.
2. Die Berechnungen laufen grundsätzlich schneller ab, als bei den Stammdaten mit höherer Auflösung, da weniger Datensätze vorhanden sind und die Pixel eine größere Fläche abdecken. Obwohl diese Berechnungen nicht so exakt sind, wie jene auf die hochauflösenden Stammtabellen, sind sie doch für viele 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 Übersichtsdienst gehört.
- `r_table_name` Die Rastertabelle, welche von dieser Übersicht bedient wird.
- `r_raster_column` Die Rasterspalte, die diese Overviewspalte bedient.
- `overview_factor` - der Pyramidenlevel der Übersichtstabelle. Umso größer die Zahl ist, desto geringer ist die Auflösung. Wenn ein Ordner für die Bilder angegeben ist, rechnet `raster2pgsql` eine Übersicht für jede Bilddatei und ladet diese einzeln. Es wird Level 1 und die Ursprungsdatei angenommen. Beim Level 2 repräsentiert jede Kachel 4 Originalkacheln. Angenommen Sie haben einen Ordner mit Bilddateien in einer Auflösung von 5000x5000 Pixel, die Sie auf 125x125 große Kacheln zerlegen wollen. Für jede Bilddatei enthält die Basistabelle  $(5000*5000)/(125*125)$  Datensätze = 1600, Ihre (l=2) `o_2` Tabelle hat dann eine Obergrenze von  $(1600/Power(2,2)) = 400$  Zeilen, Ihre (l=3) `o_3` ( $1600/Power(2,3)$ ) = 200 Zeilen. Wenn sich die Pixel nicht durch die Größe Ihrer Kacheln teilen lassen, erhalten Sie einige Ausschusskacheln (Kacheln die nicht zur Gänze gefüllt sind). Beachten Sie bitte, dass jede durch `raster2pgsql` erzeugte Übersichtskachel dieselbe Pixelanzahl hat wie die ursprüngliche Kachel, aber eine geringere Auflösung, wo ein Pixel ( $Power(2,overview\_factor)$  Pixel der Ursprungsdatei) repräsentiert.

## 10.3 Eigene Anwendungen mit PostGIS Raster erstellen

Die Tatsache, dass PostGIS Raster Ihnen SQL-Funktionen zum Rendern von Rastern in bekannten Bildformaten zur Verfügung stellt, gibt Ihnen eine Vielzahl von Möglichkeiten, diese zu rendern. Zum Beispiel können Sie OpenOffice / LibreOffice für das Rendering verwenden, wie in [Rendering PostGIS Raster graphics with LibreOffice Base Reports](#) gezeigt wird. Darüber hinaus können Sie eine Vielzahl von Sprachen verwenden, wie in diesem Abschnitt gezeigt wird.

### 10.3.1 PHP Beispiel: Ausgabe mittels `ST_AsPNG` in Verbindung mit anderen Rasterfunktionen

In diesem Abschnitt zeigen wir die Anwendung des PHP PostgreSQL Treibers und der Funktion `ST_AsGDALRaster`, um die Bänder 1,2,3 eines Rasters an einen PHP Request-Stream zu übergeben. Dieser kann dann in einen "img src" HTML Tag eingebunden werden.

Dieses Beispiel zeigt, wie Sie ein ganzes Bündel von Rasterfunktionen kombinieren können, um jene Kacheln zu erhalten, die ein bestimmtes WGS84 Umgebungsrechteck schneiden. Anschließend werden alle Bänder dieser Kacheln mit `ST_Union` vereinigt, mit `ST_Transform` in die vom Benutzer vorgegebene Projektion transformiert und das Ergebnis mit `ST_AsPNG` als PNG ausgegeben.

Sie können das unten angeführte Programm über

```
http://mywebserver/test_raster.php?srid=2249
```

aufrufen, um den Raster in "Massachusetts State Plane Feet" zu erhalten.

```
<?php
/** contents of test_raster.php */
$conn_str = 'dbname=mydb host=localhost port=5432 user=myuser password=mypwd';
$dbconn = pg_connect($conn_str);
header('Content-Type: image/png');
/**If a particular projection was requested use it otherwise use mass state plane meters ←
**/
if (!empty( $_REQUEST['srid'] ) &&& is_numeric( $_REQUEST['srid'] ) ){
    $input_srid = intval($_REQUEST['srid']);
}
else { $input_srid = 26986; }
/** The set bytea_output may be needed for PostgreSQL 9.0+, but not for 8.4 **/
$sql = "set bytea_output='escape';
SELECT ST_AsPNG(ST_Transform(
    ST_AddBand(ST_Union(rast,1), ARRAY[ST_Union(rast,2),ST_Union(rast ←
    ,3]))
    , $input_srid) ) As new_rast
FROM aerials.boston
WHERE
    ST_Intersects(rast, ST_Transform(ST_MakeEnvelope(-71.1217, 42.227, -71.1210, ←
    42.218,4326),26986) )";
$result = pg_query($sql);
$row = pg_fetch_row($result);
pg_free_result($result);
if ($row === false) return;
echo pg_unescape_bytea($row[0]);
?>
```

### 10.3.2 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 10.3.1 nur in C# implementiert.

Sie können das unten angeführte Programm über

```
http://mywebserver/TestRaster.ashx?srid=2249
```

aufrufen, um den Raster in "Massachusetts State Plane Feet" zu bekommen.

```
-- web.config connection string section --
<connectionStrings>
  <add name="DSN"
    connectionString="server=localhost;database=mydb;Port=5432;User Id=myuser;password= ←
    mypwd"/>
</connectionStrings>
```

```
// Code for TestRaster.ashx
<%@ WebHandler Language="C#" Class="TestRaster" %>
using System;
using System.Data;
using System.Web;
using Npgsql;

public class TestRaster : IHttpHandler
{
    public void ProcessRequest(HttpContext context)
    {
        context.Response.ContentType = "image/png";
        context.Response.BinaryWrite(GetResults(context));
    }

    public bool IsReusable {
        get { return false; }
    }

    public byte[] GetResults(HttpContext context)
    {
        byte[] result = null;
        NpgsqlCommand command;
        string sql = null;
        int input_srid = 26986;
    try {
        using (NpgsqlConnection conn = new NpgsqlConnection(System. ↵
            Configuration.ConfigurationManager.ConnectionStrings["DSN"]. ↵
            ConnectionString)) {
            conn.Open();

            if (context.Request["srid"] != null)
            {
                input_srid = Convert.ToInt32(context.Request["srid"]);
            }
            sql = @"SELECT ST_AsPNG(
                ST_Transform(
                    ST_AddBand(
                        ST_Union(rast,1), ARRAY[ST_Union(rast,2),ST_Union(rast,3)]
                            ,:input_srid) ) As new_rast
                FROM aerials.boston
                WHERE
                    ST_Intersects(rast,
                        ST_Transform(ST_MakeEnvelope(-71.1217, 42.227, ↵
                            -71.1210, 42.218,4326),26986) )";
            command = new NpgsqlCommand(sql, conn);
            command.Parameters.Add(new NpgsqlParameter("input_srid", input_srid));

            result = (byte[]) command.ExecuteScalar();
            conn.Close();
        }
    }
    catch (Exception ex)
    {
        result = null;
        context.Response.Write(ex.Message.Trim());
    }
}
```

```
        return result;
    }
}
```

### 10.3.3 Applikation für die Java-Konsole, welche eine Rasterabfrage als Bilddatei ausgibt

Eine einfache Java Applikation, die eine Abfrage entgegennimmt, ein Bild erzeugt und in eine bestimmte Datei ausgibt.

Sie können die neuesten PostgreSQL JDBC Treiber unter <http://jdbc.postgresql.org/download.html> herunterladen.

Sie können den unten angegebenen Code mit einem Befehl wie folgt kompilieren:

```
set env CLASSPATH ../../postgresql-9.0-801.jdbc4.jar
javac SaveQueryImage.java
jar cfm SaveQueryImage.jar Manifest.txt *.class
```

Und ihn von der Befehlszeile wie folgt aufrufen:

```
java -jar SaveQueryImage.jar "SELECT ST_AsPNG(ST_AsRaster(ST_Buffer(ST_Point(1,5),10, ' ←
quad_segs=2'),150, 150, '8BUI',100));" "test.png"
```

```
-- Manifest.txt --
Class-Path: postgresql-9.0-801.jdbc4.jar
Main-Class: SaveQueryImage
```

```
// Code for SaveQueryImage.java
import java.sql.Connection;
import java.sql.SQLException;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.io.*;

public class SaveQueryImage {
    public static void main(String[] argv) {
        System.out.println("Checking if Driver is registered with DriverManager.");

        try {
            //java.sql.DriverManager.registerDriver (new org.postgresql.Driver());
            Class.forName("org.postgresql.Driver");
        }
        catch (ClassNotFoundException cnfe) {
            System.out.println("Couldn't find the driver!");
            cnfe.printStackTrace();
            System.exit(1);
        }

        Connection conn = null;

        try {
            conn = DriverManager.getConnection("jdbc:postgresql://localhost:5432/mydb","myuser ←
", "mypwd");
            conn.setAutoCommit(false);

            PreparedStatement sGetImg = conn.prepareStatement(argv[0]);

            ResultSet rs = sGetImg.executeQuery();

            FileOutputStream fout;
            try
            {
```

```

        rs.next();
        /** Output to file name requested by user */
        fout = new FileOutputStream(new File(argv[1]) );
        fout.write(rs.getBytes(1));
        fout.close();
    }
    catch(Exception e)
    {
        System.out.println("Can't create file");
        e.printStackTrace();
    }

    rs.close();
    sGetImg.close();
    conn.close();
}
catch (SQLException se) {
    System.out.println("Couldn't connect: print out a stack trace and exit.");
    se.printStackTrace();
    System.exit(1);
}
}
}

```

### 10.3.4 Verwenden Sie PLPython um Bilder via SQL herauszuschreiben

Diese als plpython gespeicherte Prozedur erzeugt eine Datei pro Datensatz im Serververzeichnis. Benötigt die Installation von plpython. Funktioniert sowohl mit plpythonu als auch mit plpython3u.

```

CREATE OR REPLACE FUNCTION write_file (param_bytes bytea, param_filepath text)
RETURNS text
AS $$
f = open(param_filepath, 'wb+')
f.write(param_bytes)
return param_filepath
$$ LANGUAGE plpythonu;

```

```

--write out 5 images to the PostgreSQL server in varying sizes
-- note the postgresql daemon account needs to have write access to folder
-- this echos back the file names created;
SELECT write_file(ST_AsPNG(
    ST_AsRaster(ST_Buffer(ST_Point(1,5),j*5, 'quad_segs=2'),150*j, 150*j, '8BUI',100)),
    'C:/temp/slices'|| j || '.png')
FROM generate_series(1,5) As j;

```

```

write_file
-----

```

```

C:/temp/slices1.png
C:/temp/slices2.png
C:/temp/slices3.png
C:/temp/slices4.png
C:/temp/slices5.png

```

### 10.3.5 Faster mit PSQL ausgeben

Leider hat PSQL keine einfach zu benutzende Funktion für die Ausgabe von Binärdateien eingebaut. Dieser Hack baut auf der etwas veralteten "Large Object" Unterstützung von PostgreSQL auf. Um ihn anzuwenden verbinden Sie sich bitte zuerst über die Befehlszeile "psql" mit Ihrer Datenbank.

Anders als beim Ansatz mit Python, wird die Datei bei diesem Ansatz auf Ihrem lokalen Rechner erzeugt.

```
SELECT oid, lowrite(lo_open(oid, 131072), png) As num_bytes
FROM
( VALUES (lo_create(0),
  ST_AsPNG( (SELECT rast FROM aerials.boston WHERE rid=1) )
) ) As v(oid,png);
-- you'll get an output something like --
oid   | num_bytes
-----+-----
2630819 |      74860

-- next note the oid and do this replacing the c:/test.png to file path location
-- on your local computer
\lo_export 2630819 'C:/temp/aerial_samp.png'

-- this deletes the file from large object storage on db
SELECT lo_unlink(2630819);
```





## 11.1 Datentypen zur Unterstützung von Rastern.

### 11.1.1 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 Ausgabentyp in der Familie der Rasterüberlagerungsfunktionen um ein Rasterband in Polygone zu zerlegen.

#### Siehe auch

Section [13.6](#)

### 11.1.2 addbandarg

**addbandarg** — Ein zusammengesetzter Datentyp, der als Eingabewert für die Funktion "`ST_AddBand`" verwendet wird und sowohl Attribute als auch Initialwert des neuen Bandes festlegt.

#### 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\\_BandPixelT](#) 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](#)

### 11.1.3 rastbandarg

**rastbandarg** — Ein zusammengesetzter Datentyp, der verwendet wird um den Raster und, über einen Index, das Band des Rasters anzugeben.

#### Beschreibung

Ein zusammengesetzter Datentyp, der verwendet wird um den Raster und, über einen Index, das Band des Rasters anzugeben.

**rast raster** Besagter Raster

**nband integer** Der 1-basierte Wert, welcher das betreffende Rasterband kennzeichnet

**Siehe auch**

[ST\\_MapAlgebra \(callback function version\)](#)

**11.1.4 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.

**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**

Chapter [11](#)

**11.1.5 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](#)

### 11.1.6 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.

**Zählung ganzzahlig** Die Summe aller Zellen, die für eine zusammenfassende Statistik abgezählt wurden.

**Summe doppelte Genauigkeit** Die Summe aller abgezählten Zellwerte.

**mittlere doppelte Genauigkeit** Der arithmetische Mittelwert aller abgezählten Zellwerte.

**stddev doppelte Genauigkeit** Die Standardabweichung aller abgezählten Zellwerte.

**min doppelte Genauigkeit** Der Minimalwert aller abgezählten Zellwerte.

**max doppelte Genauigkeit** Der Maximalwert aller abgezählten Zellwerte.

**Siehe auch**

[ST\\_SummaryStats](#), [ST\\_SummaryStatsAgg](#)

### 11.1.7 unionarg

**unionarg** — Ein zusammengesetzter Datentyp, der als Eingabewert für die Funktion "[ST\\_Union](#)" dient. Dieser Datentyp bestimmt die zu behandelnden Bänder, sowie die Verhaltensweise des UNION Operators.

## 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](#)

## 11.2 Rastermanagement

### 11.2.1 AddRasterConstraints

**AddRasterConstraints** — Fügt die Raster-Constraints zu einer bestimmten Spalte einer bereits geladenen Rastertabelle hinzu. Diese Constraints beschränken das Koordinatentransformationssystem, den Maßstab, die Blockgröße, die Ausrichtung, die Bänder, den Bandtyp und eine Flag, die anzeigt ob die Rasterspalte regelmäßig geblockt ist. Es müssen bereits Daten in die Tabelle geladen sein, damit die Constraints abgeleitet werden können. Gibt TRUE zurück, wenn das Setzen der Constraints ausgeführt wurde; bei Problemen wird eine Meldung angezeigt.

## Synopsis

```
boolean AddRasterConstraints(name rasttable, name rastcolumn, boolean srid=true, boolean scale_x=true, boolean scale_y=true,
boolean blocksize_x=true, boolean blocksize_y=true, boolean same_alignment=true, boolean regular_blocking=false, boolean
num_bands=true , boolean pixel_types=true , boolean nodata_values=true , boolean out_db=true , boolean extent=true );
boolean AddRasterConstraints(name rasttable, name rastcolumn, text[] VARIADIC constraints);
boolean AddRasterConstraints(name rastschema, name rasttable, name rastcolumn, text[] VARIADIC constraints);
boolean AddRasterConstraints(name rastschema, name rasttable, name rastcolumn, boolean srid=true, boolean scale_x=true,
boolean scale_y=true, boolean blocksize_x=true, boolean blocksize_y=true, boolean same_alignment=true, boolean regular_blocking=f
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 [10.2.1](#) 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 Rasterquelle 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 Constraints auf die Spalte setzen**

```
CREATE TABLE myrasters(rid SERIAL primary key, rast raster);
INSERT INTO myrasters(rast)
SELECT ST_AddBand(ST_MakeEmptyRaster(1000, 1000, 0.3, -0.3, 2, 2, 0, 0,4326), 1, '8BSI':: ↵
  text, -129, NULL);

SELECT AddRasterConstraints('myrasters'::name, 'rast'::name);

-- verify if registered correctly in the raster_columns view --
SELECT srid, scale_x, scale_y, blocksize_x, blocksize_y, num_bands, pixel_types, ↵
  no_data_values
FROM raster_columns
WHERE r_table_name = 'myrasters';

srid | scale_x | scale_y | blocksize_x | blocksize_y | num_bands | pixel_types | ↵
-----+-----+-----+-----+-----+-----+-----+----- ↵
4326 |        2 |        2 |          1000 |          1000 |          1 | {8BSI}     | ↵
```

**Beispiele: Einen einzelnen Constraint setzen**

```
CREATE TABLE public.myrasters2(rid SERIAL primary key, rast raster);
INSERT INTO myrasters2(rast)
SELECT ST_AddBand(ST_MakeEmptyRaster(1000, 1000, 0.3, -0.3, 2, 2, 0, 0,4326), 1, '8BSI':: ↵
  text, -129, NULL);

SELECT AddRasterConstraints('public'::name, 'myrasters2'::name, 'rast'::name, ' ↵
  regular_blocking', 'blocksize');
-- get notice--
NOTICE: Adding regular blocking constraint
NOTICE: Adding blocksize-X constraint
NOTICE: Adding blocksize-Y constraint
```



**Siehe auch**[AddRasterConstraints](#)**11.2.3 AddOverviewConstraints**

AddOverviewConstraints — Eine Rasterspalte als Übersicht für eine andere Rasterspalte kennzeichnen.

**Synopsis**

boolean **AddOverviewConstraints**(name ovschema, name ovtable, name ovcolumn, name refschema, name reftable, name refcolumn, int ovfactor);

boolean **AddOverviewConstraints**(name ovtable, name ovcolumn, name reftable, name refcolumn, int ovfactor);

**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);

-- verify if registered correctly in the raster_overviews view --
SELECT o_table_name ot, o_raster_column oc,
       r_table_name rt, r_raster_column rc,
       overview_factor f
FROM raster_overviews WHERE o_table_name = 'res2';
 ot | oc | rt | rc | f
-----+-----+-----+-----+----
res2 | r2 | res1 | r1 | 2
(1 row)
```

**Siehe auch**

Section [10.2.2](#), [DropOverviewConstraints](#), [ST\\_CreateOverview](#), [AddRasterConstraints](#)

### 11.2.4 DropOverviewConstraints

DropOverviewConstraints — Löscht die Markierung einer Rasterspalte, die festlegt dass sie als Übersicht für eine andere Spalte dient.

#### Synopsis

```
boolean DropOverviewConstraints(name ovschema, name ovtable, name ovcolumn);
boolean DropOverviewConstraints(name ovtable, name ovcolumn);
```

#### Beschreibung

Jene Constraints einer Rasterspalte löschen, die sie als Übersicht einer anderen Rasterspalte in dem Rasterkatalog `raster_overview` 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 [10.2.2](#), [AddOverviewConstraints](#), [DropRasterConstraints](#)

### 11.2.5 PostGIS\_GDAL\_Version

PostGIS\_GDAL\_Version — Gibt die von PostGIS verwendete Version der GDAL-Bibliothek aus.

#### Synopsis

```
text PostGIS_GDAL_Version();
```

#### 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](#)

### 11.2.6 PostGIS\_Raster\_Lib\_Build\_Date

PostGIS\_Raster\_Lib\_Build\_Date — Gibt einen vollständigen Bericht aus, wann die Rasterbibliothek kompiliert wurde.



## Synopsis

```
text PostGIS_Raster_Lib_Build_Date();
```

## 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](#)

## 11.2.7 PostGIS\_Raster\_Lib\_Version

PostGIS\_Raster\_Lib\_Version — Gibt einen vollständigen Bericht über die Version und das Kompilationsdatum der Rasterbibliothek aus.

## Synopsis

```
text PostGIS_Raster_Lib_Version();
```

## 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](#)

## 11.2.8 ST\_GDALDrivers

ST\_GDALDrivers — Gibt eine Liste der Rasterformate aus, die von PostGIS über die Bibliothek GDAL unterstützt werden. Nur die Formate mit can\_write=True können von ST\_AsGDALRaster verwendet werden.

## Synopsis

setof record **ST\_GDALDrivers**(integer OUT idx, text OUT short\_name, text OUT long\_name, text OUT can\_read, text OUT can\_write, text OUT create\_options);

## Beschreibung

Gibt eine Liste der Rasterformate aus - short\_name, long\_name und die Optionen des Urhebers - die von GDAL unterstützt werden. Verwenden Sie den "short\_name" als Übergabewert für den Parameter format 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: 2.5.0 - die Spalten can\_read und can\_write hinzugefügt.

Änderung: 2.0.6, 2.1.3 - standardmäßig ist kein Treiber aktiviert, solange die GUC oder die Umgebungsvariable "gdal\_enabled\_drivers" nicht gesetzt sind.

Verfügbarkeit: 2.0.0 - GDAL >= 1.6.0.

## Beispiele: Treiber-Liste

```
SET postgis.gdal_enabled_drivers = 'ENABLE_ALL';
SELECT short_name, long_name, can_write
FROM st_gdaldrivers()
ORDER BY short_name;
```

short_name	long_name	can_write
AAIGrid	Arc/Info ASCII Grid	t
ACE2	ACE2	f
ADRG	ARC Digitized Raster Graphics	f
AIG	Arc/Info Binary Grid	f
AirSAR	AirSAR Polarimetric Image	f
ARG	Azavea Raster Grid format	t
BAG	Bathymetry Attributed Grid	f
BIGGIF	Graphics Interchange Format (.gif)	f
BLX	Magellan topo (.blx)	t
BMP	MS Windows Device Independent Bitmap	f
BSB	Maptech BSB Nautical Charts	f
PAux	PCI .aux Labelled	f
PCIDSK	PCIDSK Database File	f
PCRaster	PCRaster Raster File	f
PDF	Geospatial PDF	f
PDS	NASA Planetary Data System	f
PDS4	NASA Planetary Data System 4	t
PLMOAIC	Planet Labs Mosaics API	f
PLSCENES	Planet Labs Scenes API	f
PNG	Portable Network Graphics	t
PNM	Portable Pixmap Format (netpbm)	f
PRF	Racurs PHOTOMOD PRF	f
R	R Object Data Store	t
Rasterlite	Rasterlite	t
RDA	DigitalGlobe Raster Data Access driver	f
RIK	Swedish Grid RIK (.rik)	f
RMF	Raster Matrix Format	f
ROI_PAC	ROI_PAC raster	f
RPFTOC	Raster Product Format TOC format	f
RRASTER	R Raster	f
RS2	RadarSat 2 XML Product	f

RST	Idrisi Raster A.1	t
SAFE	Sentinel-1 SAR SAFE Product	f
SAGA	SAGA GIS Binary Grid (.sdat, .sg-grd-z)	t
SAR_CEOS	CEOS SAR Image	f
SDTS	SDTS Raster	f
SENTINEL2	Sentinel 2	f
SGI	SGI Image File Format 1.0	f
SNODAS	Snow Data Assimilation System	f
SRP	Standard Raster Product (ASRP/USRP)	f
SRTMHGT	SRTMHGT File Format	t
Terragen	Terragen heightfield	f
TIL	EarthWatch .TIL	f
TSX	TerraSAR-X Product	f
USGSDEM	USGS Optional ASCII DEM (and CDED)	t
VICAR	MIPL VICAR file	f
VRT	Virtual Raster	t
WCS	OGC Web Coverage Service	f
WMS	OGC Web Map Service	t
WMTS	OGC Web Map Tile Service	t
XPM	X11 PixMap Format	t
XYZ	ASCII Gridded XYZ	t
ZMap	ZMap Plus Grid	t

**Beispiele: Liste mit den Optionen für jeden Treiber**

```
-- Output the create options XML column of JPEG as a table --
-- Note you can use these creator options in ST_AsGDALRaster options argument
SELECT (xpath('@name', g.opt))[1]::text As oname,
       (xpath('@type', g.opt))[1]::text As otype,
       (xpath('@description', g.opt))[1]::text As descrip
FROM (SELECT unnest(xpath('/CreationOptionList/Option', create_options::xml)) As opt
FROM st_gdaldrivers()
WHERE short_name = 'JPEG') As g;
```

oname	otype	descrip
PROGRESSIVE	boolean	whether to generate a progressive JPEG
QUALITY	int	good=100, bad=0, default=75
WORLDFILE	boolean	whether to generate a worldfile
INTERNAL_MASK	boolean	whether to generate a validity mask
COMMENT	string	Comment
SOURCE_ICC_PROFILE	string	ICC profile encoded in Base64
EXIF_THUMBNAIL	boolean	whether to generate an EXIF thumbnail(overview). By default its max dimension will be 128
THUMBNAIL_WIDTH	int	Forced thumbnail width
THUMBNAIL_HEIGHT	int	Forced thumbnail height

(9 rows)

```
-- raw xml output for creator options for GeoTiff --
SELECT create_options
FROM st_gdaldrivers()
WHERE short_name = 'GTiff';

<CreationOptionList>
  <Option name="COMPRESS" type="string-select">
    <Value
>NONE</Value>
    <Value
>LZW</Value>
```

```

    <Value
>PACKBITS</Value>
    <Value
>JPEG</Value>
    <Value
>CCITTRLE</Value>
    <Value
>CCITTFAX3</Value>
    <Value
>CCITTFAX4</Value>
    <Value
>DEFLATE</Value>
  </Option>
  <Option name="PREDICTOR" type="int" description="Predictor Type"/>
  <Option name="JPEG_QUALITY" type="int" description="JPEG quality 1-100" default="75"/>
  <Option name="ZLEVEL" type="int" description="DEFLATE compression level 1-9" default ←
    ="6"/>
  <Option name="NBITS" type="int" description="BITS for sub-byte files (1-7), sub-uint16 ←
    (9-15), sub-uint32 (17-31)"/>
  <Option name="INTERLEAVE" type="string-select" default="PIXEL">
    <Value
>BAND</Value>
    <Value
>PIXEL</Value>
  </Option>
  <Option name="TILED" type="boolean" description="Switch to tiled format"/>
  <Option name="TFW" type="boolean" description="Write out world file"/>
  <Option name="RPB" type="boolean" description="Write out .RPB (RPC) file"/>
  <Option name="BLOCKXSIZE" type="int" description="Tile Width"/>
  <Option name="BLOCKYSIZE" type="int" description="Tile/Strip Height"/>
  <Option name="PHOTOMETRIC" type="string-select">
    <Value
>MINISBLACK</Value>
    <Value
>MINISWHITE</Value>
    <Value
>PALETTE</Value>
    <Value
>RGB</Value>
    <Value
>CMYK</Value>
    <Value
>YCBCR</Value>
    <Value
>CIELAB</Value>
    <Value
>ICCLAB</Value>
    <Value
>ITULAB</Value>
  </Option>
  <Option name="SPARSE_OK" type="boolean" description="Can newly created files have ←
    missing blocks?" default="FALSE"/>
  <Option name="ALPHA" type="boolean" description="Mark first extrasample as being alpha ←
    "/>
  <Option name="PROFILE" type="string-select" default="GDALGeoTIFF">
    <Value
>GDALGeoTIFF</Value>
    <Value
>GeoTIFF</Value>
    <Value
>BASELINE</Value>
  </Option>

```

```

    <Option name="PIXELTYPE" type="string-select">
      <Value
>DEFAULT</Value>
      <Value
>SIGNEDBYTE</Value>
    </Option>
    <Option name="BIGTIFF" type="string-select" description="Force creation of BigTIFF file ←
">
      <Value
>YES</Value>
      <Value
>NO</Value>
      <Value
>IF_NEEDED</Value>
      <Value
>IF_SAFER</Value>
    </Option>
    <Option name="ENDIANNESS" type="string-select" default="NATIVE" description="Force ←
endianness of created file. For DEBUG purpose mostly">
      <Value
>NATIVE</Value>
      <Value
>INVERTED</Value>
      <Value
>LITTLE</Value>
      <Value
>BIG</Value>
    </Option>
    <Option name="COPY_SRC_OVERVIEWS" type="boolean" default="NO" description="Force copy ←
of overviews of source dataset (CreateCopy())"/>
</CreationOptionList>

```

-- Output the create options XML column for GTiff as a table --

```

SELECT (xpath('@name', g.opt))[1]::text As oname,
       (xpath('@type', g.opt))[1]::text As otype,
       (xpath('@description', g.opt))[1]::text As descrip,
       array_to_string(xpath('Value/text()', g.opt), ', ') As vals
FROM (SELECT unnest(xpath('/CreationOptionList/Option', create_options::xml)) As opt
FROM st_gdaldrivers()
WHERE short_name = 'GTiff') As g;

```

oname	otype	descrip	vals
COMPRESS	string-select		NONE, LZW, ←
PACKBITS, JPEG, CCITTRLE, CCITTFAX3, CCITTFAX4, DEFLATE			
PREDICTOR	int	Predictor Type ←	
JPEG_QUALITY	int	JPEG quality 1-100 ←	
ZLEVEL	int	DEFLATE compression level 1-9 ←	
NBITS	int	BITS for sub-byte files (1-7), sub-uint16 (9-15), sub-uint32 (17-31) ←	
INTERLEAVE	string-select		BAND, PIXEL
TILED	boolean	Switch to tiled format ←	
TFW	boolean	Write out world file ←	

RPB	boolean	Write out .RPB (RPC) file ↔
BLOCKXSIZE	int	Tile Width ↔
BLOCKYSIZE	int	Tile/Strip Height ↔
PHOTOMETRIC	string-select	↔
		MINISBLACK, ↔
		MINISWHITE, PALETTE, RGB, CMYK, YCBCR, CIELAB, ICCLAB, ITULAB
SPARSE_OK	boolean	Can newly created files have missing blocks? ↔
ALPHA	boolean	Mark first extrasample as being alpha ↔
PROFILE	string-select	↔
		GDALGeoTIFF, ↔
		GeoTIFF, BASELINE
PIXELTYPE	string-select	↔
		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](#)

**11.2.9 ST\_Contour**

ST\_Contour — Erzeugt einen Satz von Vektorkonturen aus dem angegebenen Rasterband unter Verwendung des [GDAL-Konturierungsalgorithmus](#).

**Synopsis**

setof record **ST\_Contour**(raster rast, integer bandnumber=1, double precision level\_interval=100.0, double precision level\_base=0.0, double precision[] fixed\_levels=ARRAY[], boolean polygonize=false);

**Beschreibung**

Erzeugt eine Reihe von Vektorkonturen aus dem angegebenen Rasterband, unter Verwendung des [GDAL Konturierungsalgorithmus](#).

Wenn der Parameter `fixed_levels` ein nicht-leeres Array ist, werden die Parameter `level_interval` und `level_base` ignoriert.

Die Eingabeparameter sind:

**rast** Das Raster zur Erzeugung der Kontur von

**bandnumber** Das Band zur Erzeugung der Kontur von

**level\_interval** Der Höhenabstand zwischen den erzeugten Konturen

**level\_base** Die "Basis", auf die sich die Konturintervalle beziehen. Normalerweise ist dies Null, kann aber auch anders sein. Um 10m-Konturen bei 5, 15, 25, ... zu erzeugen, wäre die LEVEL\_BASE 5.

**fixed\_levels** Der Höhenabstand zwischen den erzeugten Konturen

**polygonize** Wenn `true`, werden Konturpolygone anstelle von Polygonlinien erstellt.

Die Rückgabewerte sind eine Reihe von Datensätzen mit den folgenden Attributen:

**geom** Die Geometrie der Konturlinie.

**id** Ein eindeutiger Bezeichner, der der Höhenlinie von GDAL zugewiesen wird.

**value** Der Rasterwert, den die Linie darstellt. Bei einer DEM-Höheneingabe wäre dies die Höhe der Ausgangskontur.

Verfügbarkeit: 3.2.0

### Beispiel

```
WITH c AS (
SELECT (ST_Contour(rast, 1, fixed_levels => ARRAY[100.0, 200.0, 300.0])).*
FROM dem_grid WHERE rid = 1
)
SELECT st_astext(geom), id, value
FROM c;
```

### Siehe auch

[ST\\_InterpolateRaster](#)

## 11.2.10 ST\_InterpolateRaster

**ST\_InterpolateRaster** — Interpoliert eine gerasterte Oberfläche auf der Grundlage eines Eingabesatzes von 3D-Punkten, wobei die X- und Y-Werte zur Positionierung der Punkte auf dem Gitter und der Z-Wert der Punkte als Oberflächenhöhe verwendet werden.

### Synopsis

```
raster ST_InterpolateRaster(geometry input_points, text algorithm_options, raster template, integer template_band_num=1);
```

### Beschreibung

Interpoliert eine gerasterte Oberfläche auf der Grundlage eines Eingabesatzes von 3D-Punkten, wobei die X- und Y-Werte zur Positionierung der Punkte auf dem Gitter und der Z-Wert der Punkte als Oberflächenhöhe verwendet werden. Es stehen fünf Interpolationsalgorithmen zur Verfügung: inverser Abstand, inverser Abstand nächster Nachbar, gleitender Durchschnitt, nächster Nachbar und lineare Interpolation. Weitere Einzelheiten zu den Algorithmen und ihren Parametern finden Sie in der [gdal\\_grid Dokumentation](#). Weitere Informationen darüber, wie Interpolationen berechnet werden, finden Sie im [GDAL-Grid-Tutorial](#).

Die Eingabeparameter sind:

**input\_points** Die Punkte, die die Interpolation steuern. Jede Geometrie mit Z-Werten ist akzeptabel, alle Punkte in der Eingabe werden verwendet.

**algorithm\_options** Eine Zeichenkette, die den Algorithmus und die Algorithmusoptionen definiert, im Format von [gdal\\_grid](#). Für eine Interpolation mit inversem Abstand und einer Glättung von 2 würde man zum Beispiel "invdist:smoothing=2.0" verwenden.

**template** Eine Rastervorlage zur Steuerung der Geometrie des Ausgaberrasters. Die Breite, Höhe, Pixelgröße, räumliche Ausdehnung und der Pixeltyp werden aus dieser Vorlage gelesen.

**template\_band\_num** Standardmäßig wird das erste Band im Vorlagenraster für die Steuerung des Ausgaberrasters verwendet, aber das kann mit diesem Parameter angepasst werden.

Verfügbarkeit: 3.2.0

### Beispiel

```
SELECT ST_InterpolateRaster(  
  'MULTIPOINT(10.5 9.5 1000, 11.5 8.5 1000, 10.5 8.5 500, 11.5 9.5 500)::geometry,  
  'invdist:smoothing:2.0',  
  ST_AddBand(ST_MakeEmptyRaster(200, 400, 10, 10, 0.01, -0.005, 0, 0), '16BSI')  
)
```

### Siehe auch

[ST\\_Contour](#)

## 11.2.11 UpdateRasterSRID

UpdateRasterSRID — Änderung der SRID aller Raster in der vom Anwender angegebenen Spalte und Tabelle.

### Synopsis

raster **UpdateRasterSRID**(name schema\_name, name table\_name, name column\_name, integer new\_srid);  
raster **UpdateRasterSRID**(name table\_name, name column\_name, integer new\_srid);

### Beschreibung

Änderung der SRID aller Raster in der vom Anwender angegebenen Spalte und Tabelle. Diese Funktion löscht alle entsprechenden Constraints (extent, alignment und die SRID) der Spalte bevor die SRID der Rasterspalte geändert wird.



#### Note

Die Daten des Rasters (Pixelwerte der Bänder) bleiben von dieser Funktion unangetastet. Es werden lediglich die Metadaten des Rasters geändert.

Verfügbarkeit: 2.1.0

### Siehe auch

[UpdateGeometrySRID](#)

## 11.2.12 ST\_CreateOverview

ST\_CreateOverview — Erzeugt eine Version des gegebenen Raster-Coverage mit geringerer Auflösung.



## Synopsis

```
regclass ST_CreateOverview(regclass tab, name col, int factor, text algo='NearestNeighbor');
```

## Beschreibung

Erzeugt eine Übersichtstabelle mit skalierten Kacheln der Ursprungstabelle. Die Ausgabekacheln haben dieselbe Größe und haben die gleiche räumliche Ausdehnung wie die Eingabekacheln mit einer niedrigeren Auflösung (die Pixelgröße ist in beiden Richtungen  $1/\text{factor}$  der Ursprünglichen).

Auf die Übersichtstabelle werden die Constraints gesetzt und sie steht über den Katalog `raster_oversiews` 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 10.2.2](#)

## 11.3 Raster Constructors

### 11.3.1 ST\_AddBand

**ST\_AddBand** — Gibt einen Raster mit den neu hinzugefügten Band(Bändern) aus. Der Typ , der Ausgangswert und der Index für den Speicherort des Bandes kann angegeben werden. Wenn kein Index angegeben ist, wird das Band am Ende hinzugefügt.

## Synopsis

- (1) raster **ST\_AddBand**(raster rast, addbandarg[] addbandargset);
- (2) raster **ST\_AddBand**(raster rast, integer index, text pixeltype, double precision initialvalue=0, double precision nodataval=NULL);
- (3) raster **ST\_AddBand**(raster rast, text pixeltype, double precision initialvalue=0, double precision nodataval=NULL);
- (4) raster **ST\_AddBand**(raster torast, raster fromrast, integer fromband=1, integer torastindex=at\_end);
- (5) raster **ST\_AddBand**(raster torast, raster[] fromrasts, integer fromband=1, integer torastindex=at\_end);
- (6) raster **ST\_AddBand**(raster rast, integer index, text outdbfile, integer[] outdbindex, double precision nodataval=NULL);
- (7) raster **ST\_AddBand**(raster rast, text outdbfile, integer[] outdbindex, integer index=at\_end, double precision nodataval=NULL);

## Beschreibung

Gibt ein Raster zurück, dem an der angegebenen Position (Index) ein neues Band mit dem angegebenen Typ, dem angegebenen Anfangswert und dem angegebenen Nodata-Wert hinzugefügt wurde. Wenn kein Index angegeben ist, wird das Band am Ende hinzugefügt. Wenn kein `vonband` angegeben ist, wird Band 1 angenommen. Der Pixeltyp ist eine String-Darstellung eines der in `ST_BandPixelType` angegebenen Pixeltypen. Wird ein vorhandener Index angegeben, werden alle nachfolgenden Bänder  $\geq$  dieser Index um 1 erhöht. Wird ein Anfangswert angegeben, der größer ist als der Maximalwert des Pixeltyps, wird der Anfangswert auf den höchsten vom Pixeltyp zugelassenen Wert gesetzt.

Bei der Variante, welche ein Feld von `addbandarg` entgegennimmt (Variante 1), ist ein bestimmter Indexwert von "addbandarg" auf den Raster zu dem Zeitpunkt bezogen, als das Band mit diesem "addbandarg" zum Raster hinzugefügt wurde. Siehe das Beispiel "Mehrere neue Bänder" unterhalb.

Bei der Variante, die ein Feld an Rastern (Variante 5) entgegennimmt, wird wenn `torast` NULL ist, das `fromband` Band eines jeden Rasters in diesem Feld, in einem neuen Raster akkumuliert.

Bei den Varianten, die ein `outdbfile` (Varianten 6 und 7) entgegennehmen, muss der Wert den vollständigen Pfad der Rasterdatei enthalten. Die Datei muss auch für die PostgreSQL-Instanz zugänglich sein.

Erweiterung: 2.1.0 - Unterstützung für "addbandarg" hinzugefügt.

Erweiterung: 2.1.0 Unterstützung für die neuen "out-db" Bänder hinzugefügt.

## Beispiele: Ein einzelnes, neues Band

```
-- Add another band of type 8 bit unsigned integer with pixels initialized to 200
UPDATE dummy_rast
  SET rast = ST_AddBand(rast,'8BUI'::text,200)
WHERE rid = 1;
```

```
-- Create an empty raster 100x100 units, with upper left right at 0, add 2 bands (band 1 ←
  is 0/1 boolean bit switch, band2 allows values 0-15)
-- uses addbandargs
INSERT INTO dummy_rast(rid,rast)
  VALUES(10, ST_AddBand(ST_MakeEmptyRaster(100, 100, 0, 0, 1, -1, 0, 0, 0),
    ARRAY[
      ROW(1, '1BB'::text, 0, NULL),
      ROW(2, '4BUI'::text, 0, NULL)
    ]::addbandarg[]
  )
);
```

```
-- output meta data of raster bands to verify all is right --
SELECT (bmd).*
FROM (SELECT ST_BandMetaData(rast,generate_series(1,2)) As bmd
  FROM dummy_rast WHERE rid = 10) AS foo;
```

```
--result --
pixeltype | nodatavalue | isoutdb | path
-----+-----+-----+-----
1BB      |              | f       |
4BUI     |              | f       |
```

```
-- output meta data of raster -
SELECT (rmd).width, (rmd).height, (rmd).numbands
FROM (SELECT ST_MetaData(rast) As rmd
  FROM dummy_rast WHERE rid = 10) AS foo;
```

```
-- result --
upperleftx | upperlefty | width | height | scalex | scaley | skewx | skewy | srid | ←
numbands
-----+-----+-----+-----+-----+-----+-----+-----+-----+-----
```

```
0 | 0 | 100 | 100 | 1 | -1 | 0 | 0 | 0 | ←
2
```

### Beispiele: Mehrere neue Bänder

```
SELECT
*
FROM ST_BandMetadata(
  ST_AddBand(
    ST_MakeEmptyRaster(10, 10, 0, 0, 1, -1, 0, 0, 0),
    ARRAY[
      ROW(NULL, '8BUI', 255, 0),
      ROW(NULL, '16BUI', 1, 2),
      ROW(2, '32BUI', 100, 12),
      ROW(2, '32BF', 3.14, -1)
    ]::addbandarg[]
  ),
  ARRAY[]::integer[]
);
```

bandnum	pixeltype	nodatavalue	isoutdb	path
1	8BUI	0	f	
2	32BF	-1	f	
3	32BUI	12	f	
4	16BUI	2	f	

```
-- Aggregate the 1st band of a table of like rasters into a single raster
-- with as many bands as there are test_types and as many rows (new rasters) as there are ←
-- mice
-- NOTE: The ORDER BY test_type is only supported in PostgreSQL 9.0+
-- for 8.4 and below it usually works to order your data in a subselect (but not guaranteed ←
-- )
-- The resulting raster will have a band for each test_type alphabetical by test_type
-- For mouse lovers: No mice were harmed in this exercise
SELECT
  mouse,
  ST_AddBand(NULL, array_agg(rast ORDER BY test_type), 1) As rast
FROM mice_studies
GROUP BY mouse;
```

### Beispiele: Neues Out-db Band

```
SELECT
*
FROM ST_BandMetadata(
  ST_AddBand(
    ST_MakeEmptyRaster(10, 10, 0, 0, 1, -1, 0, 0, 0),
    '/home/raster/mytestraster.tif'::text, NULL::int[]
  ),
  ARRAY[]::integer[]
);
```

bandnum	pixeltype	nodatavalue	isoutdb	path
1	8BUI		t	/home/raster/mytestraster.tif
2	8BUI		t	/home/raster/mytestraster.tif
3	8BUI		t	/home/raster/mytestraster.tif

**Siehe auch**

[ST\\_BandMetaData](#), [ST\\_BandPixelType](#), [ST\\_MakeEmptyRaster](#), [ST\\_MetaData](#), [ST\\_NumBands](#), [ST\\_Reclass](#)

**11.3.2 ST\_AsRaster**

ST\_AsRaster — Konvertiert den geometrischen Datentyp von PostGIS in einen PostGIS Raster.

**Synopsis**

```
raster ST_AsRaster(geometry geom, raster ref, text pixeltype, double precision value=1, double precision nodataval=0, boolean touched=false);
raster ST_AsRaster(geometry geom, raster ref, text[] pixeltype=ARRAY['8BUI'], double precision[] value=ARRAY[1], double precision[] nodataval=ARRAY[0], boolean touched=false);
raster ST_AsRaster(geometry geom, double precision scalex, double precision scaley, double precision gridx, double precision gridy, text pixeltype, double precision value=1, double precision nodataval=0, double precision skewx=0, double precision skewy=0, boolean touched=false);
raster ST_AsRaster(geometry geom, double precision scalex, double precision scaley, double precision gridx=NULL, double precision gridy=NULL, text[] pixeltype=ARRAY['8BUI'], double precision[] value=ARRAY[1], double precision[] nodataval=ARRAY[0], double precision skewx=0, double precision skewy=0, boolean touched=false);
raster ST_AsRaster(geometry geom, double precision scalex, double precision scaley, text pixeltype, double precision value=1, double precision nodataval=0, double precision upperleftx=NULL, double precision upperlefty=NULL, double precision skewx=0, double precision skewy=0, boolean touched=false);
raster ST_AsRaster(geometry geom, double precision scalex, double precision scaley, text[] pixeltype, double precision[] value=ARRAY[1], double precision[] nodataval=ARRAY[0], double precision upperleftx=NULL, double precision upperlefty=NULL, double precision skewx=0, double precision skewy=0, boolean touched=false);
raster ST_AsRaster(geometry geom, integer width, integer height, double precision gridx, double precision gridy, text pixeltype, double precision value=1, double precision nodataval=0, double precision skewx=0, double precision skewy=0, boolean touched=false);
raster ST_AsRaster(geometry geom, integer width, integer height, double precision gridx=NULL, double precision gridy=NULL, text[] pixeltype=ARRAY['8BUI'], double precision[] value=ARRAY[1], double precision[] nodataval=ARRAY[0], double precision skewx=0, double precision skewy=0, boolean touched=false);
raster ST_AsRaster(geometry geom, integer width, integer height, text pixeltype, double precision value=1, double precision nodataval=0, double precision upperleftx=NULL, double precision upperlefty=NULL, double precision skewx=0, double precision skewy=0, boolean touched=false);
raster ST_AsRaster(geometry geom, integer width, integer height, text[] pixeltype, double precision[] value=ARRAY[1], double precision[] nodataval=ARRAY[0], double precision upperleftx=NULL, double precision upperlefty=NULL, double precision skewx=0, double precision skewy=0, boolean touched=false);
```

**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 dritte 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.

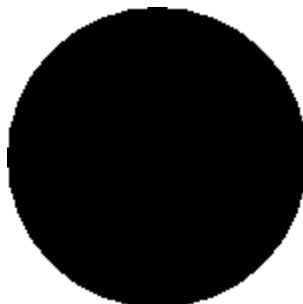
Der optionale Parameter `touched` ist standardmäßig `FALSE` und wird auf die GDAL Rasterungsoption `ALL_TOUCHED` abgebildet, welche bestimmt, ob Pixel die von Linien oder Polygonen nur berührt werden, ebenfalls gerastert werden sollen; und nicht nur die Pixel auf einem Linienzug oder mit dem Mittelpunkt 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*

```
-- this will output a black circle taking up 150 x 150 pixels --  
SELECT ST_AsPNG(ST_AsRaster(ST_Buffer(ST_Point(1,5),10),150, 150));
```



*Ein Beispiel für einen Puffer; die Grafik ist direkt in PostGIS erstellt*

```
-- the bands map to RGB bands - the value (118,154,118) - teal --
SELECT ST_AsPNG(
  ST_AsRaster(
    ST_Buffer(
      ST_GeomFromText('LINESTRING(50 50,150 150,150 50)'), 10,'join=bevel'),
      200,200,ARRAY['8BUI', '8BUI', '8BUI'], ARRAY[118,154,118], ARRAY[0,0,0]));
```

## Siehe auch

[ST\\_BandPixelType](#), [ST\\_Buffer](#), [ST\\_GDALDrivers](#), [ST\\_AsGDALRaster](#), [ST\\_AsPNG](#), [ST\\_AsJPEG](#), [ST\\_SRID](#)

### 11.3.3 ST\_Band

**ST\_Band** — Gibt einen oder mehrere Bänder eines bestehenden Rasters als neuen Raster aus. Nützlich um neue Raster aus bestehenden Rastern abzuleiten.

#### Synopsis

```
raster ST_Band(raster rast, integer[] nbands = ARRAY[1]);
raster ST_Band(raster rast, integer nband);
raster ST_Band(raster rast, text nbands, character delimiter=,);
```

#### Beschreibung

Gibt einen oder mehrere Bänder eines bestehenden Rasters in Form eines neuen Raster aus. Nützlich um neue Raster aus bestehenden Rastern abzuleiten, um einen Export auf ausgewählte Bänder einzuschränken, oder um die Reihenfolge der Rasterbänder neu zu ordnen. Wenn kein Band angegeben ist, oder keines der spezifizierten Bänder im Raster existiert, dann werden alle Bänder zurückgegeben. Dient auch als Hilfsfunktion für verschiedene Funktionen, wie das Löschen eines Bandes.



#### Warning

Bei der Funktionsvariante mit `nbands` als Text, ist das Standardtrennzeichen `,`; d.h.: Sie können `'1,2,3'` abfragen und falls Sie ein anderes Trennzeichen verwenden wollen `ST_Band(rast, '1@2@3', '@')`. Wenn Sie mehrere Bänder abfragen, empfehlen wir Ihnen unbedingt die Feldvariante der Funktion zu verwenden, z.B. `ST_Band(rast, '{1,2,3}'::int[])`; da die Variante mit der `text` Liste der Bänder in zukünftigen Versionen von PostGIS entfernt werden könnte.

Verfügbarkeit: 2.0.0

#### Beispiele

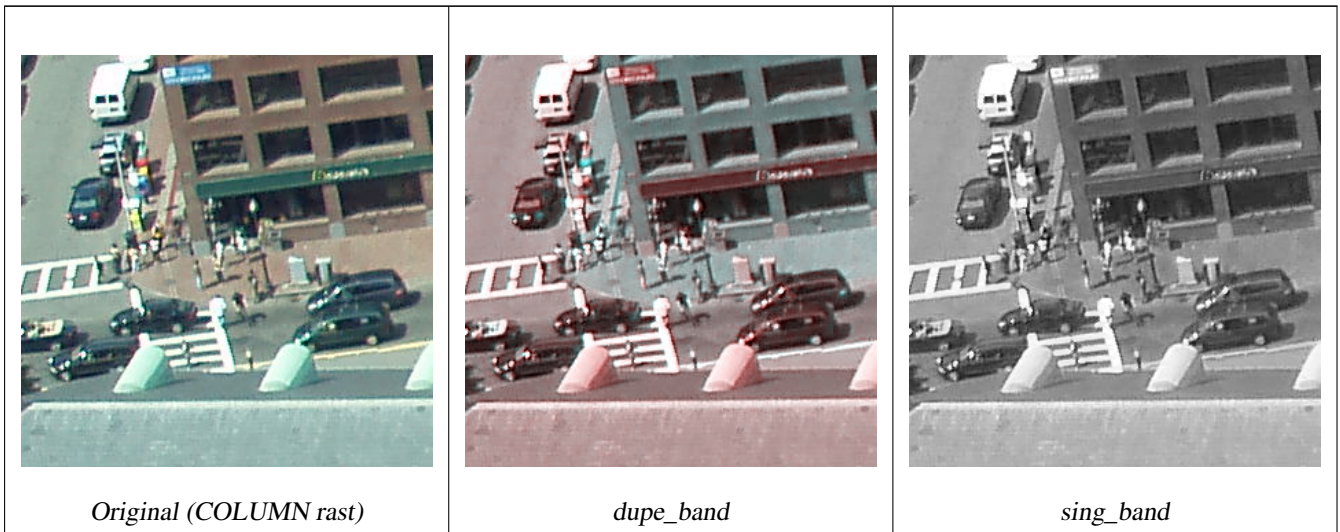
```
-- Make 2 new rasters: 1 containing band 1 of dummy, second containing band 2 of dummy and ←
  then reclassified as a 2BUI
SELECT ST_NumBands(rast1) As numb1, ST_BandPixelType(rast1) As pix1,
  ST_NumBands(rast2) As numb2, ST_BandPixelType(rast2) As pix2
FROM (
  SELECT ST_Band(rast) As rast1, ST_Reclass(ST_Band(rast,3), '100-200):1, [200-254:2', '2 ←
    BUI') As rast2
  FROM dummy_rast
  WHERE rid = 2) As foo;
```

```
numb1 | pix1 | numb2 | pix2
-----+-----+-----+-----
1 | 8BUI | 1 | 2BUI
```

```
-- Return bands 2 and 3. Using array cast syntax
SELECT ST_NumBands(ST_Band(rast, '{2,3}'::int[])) As num_bands
FROM dummy_rast WHERE rid=2;
```

```
num_bands
-----
2
```

```
-- Return bands 2 and 3. Use array to define bands
SELECT ST_NumBands(ST_Band(rast, ARRAY[2,3])) As num_bands
FROM dummy_rast
WHERE rid=2;
```



```
--Make a new raster with 2nd band of original and 1st band repeated twice,
and another with just the third band
SELECT rast, ST_Band(rast, ARRAY[2,1,1]) As dupe_band,
ST_Band(rast, 3) As sing_band
FROM samples.than_chunked
WHERE rid=35;
```

## Siehe auch

[ST\\_AddBand](#), [ST\\_NumBands](#), [ST\\_Reclass](#), [Chapter 11](#)

### 11.3.4 ST\_MakeEmptyCoverage

`ST_MakeEmptyCoverage` — Bedeckt die georeferenzierte Fläche mit einem Gitter aus leeren Rasterkacheln.

#### Synopsis

raster `ST_MakeEmptyCoverage`(integer tilewidth, integer tileheight, integer width, integer height, double precision upperleftx, double precision upperlefty, double precision scalex, double precision scaley, double precision skewx, double precision skewy, integer srid=unknown);

### Beschreibung

Erzeugt Rasterkacheln mit **ST\_MakeEmptyRaster**. Die Größe des Gitters wird über `width` & `height` angegeben. Die Kachelgröße über `tilewidth` & `tileheight`. Die abgedeckte georeferenzierte Fläche reicht vom oberen linken Eck (`upperleftx`, `upperlefty`) bis zum unteren rechten Eck (`upperleftx + width * scalex`, `upperlefty + height * scaley`).



**Note**

Beachten Sie bitte, dass bei Rastern "scaley" im Allgemeinen negativ und "scalex" positiv ist. Dadurch hat das untere rechte Eck einen niedrigeren Y-Wert und einen höheren X-Wert als die obere rechte Ecke.

Verfügbarkeit: 2.4.0

### Grundlegende Beispiele

Erzeugt ein 4x4 Gitter aus 16 Kacheln, um die WGS84 Fläche vom linken oberen Eck (22, 77) zum rechten unteren Eck (55, 33) abzudecken.

```
SELECT (ST_MetaData(tile)).* FROM ST_MakeEmptyCoverage(1, 1, 4, 4, 22, 33, (55 - 22)/(4)::float, (33 - 77)/(4)::float, 0., 0., 4326) tile;
```

upperleftx	upperlefty	width	height	scalex	scaley	skewx	skewy	srid	numbands
22	33	1	1	8.25	-11	0	0	4326	1
30.25	33	1	1	8.25	-11	0	0	4326	1
38.5	33	1	1	8.25	-11	0	0	4326	1
46.75	33	1	1	8.25	-11	0	0	4326	1
22	22	1	1	8.25	-11	0	0	4326	1
30.25	22	1	1	8.25	-11	0	0	4326	1
38.5	22	1	1	8.25	-11	0	0	4326	1
46.75	22	1	1	8.25	-11	0	0	4326	1
22	11	1	1	8.25	-11	0	0	4326	1
30.25	11	1	1	8.25	-11	0	0	4326	1
38.5	11	1	1	8.25	-11	0	0	4326	1
46.75	11	1	1	8.25	-11	0	0	4326	1
22	0	1	1	8.25	-11	0	0	4326	1
30.25	0	1	1	8.25	-11	0	0	4326	1
38.5	0	1	1	8.25	-11	0	0	4326	1
46.75	0	1	1	8.25	-11	0	0	4326	1





```
4 | 0.0005 | 0.0005 | 100 | 100 | 1 | 1 | 0 | 0 | ←
   | 4326 | 0
```

## Siehe auch

[ST\\_AddBand](#), [ST\\_MetaData](#), [ST\\_ScaleX](#), [ST\\_ScaleY](#), [ST\\_SetValue](#), [ST\\_SkewX](#), [ST\\_SkewY](#)

## 11.3.6 ST\_Tile

`ST_Tile` — Gibt Raster, die aus einer Teilungsoption 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 Teilungsoption 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, '8BUI', ←
    1, 0), 2, '8BUI', 10, 0) AS rast UNION ALL
  SELECT ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(3, 3, 3, 0, 1, -1, 0, 0, 0), 1, '8BUI', ←
    2, 0), 2, '8BUI', 20, 0) AS rast UNION ALL
  SELECT ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(3, 3, 6, 0, 1, -1, 0, 0, 0), 1, '8BUI', ←
    3, 0), 2, '8BUI', 30, 0) AS rast UNION ALL

  SELECT ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(3, 3, 0, -3, 1, -1, 0, 0, 0), 1, '8BUI' ←
    ', 4, 0), 2, '8BUI', 40, 0) AS rast UNION ALL
  SELECT ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(3, 3, 3, -3, 1, -1, 0, 0, 0), 1, '8BUI' ←
    ', 5, 0), 2, '8BUI', 50, 0) AS rast UNION ALL
  SELECT ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(3, 3, 6, -3, 1, -1, 0, 0, 0), 1, '8BUI' ←
    ', 6, 0), 2, '8BUI', 60, 0) AS rast UNION ALL

  SELECT ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(3, 3, 0, -6, 1, -1, 0, 0, 0), 1, '8BUI' ←
    ', 7, 0), 2, '8BUI', 70, 0) AS rast UNION ALL
```

```

SELECT ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(3, 3, 3, -6, 1, -1, 0, 0, 0), 1, '8BUI' ←
', 8, 0), 2, '8BUI', 80, 0) AS rast UNION ALL
SELECT ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(3, 3, 6, -6, 1, -1, 0, 0, 0), 1, '8BUI' ←
', 9, 0), 2, '8BUI', 90, 0) AS rast
), bar AS (
SELECT ST_Union(rast) AS rast FROM foo
), baz AS (
SELECT ST_Tile(rast, 3, 3, TRUE) AS rast FROM bar
)
SELECT
ST_DumpValues(rast)
FROM baz;

```

st\_dumpvalues

```

-----
(1,"{{1,1,1},{1,1,1},{1,1,1}}")
(2,"{{10,10,10},{10,10,10},{10,10,10}}")
(1,"{{2,2,2},{2,2,2},{2,2,2}}")
(2,"{{20,20,20},{20,20,20},{20,20,20}}")
(1,"{{3,3,3},{3,3,3},{3,3,3}}")
(2,"{{30,30,30},{30,30,30},{30,30,30}}")
(1,"{{4,4,4},{4,4,4},{4,4,4}}")
(2,"{{40,40,40},{40,40,40},{40,40,40}}")
(1,"{{5,5,5},{5,5,5},{5,5,5}}")
(2,"{{50,50,50},{50,50,50},{50,50,50}}")
(1,"{{6,6,6},{6,6,6},{6,6,6}}")
(2,"{{60,60,60},{60,60,60},{60,60,60}}")
(1,"{{7,7,7},{7,7,7},{7,7,7}}")
(2,"{{70,70,70},{70,70,70},{70,70,70}}")
(1,"{{8,8,8},{8,8,8},{8,8,8}}")
(2,"{{80,80,80},{80,80,80},{80,80,80}}")
(1,"{{9,9,9},{9,9,9},{9,9,9}}")
(2,"{{90,90,90},{90,90,90},{90,90,90}}")
(18 rows)

```

```

WITH foo AS (
SELECT ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(3, 3, 0, 0, 1, -1, 0, 0, 0), 1, '8BUI', ←
1, 0), 2, '8BUI', 10, 0) AS rast UNION ALL
SELECT ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(3, 3, 3, 0, 1, -1, 0, 0, 0), 1, '8BUI', ←
2, 0), 2, '8BUI', 20, 0) AS rast UNION ALL
SELECT ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(3, 3, 6, 0, 1, -1, 0, 0, 0), 1, '8BUI', ←
3, 0), 2, '8BUI', 30, 0) AS rast UNION ALL

SELECT ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(3, 3, 0, -3, 1, -1, 0, 0, 0), 1, '8BUI' ←
', 4, 0), 2, '8BUI', 40, 0) AS rast UNION ALL
SELECT ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(3, 3, 3, -3, 1, -1, 0, 0, 0), 1, '8BUI' ←
', 5, 0), 2, '8BUI', 50, 0) AS rast UNION ALL
SELECT ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(3, 3, 6, -3, 1, -1, 0, 0, 0), 1, '8BUI' ←
', 6, 0), 2, '8BUI', 60, 0) AS rast UNION ALL

SELECT ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(3, 3, 0, -6, 1, -1, 0, 0, 0), 1, '8BUI' ←
', 7, 0), 2, '8BUI', 70, 0) AS rast UNION ALL
SELECT ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(3, 3, 3, -6, 1, -1, 0, 0, 0), 1, '8BUI' ←
', 8, 0), 2, '8BUI', 80, 0) AS rast UNION ALL
SELECT ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(3, 3, 6, -6, 1, -1, 0, 0, 0), 1, '8BUI' ←
', 9, 0), 2, '8BUI', 90, 0) AS rast
), bar AS (
SELECT ST_Union(rast) AS rast FROM foo
), baz AS (
SELECT ST_Tile(rast, 3, 3, 2) AS rast FROM bar
)
SELECT

```

```
ST_DumpValues(rast)
FROM baz;

          st_dumpvalues
-----
(1, "{10,10,10},{10,10,10},{10,10,10}")
(1, "{20,20,20},{20,20,20},{20,20,20}")
(1, "{30,30,30},{30,30,30},{30,30,30}")
(1, "{40,40,40},{40,40,40},{40,40,40}")
(1, "{50,50,50},{50,50,50},{50,50,50}")
(1, "{60,60,60},{60,60,60},{60,60,60}")
(1, "{70,70,70},{70,70,70},{70,70,70}")
(1, "{80,80,80},{80,80,80},{80,80,80}")
(1, "{90,90,90},{90,90,90},{90,90,90}")
(9 rows)
```

#### Siehe auch

[ST\\_Union](#), [ST\\_Retile](#)

### 11.3.7 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](#)

### 11.3.8 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](#)

## 11.4 Zugriffsfunktionen auf Raster

### 11.4.1 ST\_GeoReference

`ST_GeoReference` — Gibt die Metadaten der Georeferenzierung, die sich üblicherweise in einem sogenannten "World File" befinden, im GDAL oder ESRI Format aus. Die Standardeinstellung ist GDAL.

## Synopsis

text `ST_GeoReference`(raster rast, text format=GDAL);

## 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](#)

### 11.4.2 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](#)

**11.4.3 ST\_IsEmpty**

**ST\_IsEmpty** — Gibt TRUE zurück, wenn der Raster leer ist (width = 0 and height = 0). Andernfalls wird FALSE zurückgegeben.

**Synopsis**

boolean **ST\_IsEmpty**(raster rast);

**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](#)

**11.4.4 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

### 11.4.5 ST\_MetaData

`ST_MetaData` — Gibt die wesentlichen Metadaten eines Rasterobjektes, wie Zellgröße, Rotation (Versatz) etc. aus

## Synopsis

```
record ST_MetaData(raster rast);
```

## Beschreibung

Gibt die grundlegenden Metadaten eines Rasters aus, wie Pixelgröße, Rotation (skew), obere, untere linke, etc.. Die zurückgegebenen Spalten sind: `upperleftx` | `upperlefty` | `width` | `height` | `scalex` | `scaley` | `skewx` | `skewy` | `srid` | `numbands`

## Beispiele

```
SELECT rid, (foo.md).*
  FROM (SELECT rid, ST_MetaData(rast) As md
  FROM dummy_rast) As foo;
```

rid	upperleftx	upperlefty	width	height	scalex	scaley	skewx	skewy	srid	numbands
1	0.5	0.5	10	20	2	3	0	0	0	0
2	3427927.75	5793244	5	5	0.05	-0.05	0	0	0	3



**Siehe auch**

[ST\\_BandMetaData](#), [ST\\_NumBands](#)

**11.4.6 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](#)

**11.4.7 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](#)

**11.4.8 ST\_PixelWidth**

`ST_PixelWidth` — Gibt die Pixelbreite in den Einheiten des Koordinatenreferenzsystems aus.

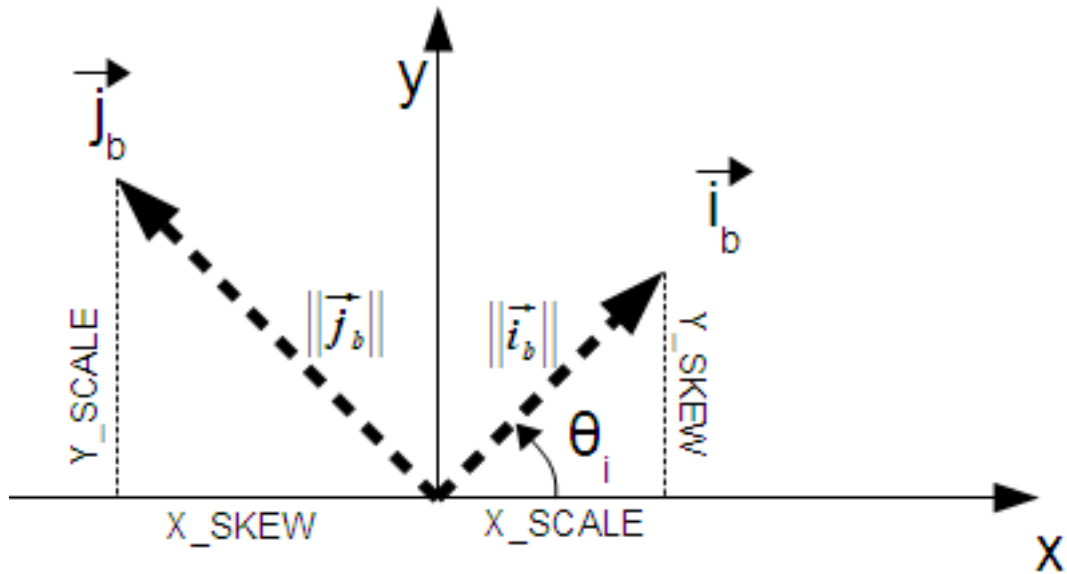
**Synopsis**

```
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](#)

**11.4.9 ST\_ScaleX**

ST\_ScaleX — Gibt die X-Komponente der Pixelbreite in den Einheiten des Koordinatenreferenzsystems aus.

## Synopsis

```
float8 ST_ScaleX(raster rast);
```

## 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](#)

### 11.4.10 ST\_ScaleY

`ST_ScaleY` — Gibt die Y-Komponente der Pixelhöhe in den Einheiten des Koordinatenreferenzsystems aus.

## Synopsis

```
float8 ST_ScaleY(raster rast);
```

## 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](#)

### 11.4.11 ST\_RasterToWorldCoord

`ST_RasterToWorldCoord` — Gibt die obere linke Ecke des Rasters in geodätischem X und Y (Länge und Breite) für eine gegebene Spalte und Zeile aus. Spalte und Zeile wird von 1 aufwärts gezählt.

#### Synopsis

```
record ST_RasterToWorldCoord(raster rast, integer xcolumn, integer yrow);
```

#### Beschreibung

Gibt die obere linke Ecke einer Spalte und Zeile als geometrisches X und Y (als geographische Länge und Breite) aus. Die zurückgegebenen X und Y sind in den Einheiten des georeferenzierten Rasters. Die Nummerierung der Spalten und Zeilen beginnt mit 1. Allerdings, wenn für einen der Parameter 0, eine negative Zahl, oder eine höhere Zahl als die betreffende Größe des Rasters übergeben wird, dann werden Koordinaten außerhalb des Rasters ausgegeben; dabei wird angenommen, dass das Gitter des Rasters auch außerhalb der Begrenzung des Rasters angewendet werden kann.

Verfügbarkeit: 2.1.0

#### Beispiele

```
-- non-skewed raster
SELECT
    rid,
    (ST_RasterToWorldCoord(rast,1, 1)).*,
    (ST_RasterToWorldCoord(rast,2, 2)).*
FROM dummy_rast
```

rid	longitude	latitude	longitude	latitude
1	0.5	0.5	2.5	3.5
2	3427927.75	5793244	3427927.8	5793243.95

```
-- skewed raster
SELECT
    rid,
    (ST_RasterToWorldCoord(rast, 1, 1)).*,
    (ST_RasterToWorldCoord(rast, 2, 3)).*
FROM (
    SELECT
        rid,
        ST_SetSkew(rast, 100.5, 0) As rast
    FROM dummy_rast
) As foo
```

rid	longitude	latitude	longitude	latitude
1	0.5	0.5	203.5	6.5
2	3427927.75	5793244	3428128.8	5793243.9

#### Siehe auch

[ST\\_RasterToWorldCoordX](#), [ST\\_RasterToWorldCoordY](#), [ST\\_SetSkew](#)

### 11.4.12 ST\_RasterToWorldCoordX

`ST_RasterToWorldCoordX` — Gibt die geodätische X Koordinate links oberhalb des Rasters, der Spalte und der Zeile aus. Die Nummerierung der Spalten und Zeilen beginnt mit 1.

#### Synopsis

```
float8 ST_RasterToWorldCoordX(raster rast, integer xcolumn);
float8 ST_RasterToWorldCoordX(raster rast, integer xcolumn, integer yrow);
```

#### Beschreibung

Gibt die obere linke X-Koordinate einer Rasterzeile in den geometrischen Einheiten des georeferenzierten Rasters aus. Die Nummerierung der Spalten und Zeilen beginnt mit 1. Wenn Sie eine negative Zahl oder eine höhere Zahl als die Anzahl der Spalten des Rasters angeben, dann bekommen Sie die Koordinaten links außerhalb und rechts außerhalb des Rasters zurück; dabei wird angenommen, dass der Versatz und die Pixelgröße gleich wie bei dem ausgewählten Raster sind.



#### Note

Bei Rastern ohne Versatz, ist die Angabe der X-Spalte ausreichend. Bei Rastern mit Versatz ist die georeferenzierte Koordinate eine Funktion von "ST\_ScaleX", "ST\_SkewX", der Zeile und der Spalte. Wenn Sie bei einem Raster mit Versatz nur die X-Spalte angeben, dann wird eine Fehlermeldung ausgegeben.

Änderung: 2.1.0 Vorgängerversionen haben dies als "ST\_Raster2WorldCoordX" bezeichnet.

#### Beispiele

```
-- non-skewed raster providing column is sufficient
SELECT rid, ST_RasterToWorldCoordX(rast,1) As 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

```
-- for fun lets skew it
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](#)

### 11.4.13 ST\_RasterToWorldCoordY

ST\_RasterToWorldCoordY — Gibt die geodätische Y Koordinate links oberhalb des Rasters, der Spalte und der Zeile aus. Die Nummerierung der Spalten und Zeilen beginnt mit 1.

#### Synopsis

```
float8 ST_RasterToWorldCoordY(raster rast, integer yrow);
float8 ST_RasterToWorldCoordY(raster rast, integer xcolumn, integer yrow);
```

#### Beschreibung

Gibt die obere linke Y-Koordinate einer RasterSpalte in den Einheiten des georeferenzierten Rasters aus. Die Nummerierung der Spalten und Zeilen beginnt mit 1. Wenn Sie eine negative Zahl oder eine höhere Zahl als die Anzahl der Spalten/Zeilen des Rasters angeben, dann bekommen Sie die Koordinaten außerhalb des Rasters zurück; dabei wird angenommen, dass Versatz und Pixelgröße gleich wie bei der ausgewählten Rasterkachel sind.



#### Note

Bei Rastern ohne Versatz ist die Angabe der Y-Spalte ausreichend. Bei Rastern mit Versatz entspricht die georeferenzierte Koordinate einer Funktion von ST\_ScaleY, ST\_SkewY, Zeile und Spalte. Wenn Sie bei einem Raster mit Versatz nur die Y-Spalte angeben, wird eine Fehlermeldung ausgegeben.

Änderung: 2.1.0 In Vorgängerversionen wurde dies als ST\_Raster2WorldCoordY bezeichnet

#### Beispiele

```
-- non-skewed raster providing row is sufficient
SELECT rid, ST_RasterToWorldCoordY(rast,1) As ylcoord,
       ST_RasterToWorldCoordY(rast,3) As y2coord,
       ST_ScaleY(rast) As pixely
FROM dummy_rast;
```

rid	ylcoord	y2coord	pixely
1	0.5	6.5	3
2	5793244	5793243.9	-0.05

```
-- for fun lets skew it
SELECT rid, ST_RasterToWorldCoordY(rast,1,1) As ylcoord,
       ST_RasterToWorldCoordY(rast,2,3) As y2coord,
       ST_ScaleY(rast) As pixely
FROM (SELECT rid, ST_SetSkew(rast,0,100.5) As rast FROM dummy_rast) As foo;
```

rid	ylcoord	y2coord	pixely
1	0.5	107	3
2	5793244	5793344.4	-0.05

#### Siehe auch

[ST\\_ScaleY](#), [ST\\_RasterToWorldCoordX](#), [ST\\_SetSkew](#), [ST\\_SkewY](#)

### 11.4.14 ST\_Rotation

ST\_Rotation — Gibt die Rotation des Rasters im Bogenmaß aus.

#### Synopsis

```
float8 ST_Rotation(raster rast);
```

#### 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](#)

### 11.4.15 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](#)

**11.4.16 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](#)

### 11.4.17 ST\_SRID

`ST_SRID` — Gibt den Identifikator des Koordinatenreferenzsystems des Rasters aus, das in der Tabelle "spatial\_ref\_sys" definiert ist.

#### Synopsis

```
integer ST_SRID(raster rast);
```

#### Beschreibung

Gibt den Identifikator des Koordinatenreferenzsystems des Rasters aus, das in der Tabelle "spatial\_ref\_sys" definiert ist.



#### Note

Ab PostGIS 2.0 ist die SRID eines nicht georeferenzierten Rasters/Geometrie 0 anstelle wie früher -1.

#### Beispiele

```
SELECT ST_SRID(rast) As srid
FROM dummy_rast WHERE rid=1;
```

```
srid
-----
0
```

#### Siehe auch

Section [4.5](#), [ST\\_SRID](#)

### 11.4.18 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](#)

### 11.4.19 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](#)

### 11.4.20 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](#)

### 11.4.21 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](#)

### 11.4.22 ST\_WorldToRasterCoord

`ST_WorldToRasterCoord` — Gibt für ein geometrisches X und Y (geographische Länge und Breite) oder für eine Punktgeometrie im Koordinatenreferenzsystem des Rasters, die obere linke Ecke als Spalte und Zeile aus.

#### Synopsis

```
record ST_WorldToRasterCoord(raster rast, geometry pt);
record ST_WorldToRasterCoord(raster rast, double precision longitude, double precision latitude);
```

#### 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](#)

### 11.4.23 ST\_WorldToRasterCoordX

`ST_WorldToRasterCoordX` — Gibt für eine Punktgeometrie (pt) oder eine globale X- und Y-Koordinate (xw, yw) die Rasterspalte im globalen Koordinatenreferenzsystem des Rasters aus.

#### Synopsis

```
integer ST_WorldToRasterCoordX(raster rast, geometry pt);
integer ST_WorldToRasterCoordX(raster rast, double precision xw);
integer ST_WorldToRasterCoordX(raster rast, double precision xw, double precision yw);
```

#### 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](#)

**11.4.24 ST\_WorldToRasterCoordY**

**ST\_WorldToRasterCoordY** — Gibt für eine Punktgeometrie (pt) oder eine globale X- und Y-Koordinate (xw, yw) die Rasterzeile im globalen Koordinatenreferenzsystem des Rasters aus.

**Synopsis**

```
integer ST_WorldToRasterCoordY(raster rast, geometry pt);
integer ST_WorldToRasterCoordY(raster rast, double precision xw);
integer ST_WorldToRasterCoordY(raster rast, double precision xw, double precision yw);
```

**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](#)

## 11.5 Zugriffsfunktionen auf Rasterbänder

### 11.5.1 ST\_BandMetaData

`ST_BandMetaData` — Gibt die grundlegenden Metadaten eines bestimmten Rasterbandes aus. Wenn der Parameter "bandnum" nicht angegeben ist, wird das 1ste Band angenommen.

#### Synopsis

- (1) record `ST_BandMetaData`(raster rast, integer band=1);
- (2) record `ST_BandMetaData`(raster rast, integer[] band);

#### Beschreibung

Gibt die grundlegenden Metadaten eines Rasterbandes aus. Die zurückgegebenen Spalten sind `pixeltype`, `nodatavalue`, `isoutdb`, `path`, `filesize` und `filetimestamp`.



#### Note

Wenn der Raster keine Bänder beinhaltet wird ein Fehler gemeldet.



#### Note

Wenn für das Band kein NODATA-Wert vergeben wurde, wird der NODATA-Wert auf NULL gesetzt.



#### Note

Wenn `isoutdb` False ist, dann sind `path`, `outdbbandnum`, `filesize` und `filetimestamp` NULL. Wenn der Zugriff auf `otdb`-Raster deaktiviert ist, dann sind `filesize` und `filetimestamp` ebenfalls NULL.

Erweiterung: 2.5.0 inkludiert jetzt `outdbbandnum`, `filesize` und `filetimestamp` für `outdb` Raster.

#### Beispiele: Variante 1

```
SELECT
  rid,
  (foo.md).*
FROM (
  SELECT
    rid,
    ST_BandMetaData(rast, 1) AS md
  FROM dummy_rast
  WHERE rid=2
) As foo;
```

rid	pixeltype	nodatavalue	isoutdb	path	outdbbandnum
2	8BUI	0	f		

**Beispiele: Variante 2**

```

WITH foo AS (
  SELECT
    ST_AddBand(NULL::raster, '/home/pele/devel/geo/postgis-git/raster/test/regress/ ←
      loader/Projected.tif', NULL::int[]) AS rast
)
SELECT
  *
FROM ST_BandMetadata(
  (SELECT rast FROM foo),
  ARRAY[1,3,2]::int[]
);

```

bandnum	pixeltype	nodatavalue	isoutdb	outdbbandnum	filesize	filetimestamp	path
1	8BUI		t		1	12345	1521807257
3	8BUI		t		3	12345	1521807257
2	8BUI		t		2	12345	1521807257

**Siehe auch**

[ST\\_MetaData](#), [ST\\_BandPixelType](#)

**11.5.2 ST\_BandNoDataValue**

`ST_BandNoDataValue` — Gibt den NODATA Wert des gegebenen Bandes aus. Wenn der Parameter "bandnum" nicht angegeben ist, wird das 1ste Band angenommen.

**Synopsis**

```
double precision ST_BandNoDataValue(raster rast, integer bandnum=1);
```

**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](#)**11.5.3 ST\_BandIsNoData**

ST\_BandIsNoData — Gibt TRUE aus, wenn das Band ausschließlich aus NODATA Werten besteht.

**Synopsis**

boolean **ST\_BandIsNoData**(raster rast, integer band, boolean forceChecking=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\_SetBandsNodata() anwenden, oder ST\_SetBandNodataValue() mit TRUE als letzten Übergabewert ausführen. Siehe [ST\\_SetBandsNoData](#).

**Beispiele**

```
-- Create dummy table with one raster column
create table dummy_rast (rid integer, rast raster);

-- Add raster with two bands, one pixel/band. In the first band, nodatavalue = pixel value ←
  = 3.
-- In the second band, nodatavalue = 13, pixel value = 4
insert into dummy_rast values(1,
(
'01' -- little endian (uint8 ndr)
||
'0000' -- version (uint16 0)
||
'0200' -- nBands (uint16 0)
||
'17263529ED684A3F' -- scaleX (float64 0.000805965234044584)
||
'F9253529ED684ABF' -- scaleY (float64 -0.00080596523404458)
||
'1C9F33CE69E352C0' -- ipX (float64 -75.5533328537098)
||
'718F0E9A27A44840' -- ipY (float64 49.2824585505576)
||
'ED50EB853EC32B3F' -- skewX (float64 0.000211812383858707)
||
'7550EB853EC32B3F' -- skewY (float64 0.000211812383858704)
```

```

||
'E6100000' -- SRID (int32 4326)
||
'0100' -- width (uint16 1)
||
'0100' -- height (uint16 1)
||
'6' -- hasnodatavalue and isnodata value set to true.
||
'2' -- first band type (4BUI)
||
'03' -- novalue==3
||
'03' -- pixel(0,0)==3 (same that nodata)
||
'0' -- hasnodatavalue set to false
||
'5' -- second band type (16BSI)
||
'0D00' -- novalue==13
||
'0400' -- pixel(0,0)==4
)::raster
);

select st_bandisnodata(rast, 1) from dummy_rast where rid = 1; -- Expected true
select st_bandisnodata(rast, 2) from dummy_rast where rid = 1; -- Expected false

```

**Siehe auch**

[ST\\_BandNoDataValue](#), [ST\\_NumBands](#), [ST\\_SetBandNoDataValue](#), [ST\\_SetBandIsNoData](#)

**11.5.4 ST\_BandPath**

`ST_BandPath` — Gibt den Dateipfad aus, unter dem das Band im Dateisystem gespeichert ist. Wenn "bandnum" nicht angegeben ist, wird 1 angenommen.

**Synopsis**

```
text ST_BandPath(raster rast, integer bandnum=1);
```

**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****11.5.5 ST\_BandFileSize**

`ST_BandFileSize` — Gibt die Dateigröße eines im Dateisystem gespeicherten Bandes aus. Wenn "bandnum" nicht angegeben ist, wird 1 angenommen.

**Synopsis**

```
bigint ST_BandFileSize(raster rast, integer bandnum=1);
```

**Beschreibung**

Gibt die Dateigröße eines im Dateisystem gespeicherten Bandes aus. Wenn die Funktion mit einem indb-Band aufgerufen wird oder der outdb-Zugriff deaktiviert ist, wird eine Fehlermeldung ausgegeben.

Diese Funktion wird üblicherweise gemeinsam mit `ST_BandPath()` und `ST_BandFileTimestamp()` verwendet. Auf diese Weise kann ein Client feststellen, ob er die selbe Sicht auf den Dateinamen eines outdb-Rasters hat wie der Server.

Verfügbarkeit: 2.5.0

**Beispiele**

```
SELECT ST_BandFileSize(rast,1) FROM dummy_rast WHERE rid = 1;

 st_bandfilesize
-----
          240574
```

**11.5.6 ST\_BandFileTimestamp**

`ST_BandFileTimestamp` — Gibt den Zeitstempel eines im Dateisystem gespeicherten Bandes aus. Wenn "bandnum" nicht angegeben ist, wird 1 angenommen.

**Synopsis**

```
bigint ST_BandFileTimestamp(raster rast, integer bandnum=1);
```

**Beschreibung**

Gibt den Zeitstempel (Anzahl der Sekunden seit Jan 1st 1970 00:00:00 UTC) eines im Dateisystem gespeicherten Bandes aus. Wenn die Funktion mit einem indb-Band aufgerufen wird oder der outdb-Zugriff deaktiviert ist, wird eine Fehlermeldung ausgegeben.

Diese Funktion wird üblicherweise gemeinsam mit `ST_BandPath()` und `ST_BandFileSize()` verwendet. Auf diese Weise kann ein Client feststellen, ob er die selbe Sicht auf den Dateinamen eines outdb-Rasters hat wie der Server.

Verfügbarkeit: 2.5.0

**Beispiele**

```
SELECT ST_BandFileTimestamp(rast,1) FROM dummy_rast WHERE rid = 1;

 st_bandfiletimestamp
-----
          1521807257
```

**11.5.7 ST\_BandPixelType**

`ST_BandPixelType` — Gibt den Pixeltyp des angegebenen Bandes aus. Wenn der Parameter "bandnum" nicht angegeben ist, wird das 1ste Band angenommen.

**Synopsis**

text **ST\_BandPixelType**(raster rast, integer bandnum=1);

**Beschreibung**

Gibt eine Beschreibung des Datentyps und der Größe der Zellwerte in dem gegebenen Band zurück.

Im Folgenden die 11 unterstützten Pixeltypen

- 1BB - 1-Bit Boolean
- 2BUI - 2-Bit vorzeichenlose Ganzzahl
- 4BUI - 4-Bit vorzeichenlose Ganzzahl
- 8BSI - 8-Bit Ganzzahl
- 8BUI - 8-Bit vorzeichenlose Ganzzahl
- 16BSI - 16-Bit Ganzzahl
- 16BUI - 16-bit vorzeichenlose Ganzzahl
- 32BSI - 32-Bit Ganzzahl
- 32BUI - 32-Bit vorzeichenlose Ganzzahl
- 32BF - 32-Bit Float
- 64BF - 64-Bit Float

**Beispiele**

```
SELECT ST_BandPixelType(rast,1) As btype1,
       ST_BandPixelType(rast,2) As btype2, ST_BandPixelType(rast,3) As btype3
FROM dummy_rast
WHERE rid = 2;
```

```
 btype1 | btype2 | btype3
-----+-----+-----
 8BUI   | 8BUI   | 8BUI
```

**Siehe auch**

[ST\\_NumBands](#)

**11.5.8 ST\_MinPossibleValue**

**ST\_MinPossibleValue** — Gibt den Mindestwert zurück, den dieser Pixeltyp speichern kann.

**Synopsis**

integer **ST\_MinPossibleValue**(text pixeltype);

**Beschreibung**

Gibt den Mindestwert zurück, den dieser Pixeltyp speichern kann.

**Beispiele**

```
SELECT ST_MinPossibleValue('16BSI');

 st_minpossiblevalue
-----
                -32768

SELECT ST_MinPossibleValue('8BUI');

 st_minpossiblevalue
-----
                    0
```

**Siehe auch**

[ST\\_BandPixelType](#)

**11.5.9 ST\_HasNoBand**

**ST\_HasNoBand** — Gibt TRUE aus, wenn kein Band mit der angegebenen Bandnummer existiert. Gibt den Pixeltyp des angegebenen Bandes aus. Wenn keine Bandnummer angegeben ist, wird das 1ste Band angenommen.

**Synopsis**

boolean **ST\_HasNoBand**(raster rast, integer bandnum=1);

**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](#)

## 11.6 Zugriffsfunktionen und Änderungsmethoden für Rasterpixel

### 11.6.1 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

```
-- get raster pixel polygon
SELECT i,j, ST_AsText(ST_PixelAsPolygon(foo.rast, i,j)) As blpgeom
FROM dummy_rast As foo
      CROSS JOIN generate_series(1,2) As i
      CROSS JOIN generate_series(1,1) As j
WHERE rid=2;
```

i	j	blpgeom
1	1	POLYGON((3427927.75 5793244,3427927.8 5793244,3427927.8 5793243.95,...
2	1	POLYGON((3427927.8 5793244,3427927.85 5793244,3427927.85 5793243.95, ..

#### Siehe auch

[ST\\_DumpAsPolygons](#), [ST\\_PixelAsPolygons](#), [ST\\_PixelAsPoint](#), [ST\\_PixelAsPoints](#), [ST\\_PixelAsCentroid](#), [ST\\_PixelAsCentroids](#), [ST\\_Intersection](#), [ST\\_AsText](#)

### 11.6.2 ST\_PixelAsPolygons

ST\_PixelAsPolygons — Gibt die umhüllende 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.

Datensatzformatausgabe: *geom* **geometry**, *val* double precision, *x* integer, *y* integers.

**Note**

Wenn `exclude_nodata_value = TRUE`, dann werden nur jene Pixel als Punkte zurückgegeben deren Zellwerte nicht NODATA sind.

**Note**

`ST_PixelAsPolygons` gibt eine Polygoneometrie pro Pixel zurück. Dies unterscheidet sich von `ST_DumpAsPolygons`, wo jede Geometrie einen oder mehrere Pixel mit gleichem Zellwert darstellt.

Verfügbarkeit: 2.0.0

Erweiterung: 2.1.0 Der optionale Übergabewert "exclude\_nodata\_value" wurde hinzugefügt.

Änderung: 2.1.1 Die Verhaltensweise von "exclude\_nodata\_value" wurde geändert.

**Beispiele**

```
-- get raster pixel polygon
SELECT (gv).x, (gv).y, (gv).val, ST_AsText((gv).geom) geom
FROM (SELECT ST_PixelAsPolygons(
        ST_SetValue(ST_SetValue(ST_AddBand(ST_MakeEmptyRaster(2, 2, 0, 0, 0.001, ←
        -0.001, 0.001, 0.001, 4269),
        '8BUI'::text, 1, 0),
        2, 2, 10),
        1, 1, NULL)
) gv
) foo;
```

x	y	val	geom
1	1		POLYGON((0 0,0.001 0.001,0.002 0,0.001 -0.001,0 0))
1	2	1	POLYGON((0.001 -0.001,0.002 0,0.003 -0.001,0.002 -0.002,0.001 -0.001))
2	1	1	POLYGON((0.001 0.001,0.002 0.002,0.003 0.001,0.002 0,0.001 0.001))
2	2	10	POLYGON((0.002 0,0.003 0.001,0.004 0,0.003 -0.001,0.002 0))

**Siehe auch**

[ST\\_DumpAsPolygons](#), [ST\\_PixelAsPolygon](#), [ST\\_PixelAsPoint](#), [ST\\_PixelAsPoints](#), [ST\\_PixelAsCentroid](#), [ST\\_PixelAsCentroids](#), [ST\\_AsText](#)

**11.6.3 ST\_PixelAsPoint**

`ST_PixelAsPoint` — Gibt eine Punktgeometrie der oberen linken Ecke des Rasters zurück.

**Synopsis**

geometry `ST_PixelAsPoint`(raster rast, integer columnx, integer rowy);

**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](#)

### 11.6.4 ST\_PixelAsPoints

`ST_PixelAsPoints` — Gibt eine Punktgeometrie für jedes Pixel des Rasterbandes zurück, zusammen mit dem Zellwert und den X- und Y-Rasterkoordinaten eines jeden Pixels. Die Koordinaten der Punkte entsprechen dem oberen linken Eck der Pixel.

## Synopsis

setof record `ST_PixelAsPoints`(raster rast, integer band=1, boolean exclude\_nodata\_value=TRUE);

## Beschreibung

Gibt eine Punktgeometrie für jedes Pixel des Rasterbandes zurück, zusammen mit dem Zellwert und den X- und Y-Rasterkoordinaten eines jeden Pixels. Die Koordinaten der Punkte entsprechen dem oberen linken Eck der Pixel.

Datensatzformatausgabe: *geom* **geometry**, *val* double precision, *x* integer, *y* integers.



### Note

Wenn `exclude_nodata_value = TRUE`, dann werden nur jene Pixel als Punkte zurückgegeben deren Zellwerte nicht NODATA sind.

Verfügbarkeit: 2.1.0

Änderung: 2.1.1 Die Verhaltensweise von "exclude\_nodata\_value" wurde geändert.

## Beispiele

```
SELECT x, y, val, ST_AsText(geom) FROM (SELECT (ST_PixelAsPoints(rast, 1)).* FROM ↵
  dummy_rast WHERE rid = 2) foo;
```

x	y	val	st_astext
1	1	253	POINT(3427927.75 5793244)
2	1	254	POINT(3427927.8 5793244)
3	1	253	POINT(3427927.85 5793244)
4	1	254	POINT(3427927.9 5793244)
5	1	254	POINT(3427927.95 5793244)
1	2	253	POINT(3427927.75 5793243.95)
2	2	254	POINT(3427927.8 5793243.95)
3	2	254	POINT(3427927.85 5793243.95)
4	2	253	POINT(3427927.9 5793243.95)
5	2	249	POINT(3427927.95 5793243.95)



```

1 | 3 | 250 | POINT (3427927.75 5793243.9)
2 | 3 | 254 | POINT (3427927.8 5793243.9)
3 | 3 | 254 | POINT (3427927.85 5793243.9)
4 | 3 | 252 | POINT (3427927.9 5793243.9)
5 | 3 | 249 | POINT (3427927.95 5793243.9)
1 | 4 | 251 | POINT (3427927.75 5793243.85)
2 | 4 | 253 | POINT (3427927.8 5793243.85)
3 | 4 | 254 | POINT (3427927.85 5793243.85)
4 | 4 | 254 | POINT (3427927.9 5793243.85)
5 | 4 | 253 | POINT (3427927.95 5793243.85)
1 | 5 | 252 | POINT (3427927.75 5793243.8)
2 | 5 | 250 | POINT (3427927.8 5793243.8)
3 | 5 | 254 | POINT (3427927.85 5793243.8)
4 | 5 | 254 | POINT (3427927.9 5793243.8)
5 | 5 | 254 | POINT (3427927.95 5793243.8)

```

**Siehe auch**

[ST\\_DumpAsPolygons](#), [ST\\_PixelAsPolygon](#), [ST\\_PixelAsPolygons](#), [ST\\_PixelAsPoint](#), [ST\\_PixelAsCentroid](#), [ST\\_PixelAsCentroids](#)

**11.6.5 ST\_PixelAsCentroid**

`ST_PixelAsCentroid` — Gibt den geometrischen Schwerpunkt (Punktgeometrie) der Fläche aus, die durch das Pixel repräsentiert wird.

**Synopsis**

geometry `ST_PixelAsCentroid`(raster rast, integer x, integer y);

**Beschreibung**

Gibt den geometrischen Schwerpunkt (Punktgeometrie) der Fläche aus, die durch das Pixel repräsentiert wird.

Verbessert: 3.2.0 Schneller jetzt in C implementiert.

Verfügbarkeit: 2.1.0

**Beispiele**

```
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](#)

**11.6.6 ST\_PixelAsCentroids**

`ST_PixelAsCentroids` — Gibt den geometrischen Schwerpunkt (Punktgeometrie) für jedes Pixel des Rasterbandes zurück, zusammen mit dem Zellwert und den X- und Y-Rasterkoordinaten eines jeden Pixels. Die Koordinaten der Punkte entsprechen dem geometrischen Schwerpunkt der Pixel.

## Synopsis

```
setof record ST_PixelAsCentroids(raster rast, integer band=1, boolean exclude_nodata_value=TRUE);
```

## Beschreibung

Gibt den geometrischen Schwerpunkt (Punktgeometrie) für jedes Pixel des Rasterbandes zurück, zusammen mit dem Zellwert und den X- und Y-Rasterkoordinaten eines jeden Pixels. Die Koordinaten der Punkte entsprechen dem geometrischen Schwerpunkt der Pixel.

Datensatzformatausgabe: *geom* **geometry**, *val* double precision, *x* integer, *y* integers.



### Note

Wenn `exclude_nodata_value = TRUE`, dann werden nur jene Pixel als Punkte zurückgegeben deren Zellwerte nicht NODATA sind.

Verbessert: 3.2.0 Schneller jetzt in C implementiert.

Änderung: 2.1.1 Die Verhaltensweise von "exclude\_nodata\_value" wurde geändert.

Verfügbarkeit: 2.1.0

## Beispiele

```
--LATERAL syntax requires PostgreSQL 9.3+
SELECT x, y, val, ST_AsText(geom)
  FROM (SELECT dp.* FROM dummy_rast, LATERAL ST_PixelAsCentroids(rast, 1) AS dp WHERE rid <=
        = 2) foo;
 x | y | val |          st_astext
---+---+---+-----
 1 | 1 | 253 | POINT(3427927.775 5793243.975)
 2 | 1 | 254 | POINT(3427927.825 5793243.975)
 3 | 1 | 253 | POINT(3427927.875 5793243.975)
 4 | 1 | 254 | POINT(3427927.925 5793243.975)
 5 | 1 | 254 | POINT(3427927.975 5793243.975)
 1 | 2 | 253 | POINT(3427927.775 5793243.925)
 2 | 2 | 254 | POINT(3427927.825 5793243.925)
 3 | 2 | 254 | POINT(3427927.875 5793243.925)
 4 | 2 | 253 | POINT(3427927.925 5793243.925)
 5 | 2 | 249 | POINT(3427927.975 5793243.925)
 1 | 3 | 250 | POINT(3427927.775 5793243.875)
 2 | 3 | 254 | POINT(3427927.825 5793243.875)
 3 | 3 | 254 | POINT(3427927.875 5793243.875)
 4 | 3 | 252 | POINT(3427927.925 5793243.875)
 5 | 3 | 249 | POINT(3427927.975 5793243.875)
 1 | 4 | 251 | POINT(3427927.775 5793243.825)
 2 | 4 | 253 | POINT(3427927.825 5793243.825)
 3 | 4 | 254 | POINT(3427927.875 5793243.825)
 4 | 4 | 254 | POINT(3427927.925 5793243.825)
 5 | 4 | 253 | POINT(3427927.975 5793243.825)
 1 | 5 | 252 | POINT(3427927.775 5793243.775)
 2 | 5 | 250 | POINT(3427927.825 5793243.775)
 3 | 5 | 254 | POINT(3427927.875 5793243.775)
 4 | 5 | 254 | POINT(3427927.925 5793243.775)
 5 | 5 | 254 | POINT(3427927.975 5793243.775)
```

**Siehe auch**

[ST\\_DumpAsPolygons](#), [ST\\_PixelAsPolygon](#), [ST\\_PixelAsPolygons](#), [ST\\_PixelAsPoint](#), [ST\\_PixelAsPoints](#), [ST\\_PixelAsCentroid](#)

**11.6.7 ST\_Value**

**ST\_Value** — Gibt den Zellwert eines Pixels aus, das über `columnx` und `rowy` oder durch einen bestimmten geometrischen Punkt angegeben wird. Die Bandnummern beginnen mit 1 und wenn keine Bandnummer angegeben ist, dann wird Band 1 angenommen. Wenn `exclude_nodata_value` auf `FALSE` gesetzt ist, werden auch die Pixel mit einem `nodata` Wert mit einbezogen. Wenn `exclude_nodata_value` nicht übergeben wird, dann wird er über die Metadaten des Rasters ausgelesen.

**Synopsis**

```
double precision ST_Value(raster rast, geometry pt, boolean exclude_nodata_value=true);
double precision ST_Value(raster rast, integer band, geometry pt, boolean exclude_nodata_value=true, text resample='nearest');
double precision ST_Value(raster rast, integer x, integer y, boolean exclude_nodata_value=true);
double precision ST_Value(raster rast, integer band, integer x, integer y, boolean exclude_nodata_value=true);
```

**Beschreibung**

Gibt den Wert eines bestimmten Bandes in einer bestimmten Spalte, Zeile Pixel oder an einem bestimmten Geometripunkt zurück. Die Bandnummern beginnen bei 1 und es wird angenommen, dass Band 1 ist, wenn nicht angegeben.

Wenn `exclude_nodata_value` auf `true` gesetzt ist, dann werden nur Pixel berücksichtigt, die nicht `nodata` sind. Wenn `exclude_nodata_value` auf `false` gesetzt ist, werden alle Pixel berücksichtigt.

Die zulässigen Werte des Parameters `Resample` sind "nearest", der das Standard-Resampling der nächsten Nachbarn durchführt, und "bilinear", der eine **bilineare Interpolation** durchführt, um den Wert zwischen den Pixelzentren zu schätzen.

Verbessert: 3.2.0 Das optionale Argument `resample` wurde hinzugefügt.

Erweiterung: 2.0.0 Der optionale Übergabewert "exclude\_nodata\_value" wurde hinzugefügt.

**Beispiele**

```
-- get raster values at particular postgis geometry points
-- the srid of your geometry should be same as for your raster
SELECT rid, ST_Value(rast, foo.pt_geom) As b1pval, ST_Value(rast, 2, foo.pt_geom) As b2pval
FROM dummy_rast CROSS JOIN (SELECT ST_SetSRID(ST_Point(3427927.77, 5793243.76), 0) As
    pt_geom) As foo
WHERE rid=2;
```

```
rid | b1pval | b2pval
-----+-----+-----
  2 |    252 |    79
```

```
-- general fictitious example using a real table
SELECT rid, ST_Value(rast, 3, sometable.geom) As b3pval
FROM sometable
WHERE ST_Intersects(rast, sometable.geom);
```

```
SELECT rid, ST_Value(rast, 1, 1, 1) As b1pval,
    ST_Value(rast, 2, 1, 1) As b2pval, ST_Value(rast, 3, 1, 1) As b3pval
FROM dummy_rast
WHERE rid=2;
```

rid	b1pval	b2pval	b3pval
2	253	78	70

```

--- Get all values in bands 1,2,3 of each pixel --
SELECT x, y, ST_Value(rast, 1, x, y) As b1val,
       ST_Value(rast, 2, x, y) As b2val, ST_Value(rast, 3, x, y) As b3val
FROM dummy_rast CROSS JOIN
generate_series(1, 1000) As x CROSS JOIN generate_series(1, 1000) As y
WHERE rid = 2 AND x <= ST_Width(rast) AND y <= ST_Height(rast);

```

x	y	b1val	b2val	b3val
1	1	253	78	70
1	2	253	96	80
1	3	250	99	90
1	4	251	89	77
1	5	252	79	62
2	1	254	98	86
2	2	254	118	108
:				
:				

```

--- Get all values in bands 1,2,3 of each pixel same as above but returning the upper left ←
point point of each pixel --
SELECT ST_AsText(ST_SetSRID(
  ST_Point(ST_UpperLeftX(rast) + ST_ScaleX(rast)*x,
          ST_UpperLeftY(rast) + ST_ScaleY(rast)*y),
  ST_SRID(rast))) As uplpt
, ST_Value(rast, 1, x, y) As b1val,
  ST_Value(rast, 2, x, y) As b2val, ST_Value(rast, 3, x, y) As b3val
FROM dummy_rast CROSS JOIN
generate_series(1,1000) As x CROSS JOIN generate_series(1,1000) As y
WHERE rid = 2 AND x <= ST_Width(rast) AND y <= ST_Height(rast);

```

uplpt	b1val	b2val	b3val
POINT(3427929.25 5793245.5)	253	78	70
POINT(3427929.25 5793247)	253	96	80
POINT(3427929.25 5793248.5)	250	99	90
:			

```

--- Get a polygon formed by union of all pixels
that fall in a particular value range and intersect particular polygon --
SELECT ST_AsText(ST_Union(pixpolyg)) As shadow
FROM (SELECT ST_Translate(ST_MakeEnvelope(
  ST_UpperLeftX(rast), ST_UpperLeftY(rast),
  ST_UpperLeftX(rast) + ST_ScaleX(rast),
  ST_UpperLeftY(rast) + ST_ScaleY(rast), 0
), ST_ScaleX(rast)*x, ST_ScaleY(rast)*y
) As pixpolyg, ST_Value(rast, 2, x, y) As b2val
FROM dummy_rast CROSS JOIN
generate_series(1,1000) As x CROSS JOIN generate_series(1,1000) As y
WHERE rid = 2
AND x <= ST_Width(rast) AND y <= ST_Height(rast)) As foo
WHERE

```

```

ST_Intersects(
  pixpolyg,
  ST_GeomFromText('POLYGON((3427928 5793244,3427927.75 5793243.75,3427928 ←
    5793243.75,3427928 5793244))',0)
) AND b2val != 254;

```

shadow

```

-----
MULTIPOLYGON(((3427928 5793243.9,3427928 5793243.85,3427927.95 5793243.85,3427927.95 ←
  5793243.9,
  3427927.95 5793243.95,3427928 5793243.95,3427928.05 5793243.95,3427928.05 ←
    5793243.9,3427928 5793243.9)),((3427927.95 5793243.9,3427927.95 579324
  3.85,3427927.9 5793243.85,3427927.85 5793243.85,3427927.85 5793243.9,3427927.9 ←
    5793243.9,3427927.9 5793243.95,
  3427927.95 5793243.95,3427927.95 5793243.9)),((3427927.85 5793243.75,3427927.85 ←
    5793243.7,3427927.8 5793243.7,3427927.8 5793243.75
  ,3427927.8 5793243.8,3427927.8 5793243.85,3427927.85 5793243.85,3427927.85 ←
    5793243.8,3427927.85 5793243.75)),
  ((3427928.05 5793243.75,3427928.05 5793243.7,3427928 5793243.7,3427927.95 ←
    5793243.7,3427927.95 5793243.75,3427927.95 5793243.8,3427
  927.95 5793243.85,3427928 5793243.85,3427928 5793243.8,3427928.05 5793243.8,
  3427928.05 5793243.75)),((3427927.95 5793243.75,3427927.95 5793243.7,3427927.9 ←
    5793243.7,3427927.85 5793243.7,
  3427927.85 5793243.75,3427927.85 5793243.8,3427927.85 5793243.85,3427927.9 5793243.85,
  3427927.95 5793243.85,3427927.95 5793243.8,3427927.95 5793243.75)))

```

```

--- Checking all the pixels of a large raster tile can take a long time.
--- You can dramatically improve speed at some lose of precision by orders of magnitude
-- by sampling pixels using the step optional parameter of generate_series.
-- This next example does the same as previous but by checking 1 for every 4 (2x2) pixels ←
-- and putting in the last checked
-- putting in the checked pixel as the value for subsequent 4

```

```

SELECT ST_AsText(ST_Union(pixpolyg)) As shadow
FROM (SELECT ST_Translate(ST_MakeEnvelope(
  ST_UpperLeftX(rast), ST_UpperLeftY(rast),
  ST_UpperLeftX(rast) + ST_ScaleX(rast)*2,
  ST_UpperLeftY(rast) + ST_ScaleY(rast)*2, 0
), ST_ScaleX(rast)*x, ST_ScaleY(rast)*y
) As pixpolyg, ST_Value(rast, 2, x, y) As b2val
FROM dummy_rast CROSS JOIN
generate_series(1,1000,2) As x CROSS JOIN generate_series(1,1000,2) As y
WHERE rid = 2
AND x <= ST_Width(rast) AND y <= ST_Height(rast) ) As foo
WHERE
  ST_Intersects(
    pixpolyg,
    ST_GeomFromText('POLYGON((3427928 5793244,3427927.75 5793243.75,3427928 ←
      5793243.75,3427928 5793244))',0)
  ) AND b2val != 254;

shadow
-----
MULTIPOLYGON(((3427927.9 5793243.85,3427927.8 5793243.85,3427927.8 5793243.95,
  3427927.9 5793243.95,3427928 5793243.95,3427928.1 5793243.95,3427928.1 5793243.85,3427928 ←
    5793243.85,3427927.9 5793243.85)),
  ((3427927.9 5793243.65,3427927.8 5793243.65,3427927.8 5793243.75,3427927.8 ←
    5793243.85,3427927.9 5793243.85,
  3427928 5793243.85,3427928 5793243.75,3427928.1 5793243.75,3427928.1 5793243.65,3427928 ←

```

```
5793243.65,3427927.9 5793243.65) ) )
```

## Siehe auch

[ST\\_SetValue](#), [ST\\_DumpAsPolygons](#), [ST\\_NumBands](#), [ST\\_PixelAsPolygon](#), [ST\\_ScaleX](#), [ST\\_ScaleY](#), [ST\\_UpperLeftX](#), [ST\\_UpperLeftY](#), [ST\\_SRID](#), [ST\\_AsText](#), [ST\\_Point](#), [ST\\_MakeEnvelope](#), [ST\\_Intersects](#), [ST\\_Intersection](#)

## 11.6.8 ST\_NearestValue

`ST_NearestValue` — Gibt den nächstgelegenen nicht `NODATA` Wert eines bestimmten Pixels aus, das über "columnx" und "rowy" oder durch eine Punktgeometrie - im gleichen Koordinatenreferenzsystem wie der Raster - ausgewählt wird.

### Synopsis

```
double precision ST_NearestValue(raster rast, integer bandnum, geometry pt, boolean exclude_nodata_value=true);
double precision ST_NearestValue(raster rast, geometry pt, boolean exclude_nodata_value=true);
double precision ST_NearestValue(raster rast, integer bandnum, integer columnx, integer rowy, boolean exclude_nodata_value=true);
double precision ST_NearestValue(raster rast, integer columnx, integer rowy, boolean exclude_nodata_value=true);
```

### Beschreibung

Gibt den nächstgelegenen, nicht-`NODATA` Wert eines bestimmten Pixels aus, das über "columnx" und "rowy", oder durch einen geometrischen Punkt angegeben wird. Falls das angegebene Pixel den Wert `NODATA` hat, findet die Funktion das nächstgelegene Pixel, das nicht den Wert `NODATA` hat.

Die Bandnummern beginnen mit 1 und wenn keine Bandnummer angegeben ist, dann wird für `bandnum` 1 angenommen. Wenn `exclude_nodata_value` auf `FALSE` gesetzt ist, werden auch die Pixel mit einem `nodata` Wert einbezogen. Wenn `exclude_nodata_value` nicht übergeben wird, dann wird er über die Metadaten des Rasters ausgelesen.

Verfügbarkeit: 2.1.0



#### Note

`ST_NearestValue` ist eine Alternative zu `ST_Value`.

### Beispiele

```
-- pixel 2x2 has value
SELECT
  ST_Value(rast, 2, 2) AS value,
  ST_NearestValue(rast, 2, 2) AS nearestvalue
FROM (
  SELECT
    ST_SetValue(
      ST_SetValue(
        ST_SetValue(
          ST_SetValue(
            ST_SetValue(
              ST_SetValue(
                ST_AddBand(
                  ST_MakeEmptyRaster(5, 5, -2, 2, 1, -1, 0, 0, 0),
                  '8BUI'::text, 1, 0
                ),
                1, 0
              ),
              1, 0
            ),
            1, 0
          ),
          1, 0
        ),
        1, 0
      ),
      1, 0
    ),
    1, 0
  ),
  1, 0
```

```

        1, 1, 0.
    ),
    2, 3, 0.
),
    3, 5, 0.
),
    4, 2, 0.
),
    5, 4, 0.
) AS rast
) AS foo

value | nearestvalue
-----+-----
1 | 1

-- pixel 2x3 is NODATA
SELECT
    ST_Value(rast, 2, 3) AS value,
    ST_NearestValue(rast, 2, 3) AS nearestvalue
FROM (
    SELECT
        ST_SetValue(
            ST_SetValue(
                ST_SetValue(
                    ST_SetValue(
                        ST_SetValue(
                            ST_AddBand(
                                ST_MakeEmptyRaster(5, 5, -2, 2, 1, -1, 0, 0, 0),
                                '8BUI'::text, 1, 0
                            ),
                            1, 1, 0.
                        ),
                        2, 3, 0.
                    ),
                    3, 5, 0.
                ),
                4, 2, 0.
            ),
            5, 4, 0.
        ) AS rast
    ) AS foo

value | nearestvalue
-----+-----
| 1

```

**Siehe auch**

[ST\\_Neighborhood](#), [ST\\_Value](#)

**11.6.9 ST\_SetZ**

**ST\_SetZ** — Gibt eine Geometrie mit denselben X/Y-Koordinaten wie die Eingabegeometrie zurück, wobei die Werte aus dem Raster mit dem gewünschten Resample-Algorithmus in die Z-Dimension kopiert werden.

**Synopsis**

geometry **ST\_SetZ**(raster rast, geometry geom, text resample=nearest, integer band=1);

## Beschreibung

Gibt eine Geometrie mit denselben X/Y-Koordinaten wie die Eingabegeometrie zurück, wobei die Werte aus dem Raster mit dem gewünschten Resample-Algorithmus in die Z-Dimensionen kopiert werden.

Der Parameter `resample` kann auf "nearest" gesetzt werden, um die Werte aus der Zelle zu kopieren, in die jeder Scheitelpunkt fällt, oder auf "bilinear", um **bilineare Interpolation** zu verwenden, um einen Wert zu berechnen, der auch die benachbarten Zellen berücksichtigt.

Verfügbarkeit: 3.2.0

## Beispiele

```
--
-- 2x2 test raster with values
--
-- 10 50
-- 40 20
--
WITH test_raster AS (
SELECT
ST_SetValues(
  ST_AddBand(
    ST_MakeEmptyRaster(width => 2, height => 2,
      upperleftx => 0, upperlefty => 2,
      scalex => 1.0, scaley => -1.0,
      skewx => 0, skewy => 0, srid => 4326),
    index => 1, pixeltype => '16BSI',
    initialvalue => 0,
    nodataval => -999),
    1,1,1,
    newvalueset =>ARRAY[ARRAY[10.0::float8, 50.0::float8], ARRAY[40.0::float8, 20.0::float8 ←
      ]]) AS rast
)
SELECT
ST_AsText(
  ST_SetZ(
    rast,
    band => 1,
    geom => 'SRID=4326;LINESTRING(1.0 1.9, 1.0 0.2)::geometry,
    resample => 'bilinear'
  ))
FROM test_raster

          st_astext
-----
LINESTRING Z (1 1.9 38,1 0.2 27)
```

## Siehe auch

[ST\\_Value](#), [ST\\_SetM](#)

### 11.6.10 ST\_SetM

`ST_SetM` — Gibt eine Geometrie mit denselben X/Y-Koordinaten wie die Eingabegeometrie zurück, wobei die Werte aus dem Raster mit dem gewünschten Resample-Algorithmus in die Dimension M kopiert werden.



## Synopsis

geometry **ST\_SetM**(raster rast, geometry geom, text resample=nearest, integer band=1);

## Beschreibung

Liefert eine Geometrie mit denselben X/Y-Koordinaten wie die Eingabegeometrie und mit Werten aus dem Raster, die unter Verwendung des gewünschten Resample-Algorithmus in die M-Dimensionen kopiert wurden.

Der Parameter `resample` kann auf "nearest" gesetzt werden, um die Werte aus der Zelle zu kopieren, in die jeder Scheitelpunkt fällt, oder auf "bilinear", um **bilineare Interpolation** zu verwenden, um einen Wert zu berechnen, der auch die benachbarten Zellen berücksichtigt.

Verfügbarkeit: 3.2.0

## Beispiele

```
--
-- 2x2 test raster with values
--
-- 10 50
-- 40 20
--
WITH test_raster AS (
SELECT
ST_SetValues(
  ST_AddBand(
    ST_MakeEmptyRaster(width => 2, height => 2,
      upperleftx => 0, upperlefty => 2,
      scalex => 1.0, scaley => -1.0,
      skewx => 0, skewy => 0, srid => 4326),
    index => 1, pixeltype => '16BSI',
    initialvalue => 0,
    nodataval => -999),
    1,1,1,
    newvalueset =>ARRAY[ARRAY[10.0::float8, 50.0::float8], ARRAY[40.0::float8, 20.0::float8 ←
    ]]) AS rast
)
SELECT
ST_AsText(
  ST_SetM(
    rast,
    band => 1,
    geom => 'SRID=4326;LINESTRING(1.0 1.9, 1.0 0.2)::geometry,
    resample => 'bilinear'
  ))
FROM test_raster

          st_astext
-----
LINESTRING M (1 1.9 38,1 0.2 27)
```

## Siehe auch

[ST\\_Value](#), [ST\\_SetZ](#)

### 11.6.11 ST\_Neighborhood

**ST\_Neighborhood** — Gibt ein 2-D Feld in "Double Precision" aus, das sich aus nicht NODATA Werten um ein bestimmtes Pixel herum zusammensetzt. Das Pixel Kann über "columnx" und "rowy" oder über eine Punktgeometrie - im gleichen Koordinatenreferenzsystem wie der Raster - ausgewählt werden.

#### Synopsis

```
double precision[][] ST_Neighborhood(raster rast, integer bandnum, integer columnX, integer rowY, integer distanceX, integer distanceY, boolean exclude_nodata_value=true);
```

```
double precision[][] ST_Neighborhood(raster rast, integer columnX, integer rowY, integer distanceX, integer distanceY, boolean exclude_nodata_value=true);
```

```
double precision[][] ST_Neighborhood(raster rast, integer bandnum, geometry pt, integer distanceX, integer distanceY, boolean exclude_nodata_value=true);
```

```
double precision[][] ST_Neighborhood(raster rast, geometry pt, integer distanceX, integer distanceY, boolean exclude_nodata_value=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 Koordinatenreferenzsystem wie der Raster vorliegt übergeben werden. Die Parameter distanceX und distanceY bestimmen die Anzahl der Pixel auf der X- und Y-Achse, um das festgelegte Pixel herum; z.B.: Sie möchten alle Werte innerhalb einer Entfernung von 3 Pixel entlang der X-Achse und einer Entfernung von 2 Pixel entlang der Y-Achse, um das Pixel von Interesse herum. Der Wert im Zentrum des 2-D Feldes ist der Wert jenes Pixels, das über columnX und rowY oder über einen geometrischen Punkt übergeben wurde.

Die Bandnummern beginnen mit 1 und wenn keine Bandnummer angegeben ist, dann wird für bandnum 1 angenommen. Wenn exclude\_nodata\_value auf FALSE gesetzt ist, werden auch die Pixel mit einem nodata Wert einbezogen. Wenn exclude\_nodata\_value nicht übergeben wird, dann wird er über die Metadaten des Rasters ausgelesen.



#### Note

Die Anzahl der Elemente entlang jeder Achse des zurückgegebenen 2D-Feldes ergibt sich aus  $2 * (distanceX|distanceY) + 1$ . So ergibt sich bei einer distanceX und einer distanceY von je 1, ein Feld von 3x3.



#### Note

Das 2-D Feld kann einer beliebigen, integrierten Funktion zur Weiterverarbeitung übergeben werden, wie z.B. an ST\_Min4ma, ST\_Sum4ma, ST\_Mean4ma.

Verfügbarkeit: 2.1.0

#### Beispiele

```
-- pixel 2x2 has value
SELECT
  ST_Neighborhood(rast, 2, 2, 1, 1)
FROM (
  SELECT
    ST_SetValues(
      ST_AddBand(
        ST_MakeEmptyRaster(5, 5, -2, 2, 1, -1, 0, 0, 0),
        '8BUI'::text, 1, 0
```

```

    ),
    1, 1, 1, ARRAY[
        [0, 1, 1, 1, 1],
        [1, 1, 1, 0, 1],
        [1, 0, 1, 1, 1],
        [1, 1, 1, 1, 0],
        [1, 1, 0, 1, 1]
    ]::double precision[],
    1
) AS rast
) AS foo

st_neighborhood
-----
{{NULL,1,1},{1,1,1},{1,NULL,1}}

```

```

-- pixel 2x3 is NODATA
SELECT
    ST_Neighborhood(rast, 2, 3, 1, 1)
FROM (
    SELECT
        ST_SetValues(
            ST_AddBand(
                ST_MakeEmptyRaster(5, 5, -2, 2, 1, -1, 0, 0, 0),
                '8BUI'::text, 1, 0
            ),
            1, 1, 1, ARRAY[
                [0, 1, 1, 1, 1],
                [1, 1, 1, 0, 1],
                [1, 0, 1, 1, 1],
                [1, 1, 1, 1, 0],
                [1, 1, 0, 1, 1]
            ]::double precision[],
            1
        ) AS rast
    ) AS foo

st_neighborhood
-----
{{1,1,1},{1,NULL,1},{1,1,1}}

```

```

-- pixel 3x3 has value
-- exclude_nodata_value = FALSE
SELECT
    ST_Neighborhood(rast, 3, 3, 1, 1, false)
FROM ST_SetValues(
    ST_AddBand(
        ST_MakeEmptyRaster(5, 5, -2, 2, 1, -1, 0, 0, 0),
        '8BUI'::text, 1, 0
    ),
    1, 1, 1, ARRAY[
        [0, 1, 1, 1, 1],
        [1, 1, 1, 0, 1],
        [1, 0, 1, 1, 1],
        [1, 1, 1, 1, 0],
        [1, 1, 0, 1, 1]
    ]::double precision[],
    1
) AS rast

st_neighborhood

```

```
-----
{{1,1,0},{0,1,1},{1,1,1}}
```

## Siehe auch

[ST\\_NearestValue](#), [ST\\_Min4ma](#), [ST\\_Max4ma](#), [ST\\_Sum4ma](#), [ST\\_Mean4ma](#), [ST\\_Range4ma](#), [ST\\_Distinct4ma](#), [ST\\_StdDev4ma](#)

## 11.6.12 ST\_SetValue

**ST\_SetValue** — Setzt den Wert für ein Pixel eines Bandes, das über `columnx` und `rowy` festgelegt wird, oder für die Pixel die eine bestimmte Geometrie schneiden, und gibt den veränderten Raster zurück. Die Bandnummerierung beginnt mit 1; wenn die Bandnummer nicht angegeben ist, wird 1 angenommen.

### Synopsis

```
raster ST_SetValue(raster rast, integer bandnum, geometry geom, double precision newvalue);
raster ST_SetValue(raster rast, geometry geom, double precision newvalue);
raster ST_SetValue(raster rast, integer bandnum, integer columnx, integer rowy, double precision newvalue);
raster ST_SetValue(raster rast, integer columnx, integer rowy, double precision newvalue);
```

### Beschreibung

Setzt die Werte bestimmter Pixel eines Bandes auf einen neuen Wert und gibt den veränderten Raster zurück. Die Pixel können über die Rasterzeile und die Rasterspalte, oder über eine Geometrie festgelegt werden. Wenn die Bandnummer nicht angegeben ist, wird 1 angenommen.

Erweiterung: 2.1.0 Die geometrische Variante von `ST_SetValue()` unterstützt nun jeden geometrischen Datentyp, nicht nur `POINT`. Die geometrische Variante ist ein Adapter für die `geomval[]` Variante von `ST_SetValues()`

### Beispiele

```
-- Geometry example
SELECT (foo.geomval).val, ST_AsText(ST_Union((foo.geomval).geom))
FROM (SELECT ST_DumpAsPolygons(
        ST_SetValue(rast,1,
                    ST_Point(3427927.75, 5793243.95),
                    50)
        ) As geomval
FROM dummy_rast
where rid = 2) As foo
WHERE (foo.geomval).val < 250
GROUP BY (foo.geomval).val;
```

val	st_astext
50	POLYGON((3427927.75 5793244,3427927.75 5793243.95,3427927.8 579324 ...
249	POLYGON((3427927.95 5793243.95,3427927.95 5793243.85,3427928 57932 ...

```
-- Store the changed raster --
UPDATE dummy_rast SET rast = ST_SetValue(rast,1, ST_Point(3427927.75, 5793243.95),100)
WHERE rid = 2 ;
```

**Siehe auch**

[ST\\_Value](#), [ST\\_DumpAsPolygons](#)

**11.6.13 ST\_SetValues**

`ST_SetValues` — Gibt einen Raster zurück, der durch das Setzen der Werte eines bestimmten Bandes verändert wurde.

**Synopsis**

```
raster ST_SetValues(raster rast, integer nband, integer columnx, integer rowy, double precision[][] newvalueset, boolean[][] noset=NULL, boolean keepnodata=FALSE);
raster ST_SetValues(raster rast, integer nband, integer columnx, integer rowy, double precision[][] newvalueset, double precision nosetvalue, boolean keepnodata=FALSE);
raster ST_SetValues(raster rast, integer nband, integer columnx, integer rowy, integer width, integer height, double precision newvalue, boolean keepnodata=FALSE);
raster ST_SetValues(raster rast, integer columnx, integer rowy, integer width, integer height, double precision newvalue, boolean keepnodata=FALSE);
raster ST_SetValues(raster rast, integer nband, geomval[] geomvalset, boolean keepnodata=FALSE);
```

**Beschreibung**

Gibt einen veränderten Raster zurück, der durch die Zuweisung neuer Zellwerte auf festgelegte Pixel des ausgewiesenen Rasterbandes entsteht. `columnx` und `rowy` sind bei 1 beginnend indiziert.

Wenn `keepnodata TRUE` ist, dann werden die Pixel mit dem Wert `NODATA` nicht mit dem entsprechenden Wert in `newvalueset` belegt.

Bei der Variante 1 werden die betreffenden Pixel über die Pixelkoordinaten `columnx` und `rowy`, und durch die Größe des Feldes `newvalueset` festgelegt. Über den Parameter `noset` kann verhindert werden, dass bestimmte, in `newvalueset` auftretende Pixelwerte gesetzt werden (da PostgreSQL keine unregelmäßigen Felder/"ragged arrays" zulässt). Siehe das Beispiel mit der Variante 1.

Variante 2 gleicht Variante 1, aber mit einem einfachen `nosetvalue` in Double Precision anstelle des booleschen Feldes `noset`. Elemente in `newvalueset` mit dem Wert `nosetvalue` werden übersprungen. Siehe das Beispiel mit der Variante 2.

Bei der Variante 3 werden die betreffenden Pixel über die Pixelkoordinaten `columnx` und `rowy`, und durch `width` und `height` festgelegt. Siehe das Beispiel mit der Variante 3.

Variante 4 entspricht Variante3 mit der Ausnahme, dass die Pixel des ersten Bandes von `rast` gesetzt werden.

Bei der Variante 5 wird ein Feld von `geomval` verwendet um die Pixel zu bestimmen. Wenn die gesamte Geometrie des Feldes vom Datentyp `POINT` oder `MULTIPOINT` ist, verwendet die Funktion Länge und Breite eines jeden Punktes um den Wert eines Pixels direkt zu setzen. Andernfalls wird die Geometrie in Raster umgewandelt über die dann in einem Schritt iteriert wird. Siehe das Beispiel mit der Variante 5.

Verfügbarkeit: 2.1.0

**Beispiele: Variante 1**

```
/*
The ST_SetValues() does the following...

+ - + - + - +           + - + - + - +
| 1 | 1 | 1 |           | 1 | 1 | 1 |
+ - + - + - +           + - + - + - +
```

```

| 1 | 1 | 1 |   =
>   | 1 | 9 | 9 |
+ - + - + - +   + - + - + - +
| 1 | 1 | 1 |   | 1 | 9 | 9 |
+ - + - + - +   + - + - + - +
*/
SELECT
    (poly).x,
    (poly).y,
    (poly).val
FROM (
SELECT
    ST_PixelAsPolygons(
        ST_SetValues(
            ST_AddBand(
                ST_MakeEmptyRaster(3, 3, 0, 0, 1, -1, 0, 0, 0),
                1, '8BUI', 1, 0
            ),
            1, 2, 2, ARRAY[[9, 9], [9, 9]]::double precision[]
        )
    ) AS poly
) foo
ORDER BY 1, 2;

```

```

x | y | val
---+---+---
1 | 1 | 1
1 | 2 | 1
1 | 3 | 1
2 | 1 | 1
2 | 2 | 9
2 | 3 | 9
3 | 1 | 1
3 | 2 | 9
3 | 3 | 9

```

```

/*
The ST_SetValues() does the following...

+ - + - + - +   + - + - + - +
| 1 | 1 | 1 |   | 9 | 9 | 9 |
+ - + - + - +   + - + - + - +
| 1 | 1 | 1 |   =
>   | 9 |   | 9 |
+ - + - + - +   + - + - + - +
| 1 | 1 | 1 |   | 9 | 9 | 9 |
+ - + - + - +   + - + - + - +
*/
SELECT
    (poly).x,
    (poly).y,
    (poly).val
FROM (
SELECT
    ST_PixelAsPolygons(
        ST_SetValues(
            ST_AddBand(
                ST_MakeEmptyRaster(3, 3, 0, 0, 1, -1, 0, 0, 0),
                1, '8BUI', 1, 0
            ),

```

```

        1, 1, 1, ARRAY[[9, 9, 9], [9, NULL, 9], [9, 9, 9]]::double precision[][])
    ) AS poly
) foo
ORDER BY 1, 2;

```

x	y	val
1	1	9
1	2	9
1	3	9
2	1	9
2	2	
2	3	9
3	1	9
3	2	9
3	3	9

```

/*
The ST_SetValues() does the following...

```

+ - + - + - +		+ - + - + - +
1   1   1		9   9   9
+ - + - + - +		+ - + - + - +
1   1   1	=>	1     9
+ - + - + - +		+ - + - + - +
1   1   1		9   9   9
+ - + - + - +		+ - + - + - +

```

*/
SELECT
    (poly).x,
    (poly).y,
    (poly).val
FROM (
SELECT
    ST_PixelAsPolygons(
        ST_SetValues(
            ST_AddBand(
                ST_MakeEmptyRaster(3, 3, 0, 0, 1, -1, 0, 0, 0),
                1, '8BUI', 1, 0
            ),
            1, 1, 1,
            ARRAY[[9, 9, 9], [9, NULL, 9], [9, 9, 9]]::double precision[][],
            ARRAY[[false], [true]]::boolean[][]
        )
    ) AS poly
) foo
ORDER BY 1, 2;

```

x	y	val
1	1	9
1	2	1
1	3	9
2	1	9
2	2	
2	3	9
3	1	9
3	2	9
3	3	9

```

/*
The ST_SetValues() does the following...

+ - + - + - +           + - + - + - +
|  | 1 | 1 |           |  | 9 | 9 |
+ - + - + - +           + - + - + - +
| 1 | 1 | 1 |   =>   | 1 |   | 9 |
+ - + - + - +           + - + - + - +
| 1 | 1 | 1 |           | 9 | 9 | 9 |
+ - + - + - +           + - + - + - +
*/
SELECT
  (poly).x,
  (poly).y,
  (poly).val
FROM (
SELECT
  ST_PixelAsPolygons(
    ST_SetValues(
      ST_SetValue(
        ST_AddBand(
          ST_MakeEmptyRaster(3, 3, 0, 0, 1, -1, 0, 0, 0),
          1, '8BUI', 1, 0
        ),
        1, 1, 1, NULL
      ),
      1, 1, 1,
      ARRAY[[9, 9, 9], [9, NULL, 9], [9, 9, 9]]::double precision[][],
      ARRAY[[false], [true]]::boolean[][],
      TRUE
    )
  ) AS poly
) foo
ORDER BY 1, 2;

 x | y | val
---+---+-----
 1 | 1 |
 1 | 2 |  1
 1 | 3 |  9
 2 | 1 |  9
 2 | 2 |
 2 | 3 |  9
 3 | 1 |  9
 3 | 2 |  9
 3 | 3 |  9

```

### Beispiele: Variante 2

```

/*
The ST_SetValues() does the following...

+ - + - + - +           + - + - + - +
| 1 | 1 | 1 |           | 1 | 1 | 1 |
+ - + - + - +           + - + - + - +
| 1 | 1 | 1 |   =>   | 1 | 9 | 9 |
+ - + - + - +           + - + - + - +
| 1 | 1 | 1 |           | 1 | 9 | 9 |
+ - + - + - +           + - + - + - +

```



```

*/
SELECT
    (poly).x,
    (poly).y,
    (poly).val
FROM (
SELECT
    ST_PixelAsPolygons(
        ST_SetValues(
            ST_AddBand(
                ST_MakeEmptyRaster(3, 3, 0, 0, 1, -1, 0, 0, 0),
                1, '8BUI', 1, 0
            ),
            1, 1, 1, ARRAY[[-1, -1, -1], [-1, 9, 9], [-1, 9, 9]]::double precision[][], -1
        )
    ) AS poly
) foo
ORDER BY 1, 2;

```

```

x | y | val
---+---+-----
1 | 1 | 1
1 | 2 | 1
1 | 3 | 1
2 | 1 | 1
2 | 2 | 9
2 | 3 | 9
3 | 1 | 1
3 | 2 | 9
3 | 3 | 9

```

/\*  
This example is like the previous one. Instead of nosetvalue = -1, nosetvalue = NULL

The ST\_SetValues() does the following...

```

+ - + - + - +           + - + - + - +
| 1 | 1 | 1 |           | 1 | 1 | 1 |
+ - + - + - +           + - + - + - +
| 1 | 1 | 1 |           => | 1 | 9 | 9 |
+ - + - + - +           + - + - + - +
| 1 | 1 | 1 |           | 1 | 9 | 9 |
+ - + - + - +           + - + - + - +

```

```

*/
SELECT
    (poly).x,
    (poly).y,
    (poly).val
FROM (
SELECT
    ST_PixelAsPolygons(
        ST_SetValues(
            ST_AddBand(
                ST_MakeEmptyRaster(3, 3, 0, 0, 1, -1, 0, 0, 0),
                1, '8BUI', 1, 0
            ),
            1, 1, 1, ARRAY[[NULL, NULL, NULL], [NULL, 9, 9], [NULL, 9, 9]]::double ←
                precision[][], NULL::double precision
        )
    ) AS poly
) foo
ORDER BY 1, 2;

```

```

x | y | val
---+---+---
1 | 1 | 1
1 | 2 | 1
1 | 3 | 1
2 | 1 | 1
2 | 2 | 9
2 | 3 | 9
3 | 1 | 1
3 | 2 | 9
3 | 3 | 9

```

### Beispiele: Variante 3

```

/*
The ST_SetValues() does the following...

+ - + - + - +          + - + - + - +
| 1 | 1 | 1 |          | 1 | 1 | 1 |
+ - + - + - +          + - + - + - +
| 1 | 1 | 1 |          => | 1 | 9 | 9 |
+ - + - + - +          + - + - + - +
| 1 | 1 | 1 |          | 1 | 9 | 9 |
+ - + - + - +          + - + - + - +
*/
SELECT
    (poly).x,
    (poly).y,
    (poly).val
FROM (
SELECT
    ST_PixelAsPolygons(
        ST_SetValues(
            ST_AddBand(
                ST_MakeEmptyRaster(3, 3, 0, 0, 1, -1, 0, 0, 0),
                1, '8BUI', 1, 0
            ),
            1, 2, 2, 2, 2, 9
        )
    ) AS poly
) foo
ORDER BY 1, 2;

```

```

x | y | val
---+---+---
1 | 1 | 1
1 | 2 | 1
1 | 3 | 1
2 | 1 | 1
2 | 2 | 9
2 | 3 | 9
3 | 1 | 1
3 | 2 | 9
3 | 3 | 9

```

```

/*
The ST_SetValues() does the following...

```

```

+ - + - + - +          + - + - + - +

```

```

| 1 | 1 | 1 | | | 1 | 1 | 1 |
+ - + - + - +   => + - + - + - +
| 1 |   | 1 |   | 1 | 9 |
+ - + - + - +   + - + - + - +
| 1 | 1 | 1 |   | 1 | 9 | 9 |
+ - + - + - +   + - + - + - +
*/
SELECT
    (poly).x,
    (poly).y,
    (poly).val
FROM (
SELECT
    ST_PixelAsPolygons(
        ST_SetValues(
            ST_SetValue(
                ST_AddBand(
                    ST_MakeEmptyRaster(3, 3, 0, 0, 1, -1, 0, 0, 0),
                    1, '8BUI', 1, 0
                ),
                1, 2, 2, NULL
            ),
            1, 2, 2, 2, 2, 9, TRUE
        )
    ) AS poly
) foo
ORDER BY 1, 2;

```

```

x | y | val
---+---+-----
1 | 1 | 1
1 | 2 | 1
1 | 3 | 1
2 | 1 | 1
2 | 2 |
2 | 3 | 9
3 | 1 | 1
3 | 2 | 9
3 | 3 | 9

```

### Beispiele: Variante 5

```

WITH foo AS (
    SELECT 1 AS rid, ST_AddBand(ST_MakeEmptyRaster(5, 5, 0, 0, 1, -1, 0, 0, 0), 1, '8BUI', ←
        0, 0) AS rast
), bar AS (
    SELECT 1 AS gid, 'SRID=0;POINT(2.5 -2.5)::geometry geom UNION ALL
    SELECT 2 AS gid, 'SRID=0;POLYGON((1 -1, 4 -1, 4 -4, 1 -4, 1 -1))::geometry geom UNION ←
        ALL
    SELECT 3 AS gid, 'SRID=0;POLYGON((0 0, 5 0, 5 -1, 1 -1, 1 -4, 0 -4, 0 0))::geometry ←
        geom UNION ALL
    SELECT 4 AS gid, 'SRID=0;MULTIPOINT(0 0, 4 4, 4 -4)::geometry
)
SELECT
    rid, gid, ST_DumpValues(ST_SetValue(rast, 1, geom, gid))
FROM foo t1
CROSS JOIN bar t2
ORDER BY rid, gid;

```

```

rid | gid | st_dumpvalues

```



```

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](#)

**11.6.14 ST\_DumpValues**

ST\_DumpValues — Gibt die Werte eines bestimmten Bandes als 2-dimensionales Feld aus.

**Synopsis**

```

setof record ST_DumpValues( raster rast , integer[] nband=NULL , boolean exclude_nodata_value=true );
double precision[][] ST_DumpValues( raster rast , integer nband , boolean exclude_nodata_value=true );

```

**Beschreibung**

Gibt die Werte eines bestimmten Bandes als 2-dimensionales Feld (der erste Index entspricht der Zeile, der zweite der Spalte) aus. Wenn nband NULL oder nicht gegeben ist, werden alle Rasterbänder abgearbeitet.

Verfügbarkeit: 2.1.0

**Beispiele**

```

WITH foo AS (
  SELECT ST_AddBand(ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(3, 3, 0, 0, 1, -1, 0, 0, 0), ←
    1, '8BUI'::text, 1, 0), 2, '32BF'::text, 3, -9999), 3, '16BSI', 0, 0) AS rast
)
SELECT
  (ST_DumpValues(rast)).*
FROM foo;

```

nband	valarray
1	{{1,1,1},{1,1,1},{1,1,1}}
2	{{3,3,3},{3,3,3},{3,3,3}}
3	{{NULL,NULL,NULL},{NULL,NULL,NULL},{NULL,NULL,NULL}}

(3 rows)

```

WITH foo AS (
  SELECT ST_AddBand(ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(3, 3, 0, 0, 1, -1, 0, 0, 0), ←
    1, '8BUI'::text, 1, 0), 2, '32BF'::text, 3, -9999), 3, '16BSI', 0, 0) AS rast
)
SELECT
  (ST_DumpValues(rast, ARRAY[3, 1])).*
FROM foo;

```

```

nband |                               valarray
-----+-----
      3 | {{NULL,NULL,NULL},{NULL,NULL,NULL},{NULL,NULL,NULL}}
      1 | {{1,1,1},{1,1,1},{1,1,1}}
(2 rows)

```

```

WITH foo AS (
  SELECT ST_SetValue(ST_AddBand(ST_MakeEmptyRaster(3, 3, 0, 0, 1, -1, 0, 0, 0), 1, '8BUI ←
    ', 1, 0), 1, 2, 5) AS rast
)
SELECT
  (ST_DumpValues(rast, 1))[2][1]
FROM foo;

 st_dumpvalues
-----
              5
(1 row)

```

**Siehe auch**

[ST\\_Value](#), [ST\\_SetValue](#), [ST\\_SetValues](#)

**11.6.15 ST\_PixelOfValue**

`ST_PixelOfValue` — Gibt die columnx- und rowy-Koordinaten jener Pixel aus, deren Zellwert gleich dem gesuchten Wert ist.

**Synopsis**

```

setof record ST_PixelOfValue( raster rast , integer nband , double precision[] search , boolean exclude_nodata_value=true );
setof record ST_PixelOfValue( raster rast , double precision[] search , boolean exclude_nodata_value=true );
setof record ST_PixelOfValue( raster rast , integer nband , double precision search , boolean exclude_nodata_value=true );
setof record ST_PixelOfValue( raster rast , double precision search , boolean exclude_nodata_value=true );

```

**Beschreibung**

Gibt die columnx- und rowy-Koordinaten jener Pixel aus, deren Zellwert gleich dem gesuchten Wert ist. Wenn kein Band angegeben ist, wird Band 1 angenommen.

Verfügbarkeit: 2.1.0

**Beispiele**

```

SELECT
  (pixels).*
FROM (
  SELECT
    ST_PixelOfValue(
      ST_SetValue(
        ST_SetValue(
          ST_SetValue(
            ST_SetValue(
              ST_SetValue(
                ST_AddBand(
                  ST_MakeEmptyRaster(5, 5, -2, 2, 1, -1, 0, 0, 0),

```

```

        '8BUI'::text, 1, 0
    ),
    1, 1, 0
),
    2, 3, 0
),
    3, 5, 0
),
    4, 2, 0
),
    5, 4, 255
)
, 1, ARRAY[1, 255]) AS pixels
) AS foo

```

val	x	y
1	1	2
1	1	3
1	1	4
1	1	5
1	2	1
1	2	2
1	2	4
1	2	5
1	3	1
1	3	2
1	3	3
1	3	4
1	4	1
1	4	3
1	4	4
1	4	5
1	5	1
1	5	2
1	5	3
255	5	4
1	5	5

## 11.7 Raster Editoren

### 11.7.1 ST\_SetGeoReference

`ST_SetGeoReference` — Georeferenziert einen Raster über 6 Parameter in einem einzigen Aufruf. Die Zahlen müssen durch Leerzeichen getrennt sein. Die Funktion akzeptiert die Eingabe im Format von 'GDAL' und von 'ESRI'. Der Standardwert ist GDAL.

#### Synopsis

```

raster ST_SetGeoReference(raster rast, text georefcoords, text format=GDAL);
raster ST_SetGeoReference(raster rast, double precision upperleftx, double precision upperlefty, double precision scalex, double precision scaley, double precision skewx, double precision skewy);

```

#### 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 skewx scaley upperleftx upperlefty
```

ESRI:

```
scalex skewx scaley upperleftx + scalex*0.5 upperlefty + scaley*0.5
```



**Note**

Wenn der Raster out-db Bänder aufweist, dann kann eine Änderung der Georeferenzierung zu einem unkorrekten Zugriff auf die extern gespeicherten Daten des Bandes führen.

Erweiterung: 2.1.0 ST\_SetGeoReference(raster, double precision, ...) Variante hinzugefügt

**Beispiele**

```
WITH foo AS (
  SELECT ST_MakeEmptyRaster(5, 5, 0, 0, 1, -1, 0, 0, 0) AS rast
)
SELECT
  0 AS rid, (ST_Metadata(rast)).*
FROM foo
UNION ALL
SELECT
  1, (ST_Metadata(ST_SetGeoReference(rast, '10 0 0 -10 0.1 0.1', 'GDAL'))).*
FROM foo
UNION ALL
SELECT
  2, (ST_Metadata(ST_SetGeoReference(rast, '10 0 0 -10 5.1 -4.9', 'ESRI'))).*
FROM foo
UNION ALL
SELECT
  3, (ST_Metadata(ST_SetGeoReference(rast, 1, 1, 10, -10, 0.001, 0.001))).*
FROM foo
```

rid	upperleftx	skewx	srid	numbands	upperlefty	width	height	scalex	scaley	skewy
0	0	0	0	0	0	5	5	1	-1	0
1	0	0	0.1	0	0.1	5	5	10	-10	0
2	0.09999999999999996	0	0.09999999999999996	0	0.09999999999999996	5	5	10	-10	0
3	0.001	0	1	0	1	5	5	10	-10	0.001

**Siehe auch**

[ST\\_GeoReference](#), [ST\\_ScaleX](#), [ST\\_ScaleY](#), [ST\\_UpperLeftX](#), [ST\\_UpperLeftY](#)



## 11.7.2 ST\_SetRotation

ST\_SetRotation — Bestimmt die Rotation des Rasters in Radiant.

### Synopsis

raster **ST\_SetRotation**(raster rast, float8 rotation);

### 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](#)

## 11.7.3 ST\_SetScale

ST\_SetScale — Setzt die X- und Y-Größe der Pixel in den Einheiten des Koordinatenreferenzsystems. Entweder eine Zahl pro Pixel oder Breite und Höhe.

### Synopsis

raster **ST\_SetScale**(raster rast, float8 xy);  
 raster **ST\_SetScale**(raster rast, float8 x, float8 y);

### 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](#)

## 11.7.4 ST\_SetSkew

ST\_SetSkew — Setzt den georeferenzierten X- und Y-Versatz (oder den Rotationsparameter). Wenn nur ein Wert übergeben wird, werden X und Y auf den selben Wert gesetzt.

### Synopsis

```
raster ST_SetSkew(raster rast, float8 skewxy);
raster ST_SetSkew(raster rast, float8 skewx, float8 skewy);
```

### Beschreibung

Setzt den georeferenzierten X- und Y-Versatz (oder den Rotationsparameter). Wenn nur ein Wert übergeben wird, werden X und Y auf den selben Wert gesetzt. Siehe [World File](#) für weitere Details.

### Beispiele

```
-- Example 1
UPDATE dummy_rast SET rast = ST_SetSkew(rast,1,2) WHERE rid = 1;
SELECT rid, ST_SkewX(rast) As skewx, ST_SkewY(rast) As skewy,
  ST_GeoReference(rast) as georef
FROM dummy_rast WHERE rid = 1;
```

```

rid | skewx | skewy | georef
-----+-----+-----+-----
  1 |      1 |      2 | 2.0000000000
      : 2.0000000000
      : 1.0000000000
      : 3.0000000000
      : 0.5000000000
      : 0.5000000000

```

```

-- Example 2 set both to same number:
UPDATE dummy_rast SET rast = ST_SetSkew(rast,0) WHERE rid = 1;
SELECT rid, ST_SkewX(rast) As skewx, ST_SkewY(rast) As skewy,
       ST_GeoReference(rast) as georef
FROM dummy_rast WHERE rid = 1;

```

```

rid | skewx | skewy | georef
-----+-----+-----+-----
  1 |      0 |      0 | 2.0000000000
      : 0.0000000000
      : 0.0000000000
      : 3.0000000000
      : 0.5000000000
      : 0.5000000000

```

## Siehe auch

[ST\\_GeoReference](#), [ST\\_SetGeoReference](#), [ST\\_SkewX](#), [ST\\_SkewY](#)

### 11.7.5 ST\_SetSRID

**ST\_SetSRID** — Setzt die SRID eines Rasters auf einen bestimmten Ganzzahlwert. Die SRID wird in der Tabelle "spatial\_ref\_sys" definiert.

#### Synopsis

```
raster ST_SetSRID(raster rast, integer srid);
```

#### Beschreibung

Weist der SRID des Rasters einen bestimmten Ganzzahlwert zu.



#### Note

Diese Funktion führt keine Koordinatentransformation des Rasters durch - sie setzt nur die Metadaten, welche das Koordinatenreferenzsystem definieren, in dem der Raster vorliegt. Nützlich für spätere Koordinatentransformationen.

## Siehe auch

Section [4.5](#), [ST\\_SRID](#)

## 11.7.6 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](#)

## 11.7.7 ST\_Resample

`ST_Resample` — Skaliert einen Raster mit einem bestimmten Algorithmus, neuen Dimensionen, einer beliebigen Gitterecke und über Parameter zur Georeferenzierung des Rasters, die angegeben oder von einem anderen Raster übernommen werden können.

### Synopsis

```
raster ST_Resample(raster rast, integer width, integer height, double precision gridx=NULL, double precision gridy=NULL,
double precision skewx=0, double precision skewy=0, text algorithm=NearestNeighbor, double precision maxerr=0.125);
raster ST_Resample(raster rast, double precision scalex=0, double precision scaley=0, double precision gridx=NULL, double
precision gridy=NULL, double precision skewx=0, double precision skewy=0, text algorithm=NearestNeighbor, double preci-
sion maxerr=0.125);
raster ST_Resample(raster rast, raster ref, text algorithm=NearestNeighbor, double precision maxerr=0.125, boolean usescale=true);
raster ST_Resample(raster rast, raster ref, boolean usescale, text algorithm=NearestNeighbor, double precision maxerr=0.125);
```

### Beschreibung

Skaliert einen Raster mit einem bestimmten Algorithmus, neuen Dimensionen (width & height), einer Gitterecke (gridx & gridy) und über Parameter zur Georeferenzierung des Rasters (scalex, scaley, skewx & skewy), die angegeben oder von einem anderen Raster übernommen werden können. Wenn Sie einen Referenzraster verwenden, müssen beide Raster die selbe SRID haben.

Die neuen Pixelwerte werden mit einem der folgenden Resampling-Algorithmen berechnet:

- NearestNeighbor (englische oder amerikanische Schreibweise)
- Bilinear
- Kubisch
- CubicSpline

- Lanczos
- Max
- Min

Die Standardeinstellung ist NearestNeighbor, die am schnellsten ist, aber die schlechteste Interpolation ergibt.

Wenn `maxerr` nicht angegeben ist, wird der maximale Fehler mit 0.125 Prozent angesetzt.



#### Note

Siehe [GDAL Warp resampling methods](#) für mehr Details.

Verfügbarkeit: 2.0.0 benötigt GDAL 1.6.1+

Verbessert: 3.4.0 Max und Min Resampling Optionen hinzugefügt

### Beispiele

```
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](#)

## 11.7.8 ST\_Rescale

`ST_Rescale` — Neuabtastung eines Rasters, indem nur die Skala (oder Pixelgröße) angepasst wird. Die neuen Pixelwerte werden mit den Algorithmen NearestNeighbor (englische oder amerikanische Schreibweise), Bilinear, Cubic, CubicSpline, Lanczos, Max oder Min resampling berechnet. Die Voreinstellung ist NearestNeighbor.

### Synopsis

```
raster ST_Rescale(raster rast, double precision scalexy, text algorithm=NearestNeighbor, double precision maxerr=0.125);
raster ST_Rescale(raster rast, double precision scalex, double precision scaley, text algorithm=NearestNeighbor, double precision maxerr=0.125);
```

## Beschreibung

Sie können ein Raster neu abtasten, indem Sie nur den Maßstab (oder die Pixelgröße) anpassen. Die neuen Pixelwerte werden mit einem der folgenden Resampling-Algorithmen berechnet:

- NearestNeighbor (englische oder amerikanische Schreibweise)
- Bilinear
- Kubisch
- CubicSpline
- Lanczos
- Max
- Min

Die Standardeinstellung ist NearestNeighbor, die am schnellsten ist, aber die schlechteste Interpolation ergibt.

`scalex` und `scaley` bestimmen die neue Pixelgröße. Der Wert von "scaley" muss oftmals negativ sein, um einen ordnungsgemäß ausgerichteten Raster zu erhalten.

Wenn "scalex" oder "scaley" teilerfremd zur Breite oder Höhe des Rasters sind, dann wird der Zielraster auf die Ausdehnung des Ausgangsrasters erweitert. Um die exakte Ausdehnung des Ausgangsrasters sicher zu erhalten, siehe [ST\\_Resize](#)

`maxerr` ist der Schwellenwert bei der Näherungstransformation des Skalierungsalgorithmus (in Pixeleinheiten). Ein Standardwert von 0.125 wird verwendet, wenn kein `maxerr` angegeben wird. Dies ist dergleiche Wert, welcher von `gdalwarp` verwendet wird. Wenn der Wert auf Null gesetzt ist, wird keine Annäherung ausgeführt.



### Note

Siehe [GDAL Warp resampling methods](#) für mehr Details.



### Note

`ST_Rescale` unterscheidet sich von [ST\\_SetScale](#) darin, dass `ST_SetScale` den Raster nicht skaliert um mit der Ausdehnung des Ausgangsrasters übereinzustimmen. `ST_SetScale` ändert lediglich die Metadaten (oder die Georeferenz) des Rasters, um eine ursprünglich falsch angegebene Skalierung zu korrigieren. `ST_Rescale` berechnet die Breite und Höhe eines Rasters so, dass er mit der geographischen Ausdehnung des Ausgangsraster übereinstimmt. `ST_SetScale` verändert weder die Breite noch die Höhe des Rasters.

Verfügbarkeit: 2.0.0 benötigt GDAL 1.6.1+

Verbessert: 3.4.0 Max und Min Resampling Optionen hinzugefügt

Änderung: 2.1.0 Funktioniert jetzt auch mit Raster ohne SRID

## Beispiele

Ein einfaches Beispiel, das die Pixelgröße eines Raster von 0.001 Grad auf 0.0015 Grad ändert.

```
-- the original raster pixel size
SELECT ST_PixelWidth(ST_AddBand(ST_MakeEmptyRaster(100, 100, 0, 0, 0.001, -0.001, 0, 0, ↵
    4269), '8BUI'::text, 1, 0)) width
width
-----
```

```

0.001

-- the rescaled raster raster pixel size
SELECT ST_PixelWidth(ST_Rescale(ST_AddBand(ST_MakeEmptyRaster(100, 100, 0, 0, 0.001, ←
    -0.001, 0, 0, 4269), '8BUI'::text, 1, 0), 0.0015)) width

    width
-----
0.0015

```

**Siehe auch**

[ST\\_Resize](#), [ST\\_Resample](#), [ST\\_SetScale](#), [ST\\_ScaleX](#), [ST\\_ScaleY](#), [ST\\_Transform](#)

**11.7.9 ST\_Reskew**

`ST_Reskew` — Skaliert einen Raster, indem lediglich der Versatz (oder Rotationsparameter) angepasst wird. Neue Pixelwerte werden über `NearestNeighbor`, `bilinear`, `kubisch`, `CubicSpline` oder mit dem `Lanczos`-Filter errechnet. Die Standardeinstellung ist `NearestNeighbor`.

**Synopsis**

```

raster ST_Reskew(raster rast, double precision skewxy, text algorithm=NearestNeighbor, double precision maxerr=0.125);
raster ST_Reskew(raster rast, double precision skewx, double precision skewy, text algorithm=NearestNeighbor, double precision maxerr=0.125);

```

**Beschreibung**

Skaliert einen Raster, indem lediglich der Versatz (oder Rotationsparameter) angepasst wird. Neue Pixelwerte werden über `NearestNeighbor`, `bilinear`, `kubisch`, `CubicSpline` oder mit dem `Lanczos`-Filter errechnet. Die Standardeinstellung "NearestNeighbor" ist am schnellsten, ergibt aber die schlechteste Interpolation.

`skewx` und `skewy` legen den neuen Versatz fest.

Die Ausdehnung des Zielrasters erfasst die Ausdehnung des Ausgangsrasters.

Wenn `maxerr` nicht angegeben ist, wird der maximale Fehler mit 0.125 Prozent angesetzt.

**Note**

Siehe [GDAL Warp resampling methods](#) für mehr Details.

**Note**

`ST_Reskew` unterscheidet sich von [ST\\_SetSkew](#) darin, dass `ST_SetSkew` den Raster nicht skaliert um mit der Ausdehnung des Ausgangsrasters übereinzustimmen. `ST_SetSkew` ändert lediglich die Metadaten (oder die Georeferenz) des Rasters, um einen ursprünglich falsch angegebenen Versatz zu korrigieren. `ST_Reskew` ergibt einen Raster mit geänderter Breite und Höhe, da die Berechnung so durchgeführt wird, dass die geographischen Ausdehnung des Zielrasters mit der des Ausgangsrasters übereinstimmt. `ST_SetSkew` verändert weder die Breite noch die Höhe des Rasters.

Verfügbarkeit: 2.0.0 benötigt GDAL 1.6.1+

Änderung: 2.1.0 Funktioniert jetzt auch mit Raster ohne SRID

## Beispiele

Ein einfaches Beispiel, das den Versatz eines Rasters von 0.0 auf 0.0015 ändert.

```
-- the original raster non-rotated
SELECT ST_Rotation(ST_AddBand(ST_MakeEmptyRaster(100, 100, 0, 0, 0.001, -0.001, 0, 0, 4269) ←
, '8BUI'::text, 1, 0));

-- result
0

-- the reskewed raster raster rotation
SELECT ST_Rotation(ST_Reskew(ST_AddBand(ST_MakeEmptyRaster(100, 100, 0, 0, 0.001, -0.001, ←
0, 0, 4269), '8BUI'::text, 1, 0), 0.0015));

-- result
-0.982793723247329
```

## Siehe auch

[ST\\_Resample](#), [ST\\_Rescale](#), [ST\\_SetSkew](#), [ST\\_SetRotation](#), [ST\\_SkewX](#), [ST\\_SkewY](#), [ST\\_Transform](#)

### 11.7.10 ST\_SnapToGrid

`ST_SnapToGrid` — Skaliert einen Raster durch Fangen an einem Führungsgitter. Neue Pixelwerte werden über `NearestNeighbor`, `bilinear`, `kubisch`, `CubicSpline` oder mit dem `Lanczos`-Filter errechnet. Die Standardeinstellung ist `NearestNeighbor`.

## Synopsis

```
raster ST_SnapToGrid(raster rast, double precision gridx, double precision gridy, text algorithm=NearestNeighbor, double pre-
cision maxerr=0.125, double precision scalex=DEFAULT 0, double precision scaley=DEFAULT 0);
raster ST_SnapToGrid(raster rast, double precision gridx, double precision gridy, double precision scalex, double precision
scaley, text algorithm=NearestNeighbor, double precision maxerr=0.125);
raster ST_SnapToGrid(raster rast, double precision gridx, double precision gridy, double precision scalexy, text algorithm=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.

```
-- the original raster upper left X
SELECT ST_UpperLeftX(ST_AddBand(ST_MakeEmptyRaster(10, 10, 0, 0, 0.001, -0.001, 0, 0, 4269) ←
, '8BUI'::text, 1, 0));
-- result
0

-- the upper left of raster after snapping
SELECT ST_UpperLeftX(ST_SnapToGrid(ST_AddBand(ST_MakeEmptyRaster(10, 10, 0, 0, 0.001, ←
-0.001, 0, 0, 4269), '8BUI'::text, 1, 0), 0.0002, 0.0002));

--result
-0.0008
```

**Siehe auch**

[ST\\_Resample](#), [ST\\_Rescale](#), [ST\\_UpperLeftX](#), [ST\\_UpperLeftY](#)

**11.7.11 ST\_Resize**

`ST_Resize` — Ändert die Zellgröße - width/height - eines Rasters

**Synopsis**

```
raster ST_Resize(raster rast, integer width, integer height, text algorithm=NearestNeighbor, double precision maxerr=0.125);
raster ST_Resize(raster rast, double precision percentwidth, double precision percentheight, text algorithm=NearestNeighbor,
double precision maxerr=0.125);
raster ST_Resize(raster rast, text width, text height, text algorithm=NearestNeighbor, double precision maxerr=0.125);
```

**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	↔
1	0	0	500	500	1	-1	0	0	0	↔
2	0	0	500	100	1	-1	0	0	0	↔
3	0	0	250	900	1	-1	0	0	0	↔

(3 rows)

**Siehe auch**

[ST\\_Resample](#), [ST\\_Rescale](#), [ST\\_Reskew](#), [ST\\_SnapToGrid](#)

**11.7.12 ST\_Transform**

ST\_Transform — Projiziert einen Raster von einem bekannten Koordinatenreferenzsystem in ein anderes bekanntes Koordinatenreferenzsystem um. Die Optionen für die Skalierung sind NearestNeighbor, Bilinear, Cubisch, CubicSpline und der Lanczos-Filter, die Standardeinstellung ist NearestNeighbor.

## Synopsis

```
raster ST_Transform(raster rast, integer srid, text algorithm=NearestNeighbor, double precision maxerr=0.125, double precision scalex, double precision scaley);
raster ST_Transform(raster rast, integer srid, double precision scalex, double precision scaley, text algorithm=NearestNeighbor, double precision maxerr=0.125);
raster ST_Transform(raster rast, raster alignto, text algorithm=NearestNeighbor, double precision maxerr=0.125);
```

## 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



Wenn Sie feststellen, dass Ihre Transformationsunterstützung nicht richtig funktioniert, müssen Sie möglicherweise die Umgebungsvariable PROJSO auf die .so- oder .dll-Projektionsbibliothek setzen, die Ihr PostGIS verwendet. Sie muss nur den Namen der Datei enthalten. Unter Windows würden Sie zum Beispiel in der Systemsteuerung -> System -> Umgebungsvariablen eine Systemvariable namens PROJSO hinzufügen und sie auf `libproj.dll` setzen (wenn Sie proj 4.6.1 verwenden). Nach dieser Änderung müssen Sie Ihren PostgreSQL-Dienst/Daemon neu starten.



### Warning

Bei der Umwandlung einer Abdeckung von Kacheln sollten Sie fast immer ein Referenzraster verwenden, um sicherzustellen, dass die Kacheln gleich ausgerichtet sind und keine Lücken aufweisen, wie im Beispiel gezeigt: Variante 3.

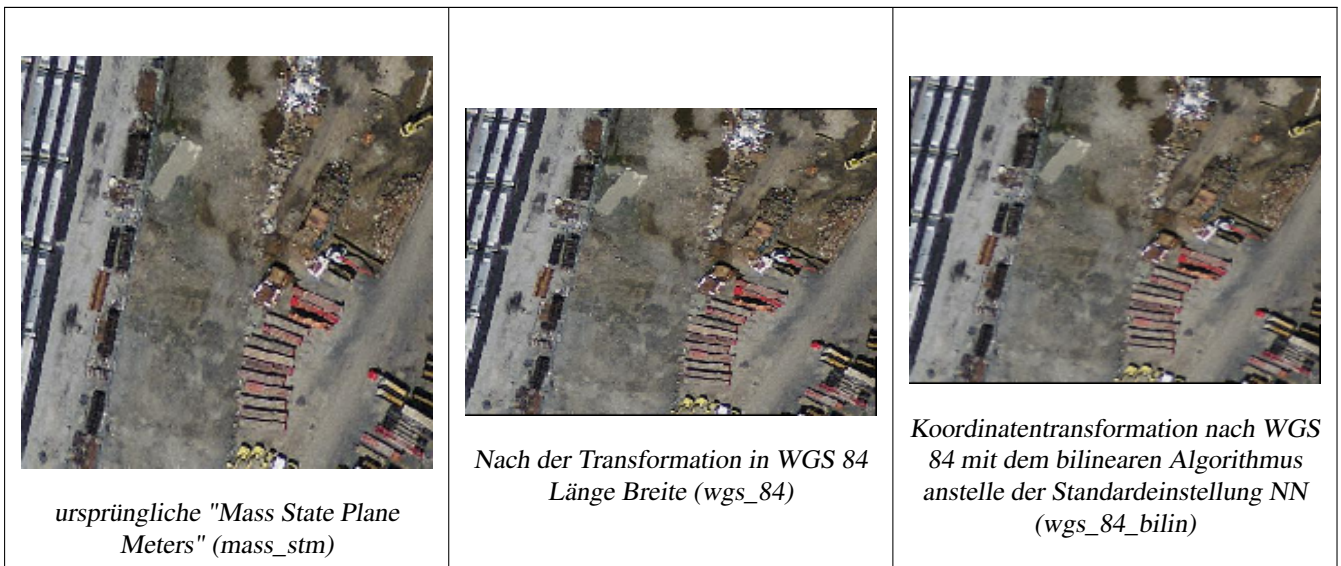
Verfügbarkeit: 2.0.0 benötigt GDAL 1.6.1+

Erweiterung: 2.1.0 Variante ST\_Transform(rast, alignto) hinzugefügt

## Beispiele

```
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, alignto)` gezeigt

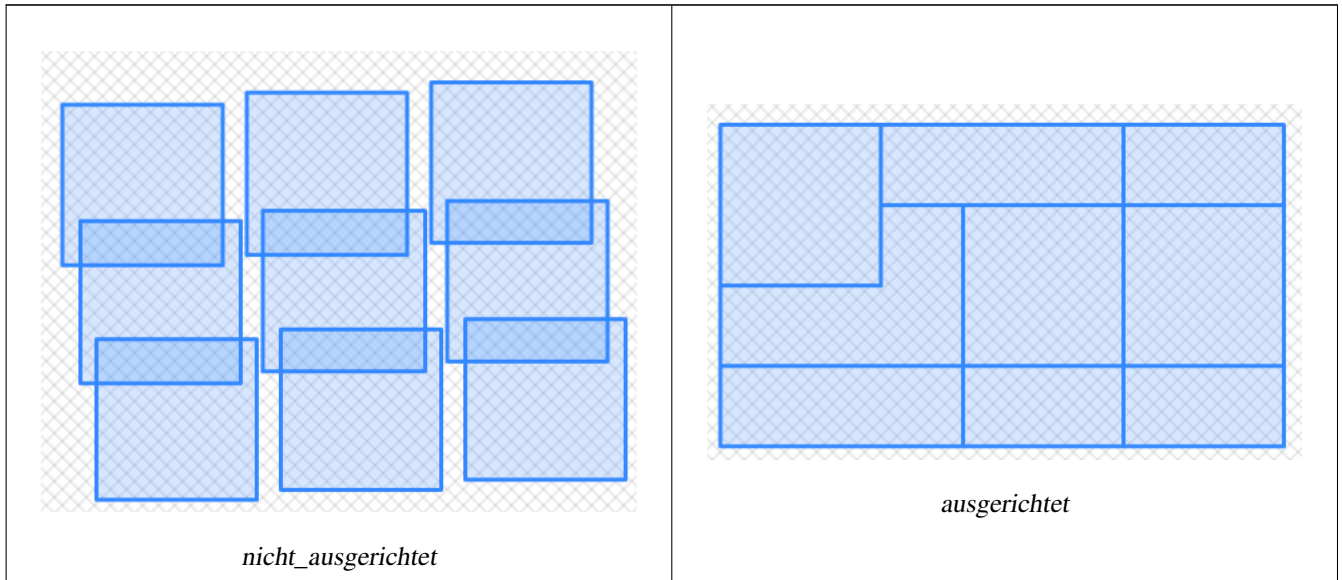
```
WITH foo AS (
  SELECT 0 AS rid, ST_AddBand(ST_MakeEmptyRaster(2, 2, -500000, 600000, 100, -100, 0, 0, ←
    2163), 1, '16BUI', 1, 0) AS rast UNION ALL
  SELECT 1, ST_AddBand(ST_MakeEmptyRaster(2, 2, -499800, 600000, 100, -100, 0, 0, 2163), ←
    1, '16BUI', 2, 0) AS rast UNION ALL
  SELECT 2, ST_AddBand(ST_MakeEmptyRaster(2, 2, -499600, 600000, 100, -100, 0, 0, 2163), ←
    1, '16BUI', 3, 0) AS rast UNION ALL

  SELECT 3, ST_AddBand(ST_MakeEmptyRaster(2, 2, -500000, 599800, 100, -100, 0, 0, 2163), ←
    1, '16BUI', 10, 0) AS rast UNION ALL
  SELECT 4, ST_AddBand(ST_MakeEmptyRaster(2, 2, -499800, 599800, 100, -100, 0, 0, 2163), ←
    1, '16BUI', 20, 0) AS rast UNION ALL
  SELECT 5, ST_AddBand(ST_MakeEmptyRaster(2, 2, -499600, 599800, 100, -100, 0, 0, 2163), ←
    1, '16BUI', 30, 0) AS rast UNION ALL

  SELECT 6, ST_AddBand(ST_MakeEmptyRaster(2, 2, -500000, 599600, 100, -100, 0, 0, 2163), ←
    1, '16BUI', 100, 0) AS rast UNION ALL
  SELECT 7, ST_AddBand(ST_MakeEmptyRaster(2, 2, -499800, 599600, 100, -100, 0, 0, 2163), ←
    1, '16BUI', 200, 0) AS rast UNION ALL
  SELECT 8, ST_AddBand(ST_MakeEmptyRaster(2, 2, -499600, 599600, 100, -100, 0, 0, 2163), ←
    1, '16BUI', 300, 0) AS rast
), bar AS (
  SELECT
    ST_Transform(rast, 4269) AS alignto
  FROM foo
  LIMIT 1
), baz AS (
  SELECT
    rid,
    rast,
    ST_Transform(rast, 4269) AS not_aligned,
    ST_Transform(rast, alignto) AS aligned
  FROM foo
  CROSS JOIN bar
)
```

```
SELECT
  ST_SameAlignment(rast) AS rast,
  ST_SameAlignment(not_aligned) AS not_aligned,
  ST_SameAlignment(aligned) AS aligned
FROM baz
```

rast	not_aligned	aligned
t	f	t



#### Siehe auch

[ST\\_Transform](#), [ST\\_SetSRID](#)

## 11.8 Editoren für Rasterbänder

### 11.8.1 ST\_SetBandNoDataValue

`ST_SetBandNoDataValue` — Setzt den NODATA Wert eines Bandes. Wenn kein Band angegeben ist, wird Band 1 angenommen. Falls ein Band keinen NODATA Wert aufweisen soll, übergeben Sie bitte für den Parameter "nodatavalue" NULL.

#### Synopsis

```
raster ST_SetBandNoDataValue(raster rast, double precision nodatavalue);
raster ST_SetBandNoDataValue(raster rast, integer band, double precision nodatavalue, boolean forcechecking=false);
```

#### Beschreibung

Setzt den NODATA Wert eines Bandes. Wenn kein Band angegeben ist, wird Band 1 angenommen. Dies beeinflusst die Ergebnisse von [ST\\_Polygon](#), [ST\\_DumpAsPolygons](#) und die Funktionen `ST_PixelAs...`.

## Beispiele

```

-- change just first band no data value
UPDATE dummy_rast
  SET rast = ST_SetBandNoDataValue(rast,1, 254)
WHERE rid = 2;

-- change no data band value of bands 1,2,3
UPDATE dummy_rast
  SET rast =
    ST_SetBandNoDataValue(
      ST_SetBandNoDataValue(
        ST_SetBandNoDataValue(
          rast,1, 254)
        ,2,99),
      3,108)
  WHERE rid = 2;

-- wipe out the nodata value this will ensure all pixels are considered for all processing ←
  functions
UPDATE dummy_rast
  SET rast = ST_SetBandNoDataValue(rast,1, NULL)
WHERE rid = 2;

```

## Siehe auch

[ST\\_BandNoDataValue](#), [ST\\_NumBands](#)

### 11.8.2 ST\_SetBandIsNoData

`ST_SetBandIsNoData` — Setzt die Flag "isnodata" für das Band auf TRUE.

#### Synopsis

```
raster ST_SetBandIsNoData(raster rast, integer band=1);
```

#### Beschreibung

Setzt die Flag "isnodata" des Bandes auf TRUE. Wenn kein Band angegeben ist, wird Band 1 angenommen. Diese Funktion sollte nur aufgerufen werden, wenn sich die Flag verändert hat. Dies ist der Fall, wenn sich die Ergebnisse von [ST\\_BandIsNoData](#) unterscheiden, da einmal mit TRUE als letzten Übergabewert und ein anderes mal ohne diesen aufgerufen wurde.

Verfügbarkeit: 2.0.0

## Beispiele

```

-- Create dummy table with one raster column
create table dummy_rast (rid integer, rast raster);

-- Add raster with two bands, one pixel/band. In the first band, nodatavalue = pixel value ←
  = 3.
-- In the second band, nodatavalue = 13, pixel value = 4
insert into dummy_rast values(1,
(
'01' -- little endian (uint8 ndr)

```

```

||
'0000' -- version (uint16 0)
||
'0200' -- nBands (uint16 0)
||
'17263529ED684A3F' -- scaleX (float64 0.000805965234044584)
||
'F9253529ED684ABF' -- scaleY (float64 -0.00080596523404458)
||
'1C9F33CE69E352C0' -- ipX (float64 -75.5533328537098)
||
'718F0E9A27A44840' -- ipY (float64 49.2824585505576)
||
'ED50EB853EC32B3F' -- skewX (float64 0.000211812383858707)
||
'7550EB853EC32B3F' -- skewY (float64 0.000211812383858704)
||
'E6100000' -- SRID (int32 4326)
||
'0100' -- width (uint16 1)
||
'0100' -- height (uint16 1)
||
'4' -- hasnodatavalue set to true, isnodata value set to false (when it should be true)
||
'2' -- first band type (4BUI)
||
'03' -- novalue==3
||
'03' -- pixel(0,0)==3 (same that nodata)
||
'0' -- hasnodatavalue set to false
||
'5' -- second band type (16BSI)
||
'0D00' -- novalue==13
||
'0400' -- pixel(0,0)==4
)::raster
);

select st_bandisnodata(rast, 1) from dummy_rast where rid = 1; -- Expected false
select st_bandisnodata(rast, 1, TRUE) from dummy_rast where rid = 1; -- Expected true

-- The isnodata flag is dirty. We are going to set it to true
update dummy_rast set rast = st_setbandisnodata(rast, 1) where rid = 1;

select st_bandisnodata(rast, 1) from dummy_rast where rid = 1; -- Expected true

```

**Siehe auch**

[ST\\_BandNoDataValue](#), [ST\\_NumBands](#), [ST\\_SetBandNoDataValue](#), [ST\\_BandIsNoData](#)

**11.8.3 ST\_SetBandPath**

`ST_SetBandPath` — Aktualisiert den externen Dateipfad und die Bandnummer eines out-db Bandes.

## Synopsis

raster **ST\_SetBandPath**(raster rast, integer band, text outdbpath, integer outdbindex, boolean force=false);

## Beschreibung

Aktualisiert den externen Rasterdateipfad und die externe Bandnummer eines out-db Bandes.



### Note

Wenn `force` auf TRUE gesetzt ist, wird die Kompatibilität (z.B. Ausrichtung, Pixelunterstützung) zwischen der externen Rasterdatei und dem PostGIS Raster nicht überprüft. Dieser Modus ist für Dateisystemänderungen vorgesehen, bei denen der externe Raster bestehen bleibt.

Verfügbarkeit: 2.5.0

## Beispiele

```
WITH foo AS (
  SELECT
    ST_AddBand(NULL::raster, '/home/pele/devel/geo/postgis-git/raster/test/regress/ ↵
      loader/Projected.tif', NULL::int[]) AS rast
)
SELECT
  1 AS query,
  *
FROM ST_BandMetadata(
  (SELECT rast FROM foo),
  ARRAY[1,3,2]::int[]
)
UNION ALL
SELECT
  2,
  *
FROM ST_BandMetadata(
  (
    SELECT
      ST_SetBandPath(
        rast,
        2,
        '/home/pele/devel/geo/postgis-git/raster/test/regress/loader/Projected2.tif ↵
        ',
        1
      ) AS rast
    FROM foo
  ),
  ARRAY[1,3,2]::int[]
)
ORDER BY 1, 2;
```

query	bandnum	pixeltype	nodatavalue	isoutdb	path	outdbbandnum
1	1	8BUI		t	/home/pele/devel/geo/postgis-git/ ↵ raster/test/regress/loader/Projected.tif	1
1	2	8BUI		t	/home/pele/devel/geo/postgis-git/ ↵ raster/test/regress/loader/Projected.tif	2



```

1 |      3 | 8BUI | | t | /home/pele/devel/geo/postgis-git/ ↔
  raster/test/regress/loader/Projected.tif |      3
2 |      1 | 8BUI | | t | /home/pele/devel/geo/postgis-git/ ↔
  raster/test/regress/loader/Projected.tif |      1
2 |      2 | 8BUI | | t | /home/pele/devel/geo/postgis-git/ ↔
raster/test/regress/loader/Projected2.tif |      1
2 |      3 | 8BUI | | t | /home/pele/devel/geo/postgis-git/ ↔
  raster/test/regress/loader/Projected.tif |      3

```

**Siehe auch**

[ST\\_BandMetaData](#), [ST\\_SetBandIndex](#)

**11.8.4 ST\_SetBandIndex**

`ST_SetBandIndex` — Aktualisiert die externe Bandnummer eines out-db Bandes.

**Synopsis**

```
raster ST_SetBandIndex(raster rast, integer band, integer outdbindex, boolean force=false);
```

**Beschreibung**

Aktualisiert die externe Bandnummer eines out-db Bandes. Die mit dem out-db Band assoziierte externe Rasterdatei wird davon nicht betroffen

**Note**

Wenn `force` auf TRUE gesetzt ist, wird die Kompatibilität (z.B. Ausrichtung, Pixelunterstützung) zwischen der externen Rasterdatei und dem PostGIS Raster nicht überprüft. Dieser Modus ist für das Verschieben von Bändern in einer externen Rasterdatei vorgesehen.

**Note**

Intern wird bei dieser Methode das PostGIS Raster Band mit dem Index `band` durch ein neues Band ersetzt, ohne dass die existierende Pfadinformation aktualisiert wird.

Verfügbarkeit: 2.5.0

**Beispiele**

```

WITH foo AS (
  SELECT
    ST_AddBand(NULL::raster, '/home/pele/devel/geo/postgis-git/raster/test/regress/ ↔
      loader/Projected.tif', NULL::int[]) AS rast
)
SELECT
  1 AS query,
  *
FROM ST_BandMetadata (
  (SELECT rast FROM foo),
  ARRAY[1,3,2]::int[]
)

```

```

UNION ALL
SELECT
    2,
    *
FROM ST_BandMetadata (
    (
        SELECT
            ST_SetBandIndex (
                rast,
                2,
                1
            ) AS rast
        FROM foo
    ),
    ARRAY[1,3,2]::int[]
)
ORDER BY 1, 2;

```

query	bandnum	pixeltype	nodatavalue	isoutdb	path
	1	8BUI		t	/home/pele/devel/geo/postgis-git/raster/test/regress/loader/Projected.tif
	2	8BUI		t	/home/pele/devel/geo/postgis-git/raster/test/regress/loader/Projected.tif
	3	8BUI		t	/home/pele/devel/geo/postgis-git/raster/test/regress/loader/Projected.tif
	1	8BUI		t	/home/pele/devel/geo/postgis-git/raster/test/regress/loader/Projected.tif
	<b>2</b>	<b>8BUI</b>		<b>t</b>	<b>/home/pele/devel/geo/postgis-git/raster/test/regress/loader/Projected.tif</b>
	2	8BUI		t	/home/pele/devel/geo/postgis-git/raster/test/regress/loader/Projected.tif

**Siehe auch**

[ST\\_BandMetaData](#), [ST\\_SetBandPath](#)

## 11.9 Rasterband Statistik und Analytik

### 11.9.1 ST\_Count

**ST\_Count** — Gibt die Anzahl der Pixel für ein Band eines Rasters oder eines Raster-Coverage zurück. Wenn kein Band angegeben ist, wird standardmäßig Band 1 gewählt. Wenn der Parameter "exclude\_nodata\_value" auf TRUE gesetzt ist, werden nur Pixel mit Werten ungleich NODATA gezählt.

**Synopsis**

```

bigint ST_Count(raster rast, integer nband=1, boolean exclude_nodata_value=true);
bigint ST_Count(raster rast, boolean exclude_nodata_value);

```

## 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

Geändert: 3.1.0 - Die `ST_Count(rastertable, rastercolumn, ...)` Varianten wurden entfernt. Verwenden Sie stattdessen [ST\\_CountAgg](#).

Verfügbarkeit: 2.0.0

## Beispiele

```
--example will count all pixels not 249 and one will count all pixels. --
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](#)

## 11.9.2 ST\_CountAgg

`ST_CountAgg` — Aggregatfunktion. Gibt die Anzahl der Pixel in einem bestimmten Band der Raster aus. Wenn kein Band angegeben ist, wird Band 1 angenommen. Wenn "exclude\_nodata\_value" `TRUE` ist, werden nur die Pixel ohne `NODATA` Werte gezählt.

## Synopsis

```
bigint ST_CountAgg(raster rast, integer nband, boolean exclude_nodata_value, double precision sample_percent);
bigint ST_CountAgg(raster rast, integer nband, boolean exclude_nodata_value);
bigint ST_CountAgg(raster rast, boolean exclude_nodata_value);
```

## Beschreibung

Gibt die Anzahl der Pixel in einem bestimmten Band der Raster aus. Wenn kein Band angegeben ist, wird `nband` standardmäßig auf 1 gesetzt.

Wenn der Parameter `exclude_nodata_value` auf `TRUE` gesetzt ist, werden nur die Pixel des Rasters gezählt, deren Werte ungleich `NODATA` sind. Setzen Sie bitte `exclude_nodata_value` auf `FALSE` um die Anzahl aller Pixel zu erhalten

Standardmäßig werden alle Pixel abgetastet. Um eine schnellere Rückmeldung zu bekommen, können Sie den Parameter `sample_percent` auf einen Wert zwischen null (0) und eins (1) setzen.

Verfügbarkeit: 2.2.0

## Beispiele

```

WITH foo AS (
  SELECT
    rast.rast
  FROM (
    SELECT ST_SetValue(
      ST_SetValue(
        ST_SetValue(
          ST_AddBand(
            ST_MakeEmptyRaster(10, 10, 10, 10, 2, 2, 0, 0,0)
            , 1, '64BF', 0, 0
          )
          , 1, 1, 1, -10
        )
        , 1, 5, 4, 0
      )
      , 1, 5, 5, 3.14159
    ) AS rast
  ) AS rast
  FULL JOIN (
    SELECT generate_series(1, 10) AS id
  ) AS id
  ON 1 = 1
)
SELECT
  ST_CountAgg(rast, 1, TRUE)
FROM foo;

 st_countagg
-----
                20
(1 row)

```

## Siehe auch

[ST\\_Count](#), [ST\\_SummaryStats](#), [ST\\_SetBandNoDataValue](#)

### 11.9.3 ST\_Histogram

**ST\_Histogram** — Gibt Datensätze aus, welche die Verteilung der Daten eines Rasters oder eines Rastercoverage darstellen. Dabei wird die Wertemenge in Klassen aufgeteilt und für jede Klasse zusammengefasst. Wenn die Anzahl der Klassen nicht angegeben ist, wird sie automatisch berechnet.

#### Synopsis

```

SETOF record ST_Histogram(raster rast, integer nband=1, boolean exclude_nodata_value=true, integer bins=autocomputed,
double precision[] width=NULL, boolean right=false);
SETOF record ST_Histogram(raster rast, integer nband, integer bins, double precision[] width=NULL, boolean right=false);
SETOF record ST_Histogram(raster rast, integer nband, boolean exclude_nodata_value, integer bins, boolean right);
SETOF record ST_Histogram(raster rast, integer nband, integer bins, boolean right);

```

#### 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** 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** Anzahl der Klassen - die Anzahl der Datensätze die von der Funktion ausgegeben werden. Wenn die Anzahl der Klassen nicht angegeben ist, wird sie automatisch berechnet.

**right** Berechnet das Histogramm von rechts anstatt von links (Standardwert). Dies ändert das Auswahlkriterium für einen Wert x von [a,b) auf (a,b].

Geändert: 3.1.0 Die Variante `ST_Histogram(table_name, column_name)` wurde entfernt.

Verfügbarkeit: 2.0.0

### Beispiel: Einzelne Rasterkachel - Berechnung des Histogramm für die Bänder 1, 2, 3 und automatische Einteilung der Wertemenge in Klassen

```
SELECT band, (stats).*
FROM (SELECT rid, band, ST_Histogram(rast, band) As stats
      FROM dummy_rast CROSS JOIN generate_series(1,3) As band
      WHERE rid=2) As foo;
```

band	min	max	count	percent
1	249	250	2	0.08
1	250	251	2	0.08
1	251	252	1	0.04
1	252	253	2	0.08
1	253	254	18	0.72
2	78	113.2	11	0.44
2	113.2	148.4	4	0.16
2	148.4	183.6	4	0.16
2	183.6	218.8	1	0.04
2	218.8	254	5	0.2
3	62	100.4	11	0.44
3	100.4	138.8	5	0.2
3	138.8	177.2	4	0.16
3	177.2	215.6	1	0.04
3	215.6	254	4	0.16

### Beispiel: Nur Band 2 und 6 Klassen

```
SELECT (stats).*
FROM (SELECT rid, ST_Histogram(rast, 2,6) As stats
      FROM dummy_rast
      WHERE rid=2) As foo;
```

min	max	count	percent
78	107.333333	9	0.36
107.333333	136.666667	6	0.24

```

136.666667 |      166 |      0 |      0
      166 | 195.333333 |      4 |    0.16
195.333333 | 224.666667 |      1 |    0.04
224.666667 |      254 |      5 |    0.2
(6 rows)

-- Same as previous but we explicitly control the pixel value range of each bin.
SELECT (stats).*
FROM (SELECT rid, ST_Histogram(rast, 2,6,ARRAY[0.5,1,4,100,5]) As stats
      FROM dummy_rast
      WHERE rid=2) As foo;

  min | max | count | percent
-----+-----+-----+-----
   78 | 78.5 |      1 |    0.08
  78.5 | 79.5 |      1 |    0.04
  79.5 | 83.5 |      0 |      0
  83.5 | 183.5 |     17 |   0.0068
 183.5 | 188.5 |      0 |      0
 188.5 | 254 |      6 | 0.003664
(6 rows)

```

**Siehe auch**

[ST\\_Count](#), [ST\\_SummaryStats](#), [ST\\_SummaryStatsAgg](#)

**11.9.4 ST\_Quantile**

**ST\_Quantile** — Berechnet die Quantile eines Rasters oder einer Rastercoverage Tabelle im Kontext von Stichproben oder Bevölkerung. Dadurch kann untersucht werden, ob ein Wert bei 25%, 50% oder 75% Perzentil des Rasters liegt.

**Synopsis**

```

SETOF record ST_Quantile(raster rast, integer nband=1, boolean exclude_nodata_value=true, double precision[] quantiles=NULL);
SETOF record ST_Quantile(raster rast, double precision[] quantiles);
SETOF record ST_Quantile(raster rast, integer nband, double precision[] quantiles);
double precision ST_Quantile(raster rast, double precision quantile);
double precision ST_Quantile(raster rast, boolean exclude_nodata_value, double precision quantile=NULL);
double precision ST_Quantile(raster rast, integer nband, double precision quantile);
double precision ST_Quantile(raster rast, integer nband, boolean exclude_nodata_value, double precision quantile);
double precision ST_Quantile(raster rast, integer nband, double precision quantile);

```

**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.

Geändert: 3.1.0 Die Variante `ST_Quantile(table_name, column_name)` wurde entfernt.

Verfügbarkeit: 2.0.0

**Beispiele**

```
UPDATE dummy_rast SET rast = ST_SetBandNoDataValue(rast,249) WHERE rid=2;
--Example will consider only pixels of band 1 that are not 249 and in named quantiles --
```

```
SELECT (pvq).*
FROM (SELECT ST_Quantile(rast, ARRAY[0.25,0.75]) As pvq
      FROM dummy_rast WHERE rid=2) As foo
      ORDER BY (pvq).quantile;
```

```
quantile | value
-----+-----
    0.25 |   253
    0.75 |   254
```

```
SELECT ST_Quantile(rast, 0.75) As value
      FROM dummy_rast WHERE rid=2;
```

```
value
-----
   254
```

```
--real live example.  Quantile of all pixels in band 2 intersecting a geometry
SELECT rid, (ST_Quantile(rast,2)).* As pvc
      FROM o_4_boston
           WHERE ST_Intersects(rast,
                               ST_GeomFromText('POLYGON((224486 892151,224486 892200,224706 892200,224706
                               892151,224486 892151))',26986)
                               )
ORDER BY value, quantile,rid
;
```

```
rid | quantile | value
-----+-----+-----
   1 |         0 |     0
   2 |         0 |     0
  14 |         0 |     1
  15 |         0 |     2
  14 |    0.25 |    37
   1 |    0.25 |    42
  15 |    0.25 |    47
   2 |    0.25 |    50
  14 |    0.5  |    56
   1 |    0.5  |    64
  15 |    0.5  |    66
   2 |    0.5  |    77
  14 |    0.75 |    81
  15 |    0.75 |    87
   1 |    0.75 |    94
   2 |    0.75 |   106
  14 |     1  |   199
   1 |     1  |   244
   2 |     1  |   255
  15 |     1  |   255
```

**Siehe auch**

[ST\\_Count](#), [ST\\_SummaryStats](#), [ST\\_SummaryStatsAgg](#), [ST\\_SetBandNoDataValue](#)

## 11.9.5 ST\_SummaryStats

`ST_SummaryStats` — Gibt eine zusammenfassende Statistik aus, bestehend aus der Anzahl, der Summe, dem arithmetischen Mittel, der Standardabweichung, dem Minimum und dem Maximum der Werte eines Rasterbandes oder eines Rastercoverage. Wenn kein Band angegeben ist, wird Band 1 angenommen.

### Synopsis

```
summarystats ST_SummaryStats(raster rast, boolean exclude_nodata_value);
summarystats ST_SummaryStats(raster rast, integer nband, boolean exclude_nodata_value);
```

### Beschreibung

Gibt **summarystats** aus, bestehend aus der Anzahl, der Summe, dem arithmetischen Mittel, der Standardabweichung, dem Minimum und dem Maximum der Werte eines Rasterbandes oder eines Rastercoverage. Wenn kein Band angegeben ist, wird Band 1 angenommen.



#### Note

Standardmäßig werden nur jene Pixelwerte berücksichtigt, die nicht den Wert `NODATA` haben. Setzen Sie bitte `exclude_nodata_value` auf `FALSE` um die Anzahl sämtlicher Pixel zu erhalten.



#### Note

Standardmäßig werden alle Pixel abgetastet. Um eine schnellere Rückmeldung zu bekommen, können Sie den Parameter `sample_percent` auf einen Wert kleiner als 1 setzen.

Geändert: 3.1.0 `ST_SummaryStats(rastertable, rastercolumn, ...)` Varianten wurden entfernt. Verwenden Sie stattdessen `ST_SummaryStats`.

Verfügbarkeit: 2.0.0

### Beispiel: Einzelne Rasterkachel

```
SELECT rid, band, (stats).*
FROM (SELECT rid, band, ST_SummaryStats(rast, band) As stats
      FROM dummy_rast CROSS JOIN generate_series(1,3) As band
      WHERE rid=2) As foo;
```

rid	band	count	sum	mean	stddev	min	max
2	1	23	5821	253.086957	1.248061	250	254
2	2	25	3682	147.28	59.862188	78	254
2	3	25	3290	131.6	61.647384	62	254

### Beispiel: Zusammenfassen der Pixel, die interessante Bauwerke schneiden

Dieses Beispiel benötigte 574ms in PostGIS unter Windows 64-Bit, mit allen Bauwerken und Luftbildkacheln von Boston (Kacheln jeweils 150x150 Pixel ~ 134.000 Kacheln; ~ 102.000 Datensätze mit Bauwerken)

```
WITH
-- our features of interest
feat AS (SELECT gid As building_id, geom_26986 As geom FROM buildings AS b
         WHERE gid IN(100, 103,150)
```



```

),
-- clip band 2 of raster tiles to boundaries of builds
-- then get stats for these clipped regions
b_stats AS
  (SELECT building_id, (stats).*
FROM (SELECT building_id, ST_SummaryStats(ST_Clip(rast,2,geom)) As stats
FROM aerals.boston
INNER JOIN feat
ON ST_Intersects(feats.geom,rast)
) As foo
)
-- finally summarize stats
SELECT building_id, SUM(count) As num_pixels
, MIN(min) As min_pval
, MAX(max) As max_pval
, SUM(mean*count)/SUM(count) As avg_pval
FROM b_stats
WHERE count
> 0
GROUP BY building_id
ORDER BY building_id;
building_id | num_pixels | min_pval | max_pval | avg_pval
-----+-----+-----+-----+-----
100 | 1090 | 1 | 255 | 61.0697247706422
103 | 655 | 7 | 182 | 70.5038167938931
150 | 895 | 2 | 252 | 185.642458100559

```

### Beispiel: Rastercoverage

```

-- stats for each band --
SELECT band, (stats).*
FROM (SELECT band, ST_SummaryStats('o_4_boston','rast', band) As stats
FROM generate_series(1,3) As band) As foo;

band | count | sum | mean | stddev | min | max
-----+-----+-----+-----+-----+-----+-----
1 | 8450000 | 725799 | 82.7064349112426 | 45.6800222638537 | 0 | 255
2 | 8450000 | 700487 | 81.4197705325444 | 44.2161184161765 | 0 | 255
3 | 8450000 | 575943 | 74.682739408284 | 44.2143885481407 | 0 | 255

-- For a table -- will get better speed if set sampling to less than 100%
-- Here we set to 25% and get a much faster answer
SELECT band, (stats).*
FROM (SELECT band, ST_SummaryStats('o_4_boston','rast', band,true,0.25) As stats
FROM generate_series(1,3) As band) As foo;

band | count | sum | mean | stddev | min | max
-----+-----+-----+-----+-----+-----+-----
1 | 2112500 | 180686 | 82.6890480473373 | 45.6961043857248 | 0 | 255
2 | 2112500 | 174571 | 81.448503668639 | 44.2252623171821 | 0 | 255
3 | 2112500 | 144364 | 74.6765884023669 | 44.2014869384578 | 0 | 255

```

### Siehe auch

[summarystats](#), [ST\\_SummaryStatsAgg](#), [ST\\_Count](#), [ST\\_Clip](#)

## 11.9.6 ST\_SummaryStatsAgg

`ST_SummaryStatsAgg` — Aggregatfunktion. Gibt eine zusammenfassende Statistik aus, die aus der Anzahl, der Summe, dem arithmetischen Mittel, dem Minimum und dem Maximum der Werte eines bestimmten Bandes eines Rastersatzes besteht. Wenn kein Band angegeben ist, wird Band 1 angenommen.

### Synopsis

```
summarystats ST_SummaryStatsAgg(raster rast, integer nband, boolean exclude_nodata_value, double precision sample_percent);
summarystats ST_SummaryStatsAgg(raster rast, boolean exclude_nodata_value, double precision sample_percent);
summarystats ST_SummaryStatsAgg(raster rast, integer nband, boolean exclude_nodata_value);
```

### Beschreibung

Gibt `summarystats` aus, bestehend aus der Anzahl, der Summe, dem arithmetischen Mittel, der Standardabweichung, dem Minimum und dem Maximum der Werte eines Rasterbandes oder eines Rastercoverage. Wenn kein Band angegeben ist, wird Band 1 angenommen.



#### Note

Standardmäßig werden nur jene Pixelwerte berücksichtigt, die nicht `NODATA` sind. Setzen Sie bitte `exclude_nodata_value` auf `FALSE` um die Anzahl sämtlicher Pixel zu erhalten.



#### Note

Standardmäßig werden alle Pixel abgetastet. Um eine schnellere Rückmeldung zu bekommen, können Sie den Parameter `sample_percent` auf einen Wert zwischen 0 und 1 setzen.

Verfügbarkeit: 2.2.0

### Beispiele

```
WITH foo AS (
  SELECT
    rast.rast
  FROM (
    SELECT ST_SetValue(
      ST_SetValue(
        ST_SetValue(
          ST_AddBand(
            ST_MakeEmptyRaster(10, 10, 10, 10, 2, 2, 0, 0,0)
            , 1, '64BF', 0, 0
          )
          , 1, 1, 1, -10
        )
        , 1, 5, 4, 0
      )
      , 1, 5, 5, 3.14159
    ) AS rast
  ) AS rast
  FULL JOIN (
    SELECT generate_series(1, 10) AS id
  ) AS id
  ON 1 = 1
```

```

)
SELECT
    (stats).count,
    round((stats).sum::numeric, 3),
    round((stats).mean::numeric, 3),
    round((stats).stddev::numeric, 3),
    round((stats).min::numeric, 3),
    round((stats).max::numeric, 3)
FROM (
    SELECT
        ST_SummaryStatsAgg(rast, 1, TRUE, 1) AS stats
    FROM foo
) bar;

```

```

count | round | round | round | round | round
-----+-----+-----+-----+-----+-----
    20 | -68.584 | -3.429 | 6.571 | -10.000 | 3.142
(1 row)

```

## Siehe auch

[summarystats](#), [ST\\_SummaryStats](#), [ST\\_Count](#), [ST\\_Clip](#)

### 11.9.7 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;
--Example will count only pixels of band 1 that are not 249. --
```

```
SELECT (pvc).*
FROM (SELECT ST_ValueCount(rast) As pvc
      FROM dummy_rast WHERE rid=2) As foo
      ORDER BY (pvc).value;
```

value	count
250	2
251	1
252	2
253	6
254	12

```
-- Example will count all pixels of band 1 including 249 --
```

```
SELECT (pvc).*
FROM (SELECT ST_ValueCount(rast,1,false) As pvc
      FROM dummy_rast WHERE rid=2) As foo
      ORDER BY (pvc).value;
```

value	count
249	2
250	2
251	1
252	2
253	6
254	12

```
-- Example will count only non-nodata value pixels of band 2
```

```
SELECT (pvc).*
FROM (SELECT ST_ValueCount(rast,2) As pvc
      FROM dummy_rast WHERE rid=2) As foo
      ORDER BY (pvc).value;
```

value	count
78	1
79	1
88	1

```

89 | 1
96 | 1
97 | 1
98 | 1
99 | 2
112 | 2

```

```
:
```

```

--real live example. Count all the pixels in an aerial raster tile band 2 intersecting a ←
  geometry
-- and return only the pixel band values that have a count > 500
SELECT (pvc).value, SUM((pvc).count) As total
FROM (SELECT ST_ValueCount(rast,2) As pvc
      FROM o_4_boston
      WHERE ST_Intersects(rast,
        ST_GeomFromText('POLYGON((224486 892151,224486 892200,224706 892200,224706 ←
          892151,224486 892151))',26986)
        )
      ) As foo
GROUP BY (pvc).value
HAVING SUM((pvc).count) > 500
ORDER BY (pvc).value;

```

```

value | total
-----+-----
51 | 502
54 | 521

```

```

-- Just return count of pixels in each raster tile that have value of 100 of tiles that ←
  intersect a specific geometry --
SELECT rid, ST_ValueCount(rast,2,100) As count
FROM o_4_boston
WHERE ST_Intersects(rast,
  ST_GeomFromText('POLYGON((224486 892151,224486 892200,224706 892200,224706 ←
    892151,224486 892151))',26986)
  ) ;

```

```

rid | count
-----+-----
1 | 56
2 | 95
14 | 37
15 | 64

```

## Siehe auch

[ST\\_Count](#), [ST\\_SetBandNoDataValue](#)

## 11.10 Rastereingabe

### 11.10.1 ST\_RastFromWKB

`ST_RastFromWKB` — Gibt einen Rasterwert von einer Well-known-Binary (WKB) Darstellung eines Rasters zurück.





**Siehe auch**

[ST\\_RastFromWKB](#), [ST\\_AsHexWKB](#)

**11.11.2 ST\_AsHexWKB**

`ST_AsHexWKB` — Gibt die Well-known-Binary (WKB) Hex-Darstellung eines Rasters zurück.

**Synopsis**

```
bytea ST_AsHexWKB(raster rast, boolean outasin=FALSE);
```

**Beschreibung**

Gibt den Raster in binärer Hex-Darstellung zurück. Wenn `outasin` `TRUE` ist, werden Bänder die außerhalb der Datenbank liegen (`out-db`) gleich behandelt wie Bänder, die direkt in der Datenbank gespeichert (`in-db`) sind. Siehe "raster/doc/RFC2-WellKnownBinaryFormat" im Quellverzeichnis von PostGIS für Details zu dieser Darstellung.

**Note**

Standardmäßig ist in der Hex-WKB-Ausgabe der Pfad der `out-db` Bänder enthalten. Wenn ein Client keinen Zugriff auf die Rasterdatei eines "out-db" Bandes hat, können Sie `outasin` auf `TRUE` setzen.

Verfügbarkeit: 2.5.0

**Beispiele**

```
SELECT ST_AsHexWKB(rast) As rastbin FROM dummy_rast WHERE rid=1;
          st_ashexwkb
-----
0100000000000000000000000000000040000000000000008400000000000000 ←
E03F000000000000E03F00000000000000000000000000000000A0000000A001400
```

**Siehe auch**

[ST\\_RastFromHexWKB](#), [ST\\_AsBinary](#)/[ST\\_AsWKB](#)

**11.11.3 ST\_AsGDALRaster**

`ST_AsGDALRaster` — Gibt die Rasterkachel in dem ausgewiesenen Rasterformat von GDAL aus. Sie können jedes Rasterformat angeben, das von Ihrer Bibliothek unterstützt wird. Um eine Liste mit den unterstützten Formaten auszugeben, verwenden Sie bitte `ST_GDALDrivers()`.

**Synopsis**

```
bytea ST_AsGDALRaster(raster rast, text format, text[] options=NULL, integer srid=sameassource);
```



## Beschreibung

Gibt die Rasterkachel im dem ausgewiesenen Format zurück. Die Übergabewerte sind:

- `format` das Format das abgerufen wird. Dies ist abhängig von den Treibern die mit Ihrer Bibliothek "libgdal" kompiliert wurden. Üblicherweise stehen 'JPEG', 'GTiff' und 'PNG' zur Verfügung. Verwenden Sie bitte [ST\\_GDALDrivers](#), um eine Liste der von Ihrer Bibliothek unterstützten Formate zu erhalten.
- `options` ein Textfeld mit Optionen für GDAL. Gültige Optionen sind formatabhängig. Siehe [GDAL Raster format options](#) für weitere Details.
- `srs` Der Text von "proj4text" oder "srtxt" (aus der Tabelle "spatial\_ref\_sys"), der in das Rasterbild eingebettet werden soll

Verfügbarkeit: 2.0.0 - GDAL >= 1.6.0.

## Beispiel für eine JPEG Ausgabe; mehrere Kacheln als einzelner Raster

```
SELECT ST_AsGDALRaster(ST_Union(rast), 'JPEG', ARRAY['QUALITY=50']) As rastjpg
FROM dummy_rast
WHERE rast && ST_MakeEnvelope(10, 10, 11, 11);
```

## Raster mit dem "Large Object Support" von PostgreSQL exportieren

Eine Möglichkeit um Raster in einem anderen Format zu exportieren ist die Verwendung der [PostgreSQL Large Object Export Functions](#). Wir erweitern das vorherige Beispiel mit einem Export. Dafür benötigen Sie Administratorrechte für die Datenbank, da serverseitige Funktionen "Large Objects" verwendet werden. Es kann auch auf einen Server im Netzwerk exportiert werden. Wenn Sie einen lokalen Export benötigen, verwenden Sie bitte die äquivalenten "lo\_"-Funktionen von psql, welche auf das lokale Dateisystem anstatt auf das des Servers exportieren.

```
DROP TABLE IF EXISTS tmp_out ;

CREATE TABLE tmp_out AS
SELECT lo_from_bytea(0,
    ST_AsGDALRaster(ST_Union(rast), 'JPEG', ARRAY['QUALITY=50'])
) AS loid
FROM dummy_rast
WHERE rast && ST_MakeEnvelope(10, 10, 11, 11);

SELECT lo_export(loid, '/tmp/dummy.jpg')
FROM tmp_out;

SELECT lo_unlink(loid)
FROM tmp_out;
```

## Beispiel für die Ausgabe von GTIFF

```
SELECT ST_AsGDALRaster(rast, 'GTiff') As rastjpg
FROM dummy_rast WHERE rid=2;

-- Out GeoTiff with jpeg compression, 90% quality
SELECT ST_AsGDALRaster(rast, 'GTiff',
    ARRAY['COMPRESS=JPEG', 'JPEG_QUALITY=90'],
    4269) As rasttiff
FROM dummy_rast WHERE rid=2;
```

**Siehe auch**

Section [10.3](#), [ST\\_GDALDrivers](#), [ST\\_SRID](#)

**11.11.4 ST\_AsJPEG**

**ST\_AsJPEG** — Gibt die ausgewählten Bänder der Rasterkachel als einzelnes Bild (Byte-Array) im Format "Joint Photographic Exports Group" (JPEG) aus. Wenn kein Band angegeben ist und 1 oder mehr als 3 Bänder ausgewählt wurden, dann wird nur das erste Band verwendet. Wenn 3 Bänder ausgewählt wurden, werden alle 3 Bänder verwendet und auf RGB abgebildet.

**Synopsis**

```
bytea ST_AsJPEG(raster rast, text[] options=NULL);
bytea ST_AsJPEG(raster rast, integer nband, integer quality);
bytea ST_AsJPEG(raster rast, integer nband, text[] options=NULL);
bytea ST_AsJPEG(raster rast, integer[] nbands, text[] options=NULL);
bytea ST_AsJPEG(raster rast, integer[] nbands, integer quality);
```

**Beschreibung**

Gibt die ausgewählten Bänder der Rasterkachel als einzelnes Bild im Format "Joint Photographic Exports Group" (JPEG) aus. Sie können [ST\\_AsGDALRaster](#) verwenden, wenn Sie in weniger gebräuchliche Rasterformate exportieren wollen. Wenn kein Band angegeben ist und 1 oder mehr als 3 Bänder ausgewählt wurden, dann wird nur das erste Band verwendet. Wenn 3 Bänder ausgewählt wurden, werden alle 3 Bänder verwendet. Die Funktion hat viele Varianten mit vielen Optionen. Diese sind unterhalb angeführt:

- `nband` für den Export einzelner Bänder.
- `nbands` Ein Feld mit den Bändern die exportiert werden sollen (bei JPEG maximal 3). Die Reihenfolge der Bänder ist RBG; z.B. `ARRAY[3,2,1]` bedeutet, dass Band 3 auf Rot, Band 2 auf Grün und Band 1 auf Blau abgebildet wird.
- `quality` Eine Zahl zwischen 0 und 100. Umso höher die Zahl ist, umso schärfer ist das Bild.
- `options` Ein Textfeld mit GDAL Optionen für JPEG (siehe "create\_options" für JPEG unter [ST\\_GDALDrivers](#)). Gültige Optionen für JPEG sind `PROGRESSIVE ON` oder `OFF` und `QUALITY` zwischen 0 und 100 mit dem Standardwert 75. Siehe [GDAL Raster format options](#) für weitere Details.

Verfügbarkeit: 2.0.0 - GDAL >= 1.6.0.

**Beispiele: Ausgabe**

```
-- output first 3 bands 75% quality
SELECT ST_AsJPEG(rast) As rastjpg
FROM dummy_rast WHERE rid=2;

-- output only first band as 90% quality
SELECT ST_AsJPEG(rast,1,90) As rastjpg
FROM dummy_rast WHERE rid=2;

-- output first 3 bands (but make band 2 Red, band 1 green, and band 3 blue, progressive ←
and 90% quality
SELECT ST_AsJPEG(rast,ARRAY[2,1,3],ARRAY['QUALITY=90','PROGRESSIVE=ON']) As rastjpg
FROM dummy_rast WHERE rid=2;
```

## Siehe auch

Section [10.3](#), [ST\\_GDALDrivers](#), [ST\\_AsGDALRaster](#), [ST\\_AsPNG](#), [ST\\_AsTIFF](#)

### 11.11.5 ST\_AsPNG

**ST\_AsPNG** — Gibt die ausgewählten Bänder der Rasterkachel als einzelnes, übertragbares Netzwerkgraphik (PNG) Bild (Byte-Feld) aus. Wenn der Raster 1,3 oder 4 Bänder hat und keine Bänder angegeben sind, dann werden alle Bänder verwendet. Wenn der Raster 2 oder mehr als 4 Bänder hat und keine Bänder angegeben sind, dann wird nur Band 1 verwendet. Die Bänder werden in den RGB- oder den RGBA-Raum abgebildet.

#### Synopsis

```
bytea ST_AsPNG(raster rast, text[] options=NULL);
bytea ST_AsPNG(raster rast, integer nband, integer compression);
bytea ST_AsPNG(raster rast, integer nband, text[] options=NULL);
bytea ST_AsPNG(raster rast, integer[] nbands, integer compression);
bytea ST_AsPNG(raster rast, integer[] nbands, text[] options=NULL);
```

#### Beschreibung

Gibt die ausgewählten Bänder des Raster als einzelnes Bild im Format "Portable Network Graphics Image" (PNG) aus. Sie können [ST\\_AsGDALRaster](#) verwenden, wenn Sie in weniger gebräuchliche Rasterformate exportieren wollen. Wenn kein Band angegeben ist, werden die ersten 3 Bänder exportiert. Wenn SRID nicht angegeben ist, wird die SRID des Ausgangsraster verwendet. Die Funktion hat viele Varianten mit vielen Optionen. Diese sind unterhalb angeführt:

- `nband` für den Export einzelner Bänder.
- `nbands` Ein Feld mit den Bändern die exportiert werden sollen (bei PNG maximal 4). Die Reihenfolge der Bänder ist RGBA; z.B. `ARRAY[3,2,1]` bedeutet, dass Band 3 auf Rot, Band 2 auf Grün und Band 1 auf Blau abgebildet wird.
- `compression` Eine Zahl zwischen 1 und 9. Umso höher die Zahl umso größer die Komprimierung.
- `options` Ein Textfeld mit GDAL Optionen für PNG (siehe "create\_options" für PNG unter [ST\\_GDALDrivers](#)). Für PNG gibt es nur eine gültige Option "ZLEVEL" (wieviel Zeit für die Komprimierung aufgewendet werden soll - die Standardeinstellung ist 6); z.B.: `ARRAY['ZLEVEL=9']`. Ein WORLDFILE ist nicht erlaubt, da die Funktion sonst zwei Ausgaben machen müsste. Siehe [GDAL Raster format options](#) für weitere Details.

Verfügbarkeit: 2.0.0 - GDAL >= 1.6.0.

#### Beispiele

```
SELECT ST_AsPNG(rast) As rastpng
FROM dummy_rast WHERE rid=2;

-- export the first 3 bands and map band 3 to Red, band 1 to Green, band 2 to blue
SELECT ST_AsPNG(rast, ARRAY[3,1,2]) As rastpng
FROM dummy_rast WHERE rid=2;
```

## Siehe auch

[ST\\_AsGDALRaster](#), [ST\\_ColorMap](#), [ST\\_GDALDrivers](#), Section [10.3](#)

### 11.11.6 ST\_AsTIFF

`ST_AsTIFF` — Gibt die ausgewählten Bänder des Raster als einzelnes TIFF Bild (Byte-Feld) zurück. Wenn kein Band angegeben ist oder keines der angegebenen Bänder im Raster existiert, werden alle Bänder verwendet.

#### Synopsis

```
bytea ST_AsTIFF(raster rast, text[] options='', integer srid=sameassource);
bytea ST_AsTIFF(raster rast, text compression='', integer srid=sameassource);
bytea ST_AsTIFF(raster rast, integer[] nbands, text compression='', integer srid=sameassource);
bytea ST_AsTIFF(raster rast, integer[] nbands, text[] options, integer srid=sameassource);
```

#### Beschreibung

Gibt die ausgewählten Bänder des Raster als einzelnes Bild im Format " Tagged Image File Format" (TIFF) aus. Wenn kein Band angegeben ist, wird versucht alle Bänder zu verwenden. Diese Funktion ist ein Adapter für `ST_AsGDALRaster`. Verwenden Sie bitte `ST_AsGDALRaster`, wenn Sie in weniger gebräuchliche Rasterformate exportieren wollen. Wenn kein SRS-Text für die Georeferenz vorhanden ist, wird das Koordinatenreferenzsystem des Ausgangsraster verwendet. Die Funktion hat viele Varianten mit vielen Optionen. Diese sind unterhalb angeführt:

- `nbands` Ein Feld mit den Bändern die exportiert werden sollen (bei PNG maximal 3). Die Reihenfolge der Bänder ist RBG; z.B. `ARRAY[3,2,1]` bedeutet, dass Band 3 auf Rot, Band 2 auf Grün und Band 1 auf Blau abgebildet wird.
- `compression` Ein Ausdruck für die Art der Datenkompression -- JPEG90 (oder eine andere Prozentangabe), LZW, JPEG, DEFLATE9.
- `options` Ein Textfeld mit GDAL Optionen für die Erstellung eines GTiff (siehe "create\_options" für GTiff unter `ST_GDALDrivers` oder `GDAL Raster format options` für weitere Details).
- `srid` Die SRID des Koordinatenreferenzsystems des Raster. Wird als Information zur Georeferenzierung verwendet.

Verfügbarkeit: 2.0.0 - GDAL >= 1.6.0.

#### Beispiele: 90%ige JPEG Komprimierung

```
SELECT ST_AsTIFF(rast, 'JPEG90') As rasttiff
FROM dummy_rast WHERE rid=2;
```

#### Siehe auch

[ST\\_GDALDrivers](#), [ST\\_AsGDALRaster](#), [ST\\_SRID](#)

## 11.12 Rasterverarbeitung: Kartenalgebra

### 11.12.1 ST\_Clip

`ST_Clip` — Returns the raster clipped by the input geometry. If band number is not specified, all bands are processed. If `crop` is not specified or TRUE, the output raster is cropped. If `touched` is set to TRUE, then touched pixels are included, otherwise only if the center of the pixel is in the geometry it is included.

## Synopsis

```
raster ST_Clip(raster rast, integer[] nband, geometry geom, double precision[] nodataval=NULL, boolean crop=TRUE, boolean touched=FALSE);
raster ST_Clip(raster rast, integer nband, geometry geom, double precision nodataval, boolean crop=TRUE, boolean touched=FALSE);
raster ST_Clip(raster rast, integer nband, geometry geom, boolean crop, boolean touched=FALSE);
raster ST_Clip(raster rast, geometry geom, double precision[] nodataval=NULL, boolean crop=TRUE, boolean touched=FALSE);
raster ST_Clip(raster rast, geometry geom, double precision nodataval, boolean crop=TRUE, boolean touched=FALSE);
raster ST_Clip(raster rast, geometry geom, boolean crop, boolean touched=FALSE);
```

## 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, was bedeutet, dass das Ausgaberraster auf den Schnittpunkt der Ausdehnungen `geom` und `rast` beschnitten wird. Wenn `crop` auf `false` gesetzt ist, erhält das neue Raster die gleiche Ausdehnung wie `rast`. Wenn `touched` auf `true` gesetzt ist, werden alle Pixel im `rast`, die die Geometrie schneiden, ausgewählt.



### Note

Das Standardverhalten ist `touched=false`, wodurch nur Pixel ausgewählt werden, deren Mittelpunkt von der Geometrie abgedeckt ist.

Verbessert: 3.5.0 - berührtes Argument hinzugefügt.

Verfügbarkeit: 2.0.0

Erweiterung: 2.1.0 neu geschrieben in C

Die Beispiele verwenden Luftbilddaten aus Massachusetts, die auf der MassGIS-Website [MassGIS Aerial Orthos](#) verfügbar sind.

## Beispiele: Vergleich der Auswahl von "alle berührt" und "nicht alle berührt"

```
SELECT ST_Count(rast) AS count_pixels_in_orig, ST_Count(rast_touched) AS all_touched_pixels ←
, ST_Count(rast_not_touched) AS default_clip
FROM ST_AsRaster(ST_Letters('R'), scalex =
> 1.0, scaley =
> -1.0) AS r(rast)
INNER JOIN ST_GeomFromText('LINESTRING(0 1, 5 6, 10 10)') AS g(geom)
ON ST_Intersects(r.rast,g.geom)
, ST_Clip(r.rast, g.geom, touched =
> true) AS rast_touched
, ST_Clip(r.rast, g.geom, touched =
> false) AS rast_not_touched;
```

count_pixels_in_orig	all_touched_pixels	default_clip
2605	16	10

(1 row)

**Beispiele: 1 Band Clipping (nicht berührt)**

```
-- Clip the first band of an aerial tile by a 20 meter buffer.
SELECT ST_Clip(rast, 1,
              ST_Buffer(ST_Centroid(ST_Envelope(rast)),20)
              ) from aerials.boston
WHERE rid = 4;
```

```
-- Demonstrate effect of crop on final dimensions of raster
-- Note how final extent is clipped to that of the geometry
-- if crop = true
SELECT ST_XMax(ST_Envelope(ST_Clip(rast, 1, clipper, true))) As xmax_w_trim,
       ST_XMax(clipper) As xmax_clipper,
       ST_XMax(ST_Envelope(ST_Clip(rast, 1, clipper, false))) As xmax_wo_trim,
       ST_XMax(ST_Envelope(rast)) As xmax_rast_orig
FROM (SELECT rast, ST_Buffer(ST_Centroid(ST_Envelope(rast)),6) As clipper
      FROM aerials.boston
      WHERE rid = 6) As foo;
```

xmax_w_trim	xmax_clipper	xmax_wo_trim	xmax_rast_orig
230657.436173996	230657.436173996	230666.436173996	230666.436173996



*Die gesamte Rasterkachel vor dem Ausschneiden*



*Nach dem Ausschneiden*

**Beispiele: 1 Band ohne "crop" ausschneiden und die anderen Bänder ungeändert erneut hinzufügen**

```
-- Same example as before, but we need to set crop to false to be able to use ST_AddBand
-- because ST_AddBand requires all bands be the same Width and height
SELECT ST_AddBand(ST_Clip(rast, 1,
                          ST_Buffer(ST_Centroid(ST_Envelope(rast)),20),false)
                  ), ARRAY[ST_Band(rast,2),ST_Band(rast,3)] ) from aerials.boston
WHERE rid = 6;
```



*Die gesamte Rasterkachel vor dem Ausschneiden*



*Nach dem Abschneiden - unwirklich*

### Beispiele: Alle Bänder ausschneiden

```
-- Clip all bands of an aerial tile by a 20 meter buffer.
-- Only difference is we don't specify a specific band to clip
-- so all bands are clipped
SELECT ST_Clip(rast,
               ST_Buffer(ST_Centroid(ST_Envelope(rast)), 20),
               false
            ) from aerials.boston
WHERE rid = 4;
```



*Die gesamte Rasterkachel vor dem Ausschneiden*



*Nach dem Ausschneiden*

### Siehe auch

[ST\\_AddBand](#), [ST\\_Count](#), [ST\\_MapAlgebra](#) (callback function version), [ST\\_Intersection](#)



## 11.12.2 ST\_ColorMap

`ST_ColorMap` — Erzeugt aus einem bestimmten Band des Ausgangsrasters einen neuen Raster mit bis zu vier 8BUI-Bändern (Grauwert, RGB, RGBA). Wenn kein Band angegeben ist, wird Band 1 angenommen.

### Synopsis

```
raster ST_ColorMap(raster rast, integer nband=1, text colormap=grayscale, text method=INTERPOLATE);
```

```
raster ST_ColorMap(raster rast, text colormap, text method=INTERPOLATE);
```

### Beschreibung

Wendet eine `colormap` auf das Band `nband` von `rast` an, wodurch ein neuer Raster aus bis zu vier 8BUI-Bändern erstellt wird. Die Anzahl der 8BUI-Bänder des neuen Raster wird durch die Anzahl der Farbkomponenten bestimmt, die in der `colormap` definiert sind.

Wenn `nband` nicht angegeben ist, wird Band 1 angenommen.

`colormap` kann ein Schlüsselwort, ein vordefiniertes Farbschema, oder Zeilen in denen der Zellwert und die Farbkomponenten festgelegt sind.

Gültige, vordefinierte Schlüsselwörter von `colormap`:

- `grayscale` oder `greyscale` für 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**

```
-- setup test raster table --
DROP TABLE IF EXISTS funky_shapes;
CREATE TABLE funky_shapes(rast raster);

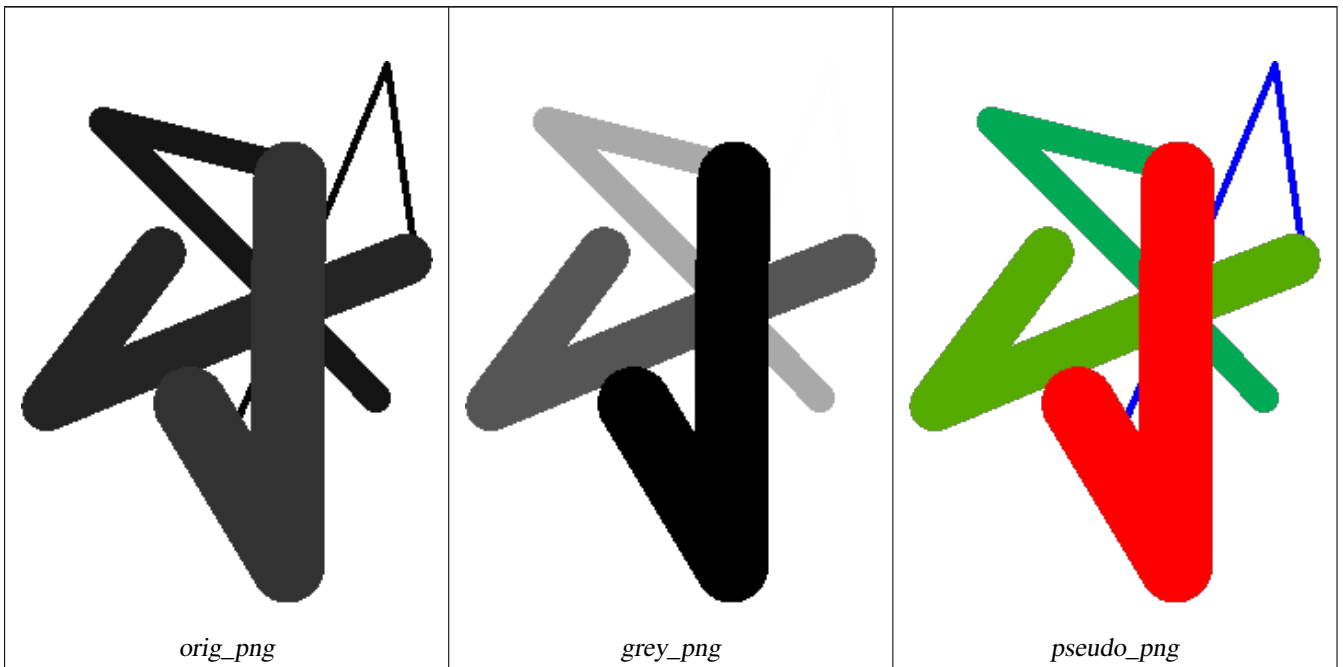
INSERT INTO funky_shapes(rast)
WITH ref AS (
  SELECT ST_MakeEmptyRaster( 200, 200, 0, 200, 1, -1, 0, 0) AS rast
)
SELECT
  ST_Union(rast)
FROM (
  SELECT
    ST_AsRaster(
      ST_Rotate(
        ST_Buffer(
          ST_GeomFromText('LINESTRING(0 2,50 50,150 150,125 50)'),
          i*2
        ),
        pi() * i * 0.125, ST_Point(50,50)
      ),
      ref.rast, '8BUI'::text, i * 5
    ) AS rast
  FROM ref
  CROSS JOIN generate_series(1, 10, 3) AS i
) AS shapes;
```

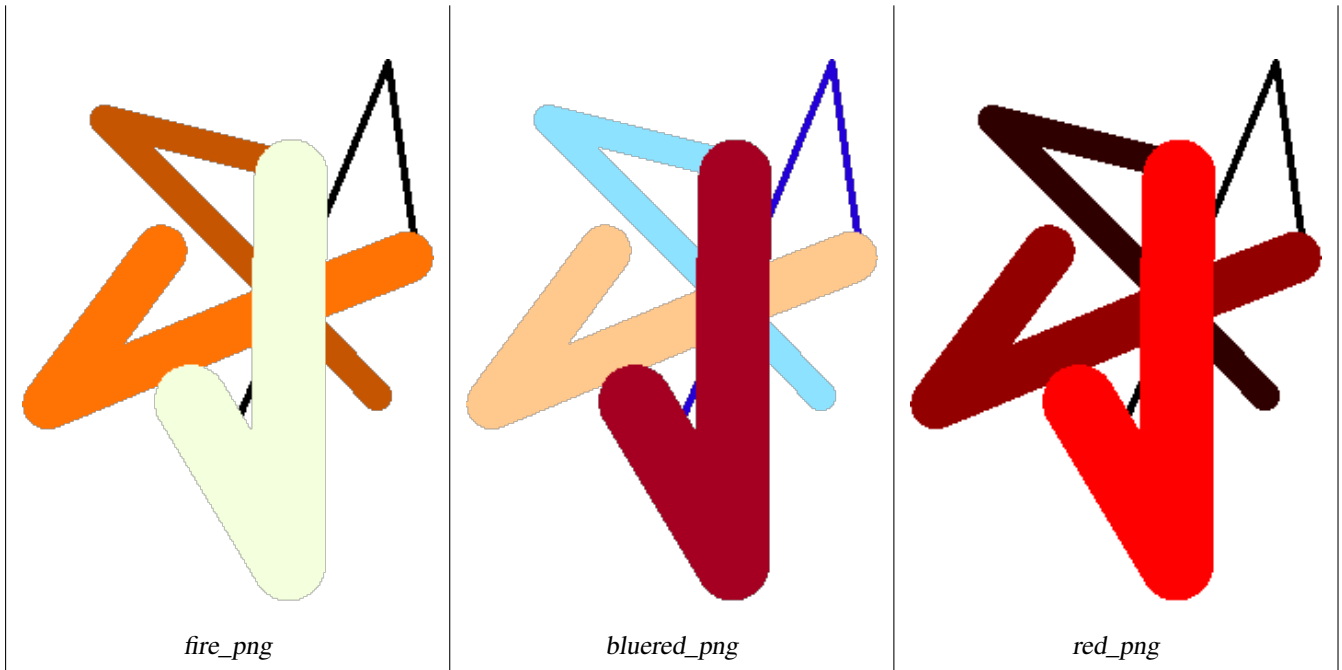
```
SELECT
  ST_NumBands(rast) As n_orig,
  ST_NumBands(ST_ColorMap(rast,1, 'greyscale')) As ngrey,
  ST_NumBands(ST_ColorMap(rast,1, 'pseudocolor')) As npseudo,
  ST_NumBands(ST_ColorMap(rast,1, 'fire')) As nfire,
  ST_NumBands(ST_ColorMap(rast,1, 'bluered')) As nbluered,
  ST_NumBands(ST_ColorMap(rast,1, '
100% 255 0 0
80% 160 0 0
50% 130 0 0
30% 30 0 0
20% 60 0 0
0% 0 0 0
nv 255 255 255
')) As nred
FROM funky_shapes;
```

```
n_orig | ngrey | npseudo | nfire | nbluered | nred
-----+-----+-----+-----+-----+-----
      1 |      1 |        4 |        4 |          4 |        3
```

**Beispiele: Vergleich verschiedener Farbtafeln mit ST\_AsPNG**

```
SELECT
  ST_AsPNG(rast) As orig_png,
  ST_AsPNG(ST_ColorMap(rast,1,'greyscale')) As grey_png,
  ST_AsPNG(ST_ColorMap(rast,1,'pseudocolor')) As pseudo_png,
  ST_AsPNG(ST_ColorMap(rast,1,'nfire')) As fire_png,
  ST_AsPNG(ST_ColorMap(rast,1,'bluered')) As bluered_png,
  ST_AsPNG(ST_ColorMap(rast,1, '
100% 255 0 0
80% 160 0 0
50% 130 0 0
30% 30 0 0
20% 60 0 0
0% 0 0 0
nv 255 255 255
')) As red_png
FROM funky_shapes;
```



**Siehe auch**

[ST\\_AsPNG](#), [ST\\_AsRaster](#) [ST\\_MapAlgebra](#) (callback function version), [ST\\_Grayscale](#) [ST\\_NumBands](#), [ST\\_Reclass](#), [ST\\_SetBandNoDataValue](#), [ST\\_Union](#)

**11.12.3 ST\_Grayscale**

**ST\_Grayscale** — Erzeugt einen neuen Raster mit einem 8BUI-Band aus dem Ausgangsraster und den angegebenen Bändern für Rot, Grün und Blau

**Synopsis**

- (1) raster **ST\_Grayscale**(raster rast, integer redband=1, integer greenband=2, integer blueband=3, text extntype=INTERSECTION);
- (2) raster **ST\_Grayscale**(rastbandarg[] rastbandargset, text extntype=INTERSECTION);

**Beschreibung**

Erzeugt einen Raster mit einem 8BUI-Band aus drei Eingabebändern (von einem oder mehreren Raster). Bänder die nicht den Pixeltyp 8BUI haben werden mit [ST\\_Reclass](#) neu klassifiziert.

**Note**

Diese Funktion unterscheidet sich insofern von [ST\\_ColorMap](#) mit dem Schlüsselwort `grayscale`, da [ST\\_ColorMap](#) lediglich ein Band bearbeitet, während diese Funktion drei Bänder für RGB erwartet. Diese Funktion verwendet folgende Gleichung zur Konvertierung von RGB auf Graustufen:  $0.2989 * RED + 0.5870 * GREEN + 0.1140 * BLUE$

Verfügbarkeit: 2.5.0

**Beispiele: Variante 1**

```
SET postgis.gdal_enabled_drivers = 'ENABLE_ALL';
SET postgis.enable_outdb_rasters = True;

WITH apple AS (
  SELECT ST_AddBand(
    ST_MakeEmptyRaster(350, 246, 0, 0, 1, -1, 0, 0, 0),
    '/tmp/apple.png'::text,
    NULL::int[]
  ) AS rast
)
SELECT
  ST_AsPNG(rast) AS original_png,
  ST_AsPNG(ST_Grayscale(rast)) AS grayscale_png
FROM apple;
```

*original\_png**grayscale\_png***Beispiele: Variante 2**

```
SET postgis.gdal_enabled_drivers = 'ENABLE_ALL';
SET postgis.enable_outdb_rasters = True;

WITH apple AS (
  SELECT ST_AddBand(
    ST_MakeEmptyRaster(350, 246, 0, 0, 1, -1, 0, 0, 0),
    '/tmp/apple.png'::text,
    NULL::int[]
  ) AS rast
)
SELECT
  ST_AsPNG(rast) AS original_png,
  ST_AsPNG(ST_Grayscale(
    ARRAY[
      ROW(rast, 1)::rastbandarg, -- red
      ROW(rast, 2)::rastbandarg, -- green
      ROW(rast, 3)::rastbandarg, -- blue
    ]::rastbandarg[]
  )) AS grayscale_png
FROM apple;
```

**Siehe auch**

[ST\\_AsPNG](#), [ST\\_Reclass](#), [ST\\_ColorMap](#)

**11.12.4 ST\_Intersection**

**ST\_Intersection** — Gibt Geometry-PixelValue Paare, oder einen Raster aus, der durch die Schnittmenge der beiden Raster bestimmt wird, oder durch die geometrische Verschneidung einer Vektorisierung des Rasters mit einem geometrischen Datentyp.

**Synopsis**

```
setof geomval ST_Intersection(geometry geom, raster rast, integer band_num=1);
setof geomval ST_Intersection(raster rast, geometry geom);
setof geomval ST_Intersection(raster rast, integer band, geometry 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\\_DumpAsPolygons](#)) und anschließend mit der Geometrie über die PostGIS Funktion [ST\\_Intersection](#)(geometry, geometry) verschritten. Wenn die verschrittene Geometrie nur aus NODATA Werten besteht, wird eine leere Geometrie zurückgegeben. Diese wird üblicherweise durch die richtige Verwendung von [ST\\_Intersects](#) in der WHERE-Klausel ausgeschlossen.

Sie können auf die Geometriebestandteile und die Werte der erzeugten "geomvals" zugreifen, indem Sie diese mit Klammern versehen und '.geom' oder '.val' am Ende des Ausdrucks hinzufügen; z.B. (ST\_Intersection(rast, geom)).geom

Die anderen Varianten, welche einen Raster zurückgeben, führen die Berechnungen im Rasterraum aus. Sie verwenden die Version mit den zwei Rastern von [ST\\_MapAlgebraExpr](#) um die Verschneidung durchzuführen.

Die Ausdehnung des resultierenden Raster entspricht der Ausdehnung des geometrischen Durchschnitts der beiden Raster. Der resultierende Raster enthält die Bänder 'BAND1', 'BAND2' und 'BOTH', gefolgt von dem Parameter `returnband`. Wenn irgendein Band Bereiche mit NODATA-Werten enthält, so werden diese Bereiche in allen Bändern des resultierenden Raster zu NODATA. Anders ausgedrückt, jedes Pixel, das ein Pixel mit NODATA-Wert schneidet wird im Ergebnis selbst zu einem Pixel mit NODATA-Wert.

Raster die aus einer Operation mit [ST\\_Intersection](#) resultieren, müssen in den Bereichen wo sie sich nicht schneiden, einen NODATA-Wert aufweisen. Sie können den NODATA Wert eines jeden resultierenden Bandes festlegen oder ersetzen, indem Sie ein Feld `nodataval[]` übergeben, das einen oder zwei NODATA Werte - 'BAND1', 'BAND2' oder 'BOTH' - enthält. Der erste Wert in dem Feld ersetzt den NODATA Wert im ersten Band, der zweite Wert ersetzt den NODATA Wert im zweiten Band. Wenn für ein übergebenes Band kein NODATA Wert festgelegt wurde und auch keiner als Feld übergeben wurde, dann wird der Wert mit der Funktion [ST\\_MinPossibleValue](#) ausgewählt. Alle Varianten, die ein Feld mit NODATA-Werten akzeptieren, nehmen auch einen einzelnen Wert entgegen, welcher dann auf alle verlangten Bänder übertragen wird.

Bei sämtlichen Varianten wird Band 1 angenommen, wenn keine Bandnummer angegeben ist. Wenn Sie die Verschneidung zwischen einem Raster und einer Geometrie als Raster ausgegeben haben wollen, sehen Sie bitte [ST\\_Clip](#).

**Note**

Um über die resultierende Ausdehnung, oder über das was für einen NODATA-Wert zurückgeben werden soll, eine bessere Kontrolle zu haben, können Sie die Variante von [ST\\_MapAlgebraExpr](#) mit den zwei Raster verwenden.

**Note**

Um eine Verschneidung von einem Rasterband mit einer Geometrie durchzuführen, verwenden Sie bitte [ST\\_Clip](#). [ST\\_Clip](#) arbeitet mit Raster mit mehreren Bändern und gibt kein Band mit der gerasterten Geometrie zurück.

**Note**

[ST\\_Intersection](#) sollte in Verbindung mit [ST\\_Intersects](#) und einem Index auf die Rasterspalte und/oder auf die Geometriespalte angewendet werden.

Enhanced: 2.0.0 - Verschneidungsoperation im Rasterraum eingeführt. In Vorgängerversionen von 2.0.0 wurde lediglich die Verschneidung im Vektorraum unterstützt.

**Beispiele: Geometrie, Raster -- das Ergebnis sind "geomvals"**

```
SELECT
  foo.rid,
  foo.gid,
  ST_AsText((foo.geomval).geom) As geomwkt,
  (foo.geomval).val
FROM (
  SELECT
    A.rid,
    g.gid,
    ST_Intersection(A.rast, g.geom) As geomval
  FROM dummy_rast AS A
  CROSS JOIN (
    VALUES
      (1, ST_Point(3427928, 5793243.85) ),
      (2, ST_GeomFromText('LINESTRING(3427927.85 5793243.75,3427927.8 5793243.75,3427927.8 5793243.8)')),
      (3, ST_GeomFromText('LINESTRING(1 2, 3 4)'))
    ) As g(gid,geom)
  WHERE A.rid = 2
) As foo;
```

rid	gid	geomwkt	val
2	1	POINT(3427928 5793243.85)	249
2	1	POINT(3427928 5793243.85)	253
2	2	POINT(3427927.85 5793243.75)	254
2	2	POINT(3427927.8 5793243.8)	251
2	2	POINT(3427927.8 5793243.8)	253
2	2	LINESTRING(3427927.8 5793243.75,3427927.8 5793243.8)	252
2	2	MULTILINESTRING((3427927.8 5793243.8,3427927.8 5793243.75),...)	250
2	3	GEOMETRYCOLLECTION EMPTY	

**Siehe auch**

[geomval](#), [ST\\_Intersects](#), [ST\\_MapAlgebraExpr](#), [ST\\_Clip](#), [ST\\_AsText](#)

**11.12.5 ST\_MapAlgebra (callback function version)**

[ST\\_MapAlgebra \(callback function version\)](#) — Die Version mit der Rückruffunktion - Gibt für einen oder mehrere Eingaberaster einen Raster mit einem Band, den Bandindizes und einer vom Anwender vorgegebenen Rückruffunktion zurück.

## Synopsis

```
raster ST_MapAlgebra(rastbandarg[] rastbandargset, regprocedure callbackfunc, text pixeltype=NULL, text extenttype=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(raster rast, integer nband, regprocedure callbackfunc, float8[] mask, boolean weighted, text pixel-
type=NULL, text extenttype=INTERSECTION, raster customextent=NULL, text[] VARIADIC userargs=NULL);
```

## Beschreibung

Gibt für einen oder mehrere Eingaberaster einen Raster mit einem Band, den Bandindizes und einer vom Anwender vorgegebenen Rückruffunktion zurück.

**rast, rast1, rast2, rastbandargset** Raster die mit der Map Algebra Operation ausgewertet werden.

`rastbandargset` ermöglicht die Anwendung einer Map Algebra Operation auf viele Raster und/oder viele Bänder. Siehe das Beispiel zu Variante 1.

**nband, nband1, nband2** Die Nummern der Rasterbänder, die ausgewertet werden sollen. Die Bänder können über "nband" als Integer oder Integer[] angegeben werden. Bei der Variante mit 2 Raster steht "nband1" für die Bänder von "rast1" und nband2 für die Bänder von rast2.

**callbackfunc** Der Parameter `callbackfunc` muss der Name und die Signatur einer SQL- oder PL/pgSQL-Funktion sein, die in eine regprocedure umgewandelt wird. Ein Beispiel für eine PL/pgSQL-Funktion ist:

```
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;
```

Der Aufruf `callbackfunc` muss drei Argumente haben: ein 3-dimensionales Double-Precision-Array, ein 2-dimensionales Integer-Array und ein variadisches 1-dimensionales Text-Array. Das erste Argument `value` ist die Menge der Werte (in doppelter Genauigkeit) aus allen Eingaberastern. Die drei Dimensionen (wobei die Indizes auf 1 basieren) sind: Raster #, Zeile y, Spalte x. Das zweite Argument `position` ist die Menge der Pixelpositionen aus dem Ausgaberraster und den Eingaberastern. Die äußere Dimension (wo die Indizes auf 0 basieren) ist das Raster #. Die Position am Index 0 der äußeren Dimension ist die Pixelposition des Ausgaberrasters. Für jede äußere Dimension gibt es zwei Elemente in der inneren Dimension für X und Y. Das dritte Argument `userargs` dient zur Übergabe von benutzerdefinierten Argumenten.

Die Übergabe eines regprocedure Arguments an eine SQL-Funktion erfordert die Übergabe der vollständigen Funktionssignatur und die anschließende Umwandlung in einen regprocedure Typ. Um die obige Beispiel-PL/pgSQL-Funktion als Argument zu übergeben, lautet das SQL für das Argument:

```
'sample_callbackfunc(double precision[], integer[], text[])':regprocedure
```

Beachten Sie, dass das Argument den Namen der Funktion, die Typen der Funktionsargumente, Anführungszeichen um den Namen und die Argumenttypen sowie einen Cast in eine regprocedure enthält.

**mask** Ein n-dimensionales Feld (Matrix) von Zahlen, das verwendet wird um Zellen für die "Map Algebra"-Rückruffunktion zu filtern. 0 bedeutet, dass ein Nachbarzellwert wie NODATA behandelt werden soll. 1 bedeutet, dass dieser Wert als Datum behandelt werden soll. Wenn der Parameter "weighted" (gewichtet) TRUE ist, dann werden diese Werte als Multiplikatoren für die Pixelwerte in der Nachbarschaft verwendet.

**weighted** Boolesche Variable (TRUE/FALSE), die angibt ob ein Wert der Maske gewichtet (mit dem ursprünglichen Wert multipliziert) werden soll oder nicht (gilt nur für den ersten, der die Maske übernimmt).

**pixeltype** Wenn `pixeltype` angegeben ist, dann hat das Band des neuen Rasters diesen Pixeltyp. Wenn für den Pixeltyp NULL oder kein Wert übergeben wird, dann hat das neue Rasterband denselben Pixeltyp wie das angegebene Band des ersten Raster (für die Lagevergleiche: INTERSECTION, UNION, FIRST, CUSTOM), oder wie das spezifizierte Band des entsprechenden Rasters (für die Lagevergleiche: SECOND, LAST). Im Zweifelsfall sollten Sie den `pixeltype` immer angeben.

Der resultierende Pixeltyp des Zielraster muss entweder aus `ST_BandPixelType` sein, weggelassen oder auf NULL gesetzt werden.

**extenttype** Mögliche Werte sind INTERSECTION (standardmäßig), UNION, FIRST (standardmäßig für Einzelraster-Varianten), SECOND, LAST, CUSTOM.

**customextent** Wenn `extenttype` CUSTOM ist, dann muss ein Raster für `customextent` übergeben werden. Siehe Beispiel 4 von Variante 1.

**distancex** Der Abstand in Pixel von der Referenzzelle in X-Richtung. Die Breite der resultierenden Matrix ergibt sich aus  $2 * \text{distancex} + 1$ . Wenn nicht angegeben, dann wird nur die Referenzzelle berücksichtigt (keine Nachbarschaft/"neighborhood of 0").

**distancey** Die Entfernung der Pixel zur Referenzzelle in Y-Richtung. Die Höhe der resultierenden Matrix ergibt sich aus  $2 * \text{distancey} + 1$ . 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, '16BUI', ←
    1, 0) AS rast
)
SELECT
  ST_MapAlgebra(
    ARRAY[ROW(rast, 1)]::rastbandarg[],
    'sample_callbackfunc(double precision[], int[], text[])':::regprocedure
  ) AS rast
FROM foo

```

### Ein Raster, mehrere Bänder

```

WITH foo AS (
  SELECT 1 AS rid, ST_AddBand(ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(2, 2, 0, 0, 1, -1, ←
    0, 0, 0), 1, '16BUI', 1, 0), 2, '8BUI', 10, 0), 3, '32BUI', 100, 0) AS rast
)
SELECT
  ST_MapAlgebra(
    ARRAY[ROW(rast, 3), ROW(rast, 1), ROW(rast, 3), ROW(rast, 2)]::rastbandarg[],
    'sample_callbackfunc(double precision[], int[], text[])':::regprocedure
  ) AS rast
FROM foo

```

### Mehrere Raster, mehrere Bänder

```

WITH foo AS (
  SELECT 1 AS rid, ST_AddBand(ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(2, 2, 0, 0, 1, -1, ←
    0, 0, 0), 1, '16BUI', 1, 0), 2, '8BUI', 10, 0), 3, '32BUI', 100, 0) AS rast UNION ←
    ALL
  SELECT 2 AS rid, ST_AddBand(ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(2, 2, 0, 0, 1, -1, ←
    0, 0, 0), 1, '16BUI', 2, 0), 2, '8BUI', 20, 0), 3, '32BUI', 300, 0) AS rast
)
SELECT
  ST_MapAlgebra(
    ARRAY[ROW(t1.rast, 3), ROW(t2.rast, 1), ROW(t2.rast, 3), ROW(t1.rast, 2)]:: ←
    rastbandarg[],
    'sample_callbackfunc(double precision[], int[], text[])':::regprocedure
  ) AS rast
FROM foo t1
CROSS JOIN foo t2
WHERE t1.rid = 1
  AND t2.rid = 2

```

Ein vollständiges Beispiel mit Coveragekacheln und Nachbarschaft. Diese Abfrage funktioniert nur mit PostgreSQL 9.1 oder höher.

```

WITH foo AS (
  SELECT 0 AS rid, ST_AddBand(ST_MakeEmptyRaster(2, 2, 0, 0, 1, -1, 0, 0, 0), 1, '16BUI', ←
    1, 0) AS rast UNION ALL
  SELECT 1, ST_AddBand(ST_MakeEmptyRaster(2, 2, 2, 0, 1, -1, 0, 0, 0), 1, '16BUI', 2, 0) ←
    AS rast UNION ALL
  SELECT 2, ST_AddBand(ST_MakeEmptyRaster(2, 2, 4, 0, 1, -1, 0, 0, 0), 1, '16BUI', 3, 0) ←
    AS rast UNION ALL

  SELECT 3, ST_AddBand(ST_MakeEmptyRaster(2, 2, 0, -2, 1, -1, 0, 0, 0), 1, '16BUI', 10, ←
    0) AS rast UNION ALL
  SELECT 4, ST_AddBand(ST_MakeEmptyRaster(2, 2, 2, -2, 1, -1, 0, 0, 0), 1, '16BUI', 20, ←
    0) AS rast UNION ALL
  SELECT 5, ST_AddBand(ST_MakeEmptyRaster(2, 2, 4, -2, 1, -1, 0, 0, 0), 1, '16BUI', 30, ←
    0) AS rast UNION ALL

```

```

SELECT 6, ST_AddBand(ST_MakeEmptyRaster(2, 2, 0, -4, 1, -1, 0, 0, 0), 1, '16BUI', 100, ←
0) AS rast UNION ALL
SELECT 7, ST_AddBand(ST_MakeEmptyRaster(2, 2, 2, -4, 1, -1, 0, 0, 0), 1, '16BUI', 200, ←
0) AS rast UNION ALL
SELECT 8, ST_AddBand(ST_MakeEmptyRaster(2, 2, 4, -4, 1, -1, 0, 0, 0), 1, '16BUI', 300, ←
0) AS rast
)
SELECT
  t1.rid,
  ST_MapAlgebra(
    ARRAY[ROW(ST_Union(t2.rast), 1)::rastbandarg[],
    'sample_callbackfunc(double precision[], int[], text[])'::regprocedure,
    '32BUI',
    'CUSTOM', t1.rast,
    1, 1
  ) AS rast
FROM foo t1
CROSS JOIN foo t2
WHERE t1.rid = 4
      AND t2.rid BETWEEN 0 AND 8
      AND ST_Intersects(t1.rast, t2.rast)
GROUP BY t1.rid, t1.rast

```

Das selbe Beispiel wie vorher, mit Coveragekacheln und Nachbarschaft, das aber auch mit PostgreSQL 9.0 funktioniert.

```

WITH src AS (
  SELECT 0 AS rid, ST_AddBand(ST_MakeEmptyRaster(2, 2, 0, 0, 1, -1, 0, 0, 0), 1, '16BUI', ←
1, 0) AS rast UNION ALL
  SELECT 1, ST_AddBand(ST_MakeEmptyRaster(2, 2, 2, 0, 1, -1, 0, 0, 0), 1, '16BUI', 2, 0) ←
AS rast UNION ALL
  SELECT 2, ST_AddBand(ST_MakeEmptyRaster(2, 2, 4, 0, 1, -1, 0, 0, 0), 1, '16BUI', 3, 0) ←
AS rast UNION ALL

  SELECT 3, ST_AddBand(ST_MakeEmptyRaster(2, 2, 0, -2, 1, -1, 0, 0, 0), 1, '16BUI', 10, ←
0) AS rast UNION ALL
  SELECT 4, ST_AddBand(ST_MakeEmptyRaster(2, 2, 2, -2, 1, -1, 0, 0, 0), 1, '16BUI', 20, ←
0) AS rast UNION ALL
  SELECT 5, ST_AddBand(ST_MakeEmptyRaster(2, 2, 4, -2, 1, -1, 0, 0, 0), 1, '16BUI', 30, ←
0) AS rast UNION ALL

  SELECT 6, ST_AddBand(ST_MakeEmptyRaster(2, 2, 0, -4, 1, -1, 0, 0, 0), 1, '16BUI', 100, ←
0) AS rast UNION ALL
  SELECT 7, ST_AddBand(ST_MakeEmptyRaster(2, 2, 2, -4, 1, -1, 0, 0, 0), 1, '16BUI', 200, ←
0) AS rast UNION ALL
  SELECT 8, ST_AddBand(ST_MakeEmptyRaster(2, 2, 4, -4, 1, -1, 0, 0, 0), 1, '16BUI', 300, ←
0) AS rast
)
WITH foo AS (
  SELECT
    t1.rid,
    ST_Union(t2.rast) AS rast
  FROM src t1
  JOIN src t2
    ON ST_Intersects(t1.rast, t2.rast)
    AND t2.rid BETWEEN 0 AND 8
  WHERE t1.rid = 4
  GROUP BY t1.rid
), bar AS (
  SELECT
    t1.rid,
    ST_MapAlgebra(

```

```

        ARRAY[ROW(t2.rast, 1)::rastbandarg[],
        'raster_nmapalgebra_test(double precision[], int[], text[])::regprocedure,
        '32BUI',
        'CUSTOM', t1.rast,
        1, 1
    ) AS rast
FROM src t1
JOIN foo t2
    ON t1.rid = t2.rid
)
SELECT
    rid,
    (ST_Metadata(rast)),
    (ST_BandMetadata(rast, 1)),
    ST_Value(rast, 1, 1, 1)
FROM bar;

```

### Beispiele: Varianten 2 und 3

#### Ein Raster, mehrere Bänder

```

WITH foo AS (
    SELECT 1 AS rid, ST_AddBand(ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(2, 2, 0, 0, 1, -1, ←
        0, 0, 0), 1, '16BUI', 1, 0), 2, '8BUI', 10, 0), 3, '32BUI', 100, 0) AS rast
)
SELECT
    ST_MapAlgebra(
        rast, ARRAY[3, 1, 3, 2)::integer[],
        'sample_callbackfunc(double precision[], int[], text[])::regprocedure
    ) AS rast
FROM foo

```

#### Ein Raster, ein Band

```

WITH foo AS (
    SELECT 1 AS rid, ST_AddBand(ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(2, 2, 0, 0, 1, -1, ←
        0, 0, 0), 1, '16BUI', 1, 0), 2, '8BUI', 10, 0), 3, '32BUI', 100, 0) AS rast
)
SELECT
    ST_MapAlgebra(
        rast, 2,
        'sample_callbackfunc(double precision[], int[], text[])::regprocedure
    ) AS rast
FROM foo

```

### Beispiele: Variante 4

#### Zwei Raster, zwei Bänder

```

WITH foo AS (
    SELECT 1 AS rid, ST_AddBand(ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(2, 2, 0, 0, 1, -1, ←
        0, 0, 0), 1, '16BUI', 1, 0), 2, '8BUI', 10, 0), 3, '32BUI', 100, 0) AS rast UNION ←
    ALL
    SELECT 2 AS rid, ST_AddBand(ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(2, 2, 0, 1, 1, -1, ←
        0, 0, 0), 1, '16BUI', 2, 0), 2, '8BUI', 20, 0), 3, '32BUI', 300, 0) AS rast
)
SELECT
    ST_MapAlgebra(
        t1.rast, 2,

```

```

        t2.rast, 1,
        'sample_callbackfunc(double precision[], int[], text[])':::regprocedure
    ) AS rast
FROM foo t1
CROSS JOIN foo t2
WHERE t1.rid = 1
      AND t2.rid = 2

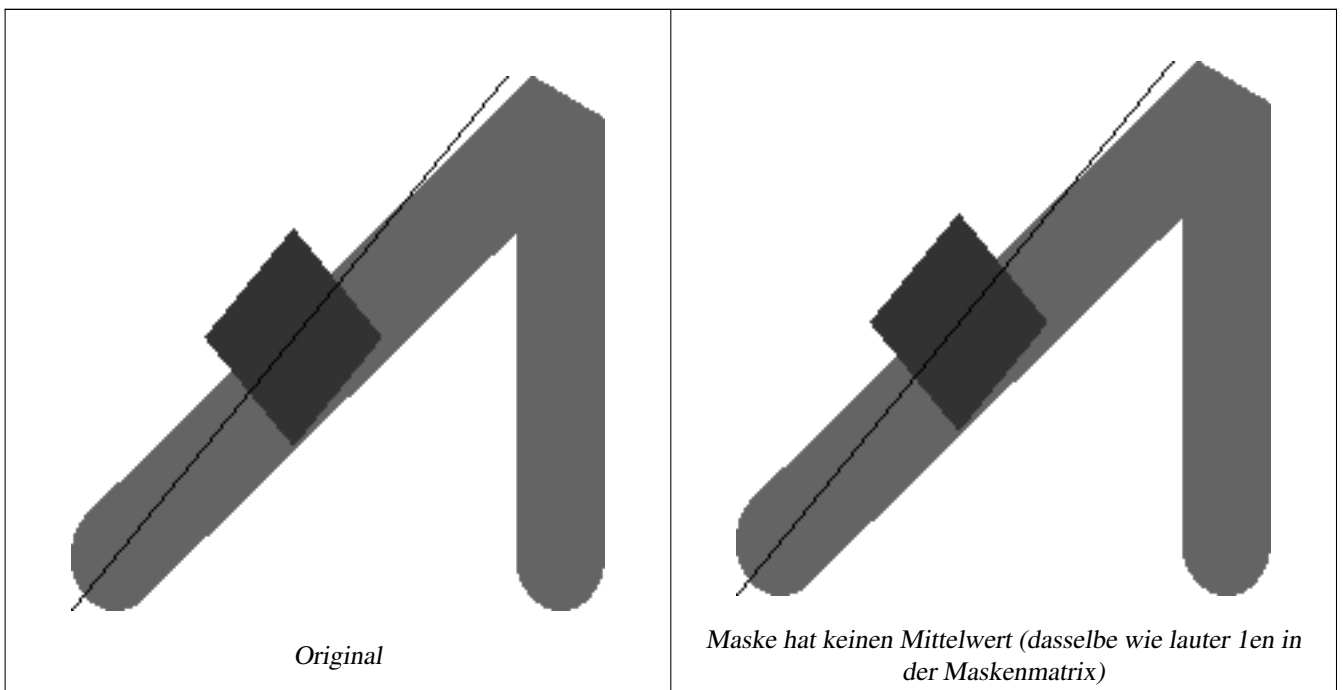
```

### Beispiele: Verwendung von Masken

```

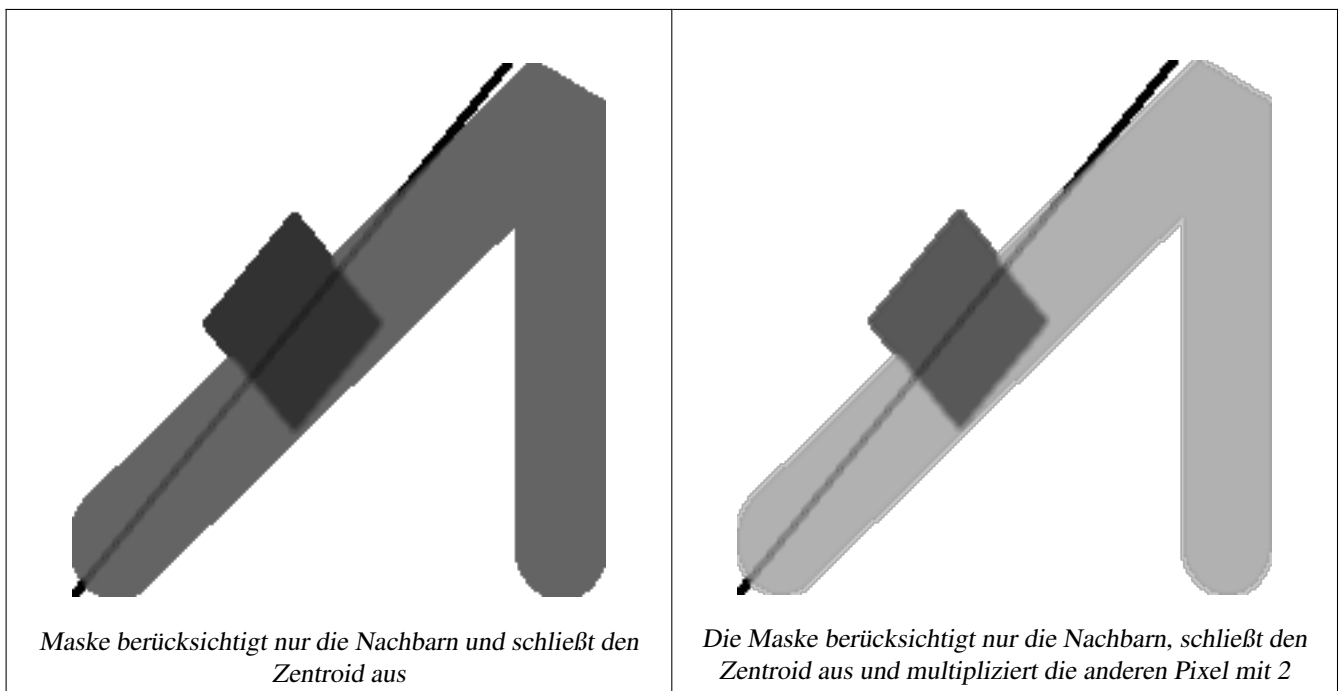
WITH foo AS (SELECT
    ST_SetBandNoDataValue(
    ST_SetValue(ST_AsRaster(
        ST_Buffer(
            ST_GeomFromText('LINESTRING(50 50,100 90,100 50)'), 5,'join=bevel'),
            200,200,ARRAY['8BUI'], ARRAY[100], ARRAY[0]), ST_Buffer('POINT(70 70)':: ←
                geometry,10,'quad_segs=1') ,50),
        'LINESTRING(20 20, 100 100, 150 98)'::geometry,1),0) AS rast )
SELECT 'original' AS title, rast
FROM foo
UNION ALL
SELECT 'no mask mean value' AS title, ST_MapAlgebra(rast,1,'ST_mean4ma(double precision[], ←
    int[], text[])':::regprocedure) AS rast
FROM foo
UNION ALL
SELECT 'mask only consider neighbors, exclude center' AS title, ST_MapAlgebra(rast,1,' ←
    ST_mean4ma(double precision[], int[], text[])':::regprocedure,
    '{{1,1,1}, {1,0,1}, {1,1,1}}'::double precision[], false) As rast
FROM foo
UNION ALL
SELECT 'mask weighted only consider neighbors, exclude center multi other pixel values by ←
    2' AS title, ST_MapAlgebra(rast,1,'ST_mean4ma(double precision[], int[], text[])':: ←
    regprocedure,
    '{{2,2,2}, {2,0,2}, {2,2,2}}'::double precision[], true) As rast
FROM foo;

```



*Original*

*Maske hat keinen Mittelwert (dasselbe wie lauter 1en in der Maskenmatrix)*

**Siehe auch**

[rastbandarg](#), [ST\\_Union](#), [ST\\_MapAlgebra \(expression version\)](#)

**11.12.6 ST\_MapAlgebra (expression version)**

`ST_MapAlgebra (expression version)` — Version mit Ausdrücken - Gibt für einen oder zwei Ausgangsraster, Bandindizes und einer oder mehreren vom Anwender vorgegebenen SQL-Ausdrücken, einen Raster mit einem Band zurück.

**Synopsis**

```
raster ST_MapAlgebra(raster rast, integer nband, text pixeltype, text expression, double precision nodataval=NULL);
raster ST_MapAlgebra(raster rast, text pixeltype, text expression, double precision nodataval=NULL);
raster ST_MapAlgebra(raster rast1, integer nband1, raster rast2, integer nband2, text expression, text pixeltype=NULL, text
extenttype=INTERSECTION, text nodata1expr=NULL, text nodata2expr=NULL, double precision nodatanodataval=NULL);
raster ST_MapAlgebra(raster rast1, raster rast2, text expression, text pixeltype=NULL, text extenttype=INTERSECTION, text
nodata1expr=NULL, text nodata2expr=NULL, double precision nodatanodataval=NULL);
```

**Beschreibung**

Version mit Ausdrücken - Gibt für einen oder zwei Ausgangsraster, Bandindizes und einer oder mehreren vom Anwender vorgegebenen SQL-Ausdrücken, einen Raster mit einem Band zurück.

Verfügbarkeit: 2.1.0

**Beschreibung: Variante 1 und 2 (ein Raster)**

Erstellt ein neues Rasterband indem eine gültige algebraische PostgreSQL Operation (`expression`) auf den Ausgangsraster (`rast`) angewendet wird. Wenn `nband` nicht angegeben ist, wird Band 1 angenommen. Der Zielraster hat die selbe Georeferenzierung, Breite und Höhe wie der ursprüngliche Raster, hat aber nur ein Band.

Wenn `pixeltype` angegeben ist, dann hat das Band des neuen Rasters diesen Pixeltyp. Wenn für den Pixeltyp NULL übergeben wird, dann hat das neue Rasterband denselben Pixeltyp wie das gegebene Band von `rast`

- Erlaubte Schlüsselwörter für `expression`
  1. `[rast]` - Zellwert der Pixel von Interesse
  2. `[rast.val]` - Zellwert der Pixel von Interesse
  3. `[rast.x]` - Rasterspalte (von 1 wegzählend) der Pixel von Interesse
  4. `[rast.y]` - Rasterzeile (von 1 wegzählend) der Pixel von Interesse

#### Beschreibung: Variante 3 und 4 (zwei Raster)

Erstellt einen neuen Raster mit einem Band, indem eine gültige algebraische PostgreSQL Operation auf die beiden, durch den Ausdruck `expression` bestimmten Ausgangsrasterbänder `rast1` und `rast2`) angewendet wird. Wenn `band1` oder `band2` nicht angegeben ist, wird Band 1 angenommen. Der Zielraster wird an dem Gitter des ersten Raster ausgerichtet (Größe, Versatz und Eckpunkte der Pixel). Die Ausdehnung des Zielrasters wird durch den Parameter `extenttype` bestimmt.

**expression** Ein algebraischer PostgreSQL Ausdruck, der zwei Raster und in PostgreSQL definierte Funktionen/Operatoren einbezieht, die den Pixelwert für sich schneidende Pixel festlegt. z.B. `(([rast1] + [rast2])/2.0)::integer`

**pixeltype** Der resultierende Pixeltyp des Zielraster muss entweder aus **ST\_BandPixelType** sein, weggelassen oder auf NULL gesetzt werden. Wenn er nicht übergeben wird oder auf NULL gesetzt ist, wird er standardmäßig auf den Pixeltyp des ersten Raster gesetzt.

**extenttype** Bestimmt die Ausdehnung des resultierenden Raster

1. `INTERSECTION` - Die Ausdehnung des neuen Raster entspricht der Schnittmenge der beiden Raster. Die 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 jener des zweiten Raster.

**nodataexpr** Ein algebraischer Ausdruck der nur `rast2` oder eine Konstante einbezieht. Dieser bestimmt was zurückgegeben wird, wenn die Pixel von `rast1` NODATA Werte sind und die räumlich übereinstimmenden Pixel von `rast2` Werte aufweisen.

**nodata2expr** Ein algebraischer Ausdruck der nur `rast1` oder eine Konstante einbezieht. Dieser bestimmt was zurückgegeben wird, wenn die Pixel von `rast2` NODATA Werte haben und die räumlich übereinstimmenden Pixel von `rast1` Werte aufweisen.

**nodatanodataval** Eine numerische Konstante die ausgegeben wird, wenn die übereinstimmenden Pixel der beiden Raster "rast1" und "rast2" nur NODATA Werte enthalten.

- Zugelassene Schlüsselwörter in `expression`, `nodataexpr` und `nodata2expr`
  1. `[rast1]` - Zellwert der Pixel von Interesse von `rast1`
  2. `[rast1.val]` - Zellwert der Pixel von Interesse von `rast1`
  3. `[rast1.x]` - Rasterspalte (von 1 wegzählend) der Pixel von Interesse von `rast1`
  4. `[rast1.y]` - Rasterzeile (von 1 wegzählend) der Pixel von Interesse von `rast1`
  5. `[rast2]` - Zellwert der Pixel von Interesse von `rast2`
  6. `[rast2.val]` - Zellwert der Pixel von Interesse von `rast2`
  7. `[rast2.x]` - Rasterspalte (von 1 wegzählend) der Pixel von Interesse von `rast2`
  8. `[rast2.y]` - Rasterzeile (von 1 wegzählend) der Pixel von Interesse von `rast2`

**Beispiele: Varianten 1 und 2**

```
WITH foo AS (
  SELECT ST_AddBand(ST_MakeEmptyRaster(10, 10, 0, 0, 1, 1, 0, 0, 0), '32BF'::text, 1, -1) ←
    AS rast
)
SELECT
  ST_MapAlgebra(rast, 1, NULL, 'ceil([rast]*[rast.x]/[rast.y]+[rast.val])')
FROM foo;
```

**Beispiele: Varianten 3 und 4**

```
WITH foo AS (
  SELECT 1 AS rid, ST_AddBand(ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(2, 2, 0, 0, 1, -1, ←
    0, 0, 0), 1, '16BUI', 1, 0), 2, '8BUI', 10, 0), 3, '32BUI'::text, 100, 0) AS rast ←
    UNION ALL
  SELECT 2 AS rid, ST_AddBand(ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(2, 2, 0, 1, 1, -1, ←
    0, 0, 0), 1, '16BUI', 2, 0), 2, '8BUI', 20, 0), 3, '32BUI'::text, 300, 0) AS rast
)
SELECT
  ST_MapAlgebra(
    t1.rast, 2,
    t2.rast, 1,
    '([rast2] + [rast1.val]) / 2'
  ) AS rast
FROM foo t1
CROSS JOIN foo t2
WHERE t1.rid = 1
  AND t2.rid = 2;
```

**Siehe auch**

[rastbandarg](#), [ST\\_Union](#), [ST\\_MapAlgebra \(callback function version\)](#)

**11.12.7 ST\_MapAlgebraExpr**

**ST\_MapAlgebraExpr** — Version mit 1 Rasterband: Erzeugt ein neues Rasterband, dass über eine gültige, algebraische PostgreSQL Operation für ein Rasterband mit gegebenen Pixeltyp erstellt wird. Wenn kein Band bestimmt ist, wird Band 1 angenommen.

**Synopsis**

```
raster ST_MapAlgebraExpr(raster rast, integer band, text pixeltype, text expression, double precision nodataval=NULL);
raster ST_MapAlgebraExpr(raster rast, text pixeltype, text expression, double precision nodataval=NULL);
```

**Beschreibung****Warning**

**ST\_MapAlgebraExpr** Seit der Version 2.1.0 überholt. Benutzen Sie stattdessen bitte [ST\\_MapAlgebra \(expression version\)](#) .

Erstellt ein neues Rasterband indem eine gültige algebraische PostgreSQL Operation (*expression*) auf den Ausgangsraster (*rast*) angewendet wird. Wenn kein *band* festgelegt ist, wird Band 1 angenommen. Der Zielraster hat die selbe Georeferenzierung, Breite und Höhe wie der ursprüngliche Raster, hat aber nur ein Band.

Wenn *pixeltype* angegeben ist, dann hat das Band des neuen Rasters diesen Pixeltyp. Wenn für den Pixeltyp NULL übergeben wird, dann hat das neue Rasterband denselben Pixeltyp wie das gegebene Band von *rast*

Im Ausdruck können Sie folgende Terme verwenden: *[rast]* für den Zellwert des ursprünglichen Bandes, *[rast.x]* für den von 1 wegzählenden Index der Rasterspalten, *[rast.y]* für den von 1 wegzählenden Index der Rasterzeilen.

Verfügbarkeit: 2.0.0

## Beispiele

Erzeugt einen Einzelbandraster, der sich durch die Anwendung der Funktion "modulo 2" auf das ursprüngliche Rasterband ergibt.

```
ALTER TABLE dummy_rast ADD COLUMN map_rast raster;
UPDATE dummy_rast SET map_rast = ST_MapAlgebraExpr(rast,NULL,'mod([rast]::numeric,2)') ←
WHERE rid = 2;

SELECT
  ST_Value(rast,1,i,j) As origval,
  ST_Value(map_rast, 1, i, j) As mapval
FROM dummy_rast
CROSS JOIN generate_series(1, 3) AS i
CROSS JOIN generate_series(1,3) AS j
WHERE rid = 2;
```

origval	mapval
253	1
254	0
253	1
253	1
254	0
254	0
250	0
254	0
254	0

Erzeugt einen neuen Einzelbandraster mit dem Pixeltyp 2BUI. Der ursprüngliche Raster wird dabei neu klassifiziert und mit einem NODATA Wert von 0 versehen.

```
ALTER TABLE dummy_rast ADD COLUMN map_rast2 raster;
UPDATE dummy_rast SET
  map_rast2 = ST_MapAlgebraExpr(rast,'2BUI'::text,'CASE WHEN [rast] BETWEEN 100 and 250 ←
  THEN 1 WHEN [rast] = 252 THEN 2 WHEN [rast] BETWEEN 253 and 254 THEN 3 ELSE 0 END':: ←
  text, '0')
WHERE rid = 2;

SELECT DISTINCT
  ST_Value(rast,1,i,j) As origval,
  ST_Value(map_rast2, 1, i, j) As mapval
FROM dummy_rast
CROSS JOIN generate_series(1, 5) AS i
CROSS JOIN generate_series(1,5) AS j
WHERE rid = 2;
```

origval	mapval
249	1
250	1



```

251 |
252 |     2
253 |     3
254 |     3

```

```

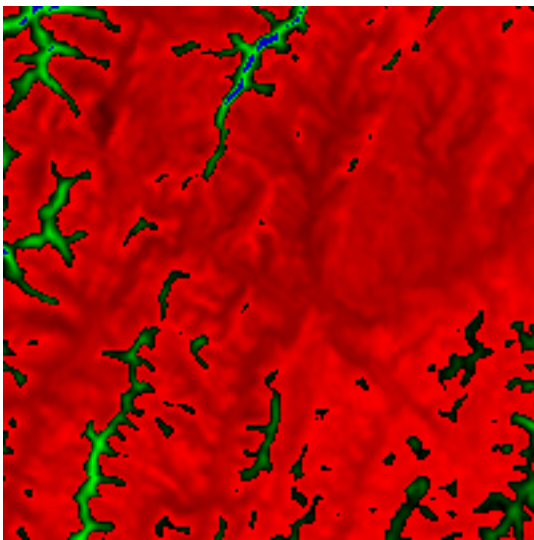
SELECT
    ST_BandPixelType(map_rast2) As b1pixtyp
FROM dummy_rast
WHERE rid = 2;

```

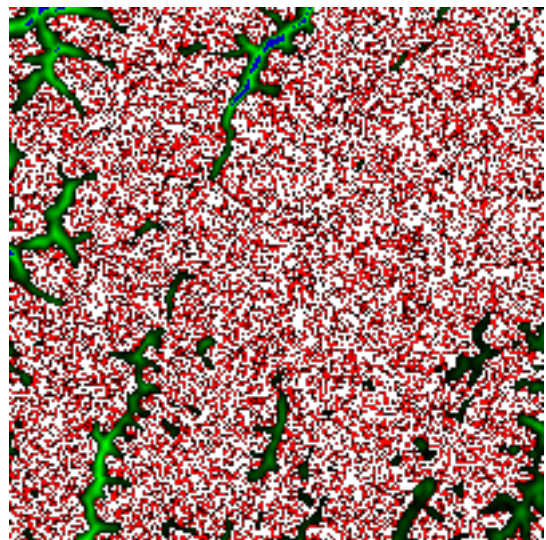
```

b1pixtyp
-----
2BUI

```



*Original (Spalte "rast\_view")*



*rast\_view\_ma*

Erzeugt einen neuen Raster mit 3 Bändern und demselben Pixeltyp als unser ursprünglicher Raster mit 3 Bändern. Das erste Band wird über einen Map Algebra Ausdruck geändert, die verbleibenden 2 Bänder sind unverändert.

```

SELECT
    ST_AddBand(
        ST_AddBand(
            ST_AddBand(
                ST_MakeEmptyRaster(rast_view),
                ST_MapAlgebraExpr(rast_view,1,NULL,'tan([rast])*[rast]')
            ),
            ST_Band(rast_view,2)
        ),
        ST_Band(rast_view, 3)
    ) As rast_view_ma
FROM wind
WHERE rid=167;

```

#### Siehe auch

[ST\\_MapAlgebraExpr](#), [ST\\_MapAlgebraFct](#), [ST\\_BandPixelType](#), [ST\\_GeoReference](#), [ST\\_Value](#)

### 11.12.8 ST\_MapAlgebraExpr

**ST\_MapAlgebraExpr** — Version mit 2 Rasterbändern: Erstellt einen neuen Einzelbandraster, indem eine gültige algebraische PostgreSQL Funktion auf die zwei Ausgangsrasterbänder und den entsprechenden Pixeltyp angewendet wird. Wenn keine Bandnummern angegeben sind, wird von jedem Raster Band 1 angenommen. Der Ergebnisraster wird nach dem Gitter des ersten Raster ausgerichtet (Skalierung, Versatz und Eckpunkte der Pixel) und hat die Ausdehnung, welche durch den Parameter "extenttype" definiert ist. Der Parameter "extenttype" kann die Werte INTERSECTION, UNION, FIRST, SECOND annehmen.

#### Synopsis

```
raster ST_MapAlgebraExpr(raster rast1, raster rast2, text expression, text pixeltype=same_as_rast1_band, text extenttype=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 (expression version)** .

Erstellt einen neuen Raster mit einem Band, indem eine gültige algebraische PostgreSQL Operation auf die beiden, durch den Ausdruck `expression` bestimmten Ausgangsrasterbänder `rast1` und `rast2`) angewendet wird. Wenn `band1` oder `band2` nicht angegeben ist, wird Band 1 angenommen. Der Zielraster wird an dem Gitter des ersten Raster ausgerichtet (Größe, Versatz und Eckpunkte der Pixel). Die Ausdehnung des Zielrasters wird durch den Parameter `extenttype` bestimmt.

**expression** Ein algebraischer PostgreSQL Ausdruck, der zwei Raster und in PostgreSQL definierte Funktionen/Operatoren einbezieht, die den Pixelwert für sich schneidende Pixel festlegt. z.B. `(([rast1] + [rast2])/2.0)::integer`

**pixeltype** Der resultierende Pixeltyp des Zielraster muss entweder aus **ST\_BandPixelType** sein, weggelassen oder auf NULL gesetzt werden. Wenn er nicht übergeben wird oder auf NULL gesetzt ist, wird er standardmäßig auf den Pixeltyp des ersten Raster gesetzt.

**extenttype** Bestimmt die Ausdehnung des resultierenden Raster

1. **INTERSECTION** - Die Ausdehnung des neuen Raster entspricht der Schnittmenge der beiden Raster. Die Standardeinstellung.
2. **UNION** - Die Ausdehnung des neuen Raster entspricht der Vereinigungsmenge der beiden Raster.
3. **FIRST** - Die Ausdehnung des neuen Raster entspricht jener des ersten Raster.
4. **SECOND** - Die Ausdehnung des neuen Raster entspricht jener des zweiten Raster.

**nodata1expr** Ein algebraischer Ausdruck der nur `rast2` oder eine Konstante einbezieht. Dieser bestimmt was zurückgegeben wird, wenn die Pixel von `rast1` NODATA Werte sind und die räumlich übereinstimmenden Pixel von `rast2` Werte aufweisen.

**nodata2expr** Ein algebraischer Ausdruck der nur `rast1` oder eine Konstante einbezieht. Dieser bestimmt was zurückgegeben wird, wenn die Pixel von `rast2` NODATA Werte haben und die räumlich übereinstimmenden Pixel von `rast1` Werte aufweisen.

**nodatanodataval** Eine numerische Konstante die ausgegeben wird, wenn die übereinstimmenden Pixel der beiden Raster "rast1" und "rast2" nur NODATA Werte enthalten.

Wenn `pixeltype` angegeben ist, dann hat das Band des neuen Rasters diesen Pixeltyp. Wenn für den Pixeltyp NULL übergeben wird oder der Pixeltyp nicht festgelegt ist, dann hat das neue Rasterband denselben Pixeltyp wie das angegebene Band von `rast1`. Sie können den Ausdruck `[rast1.val] [rast2.val]` verwenden, um auf die Pixelwerte der ursprünglichen Rasterbänder zu verweisen, und `[rast1.x]`, `[rast1.y]` etc. um auf die Positionen der Pixel über die Rasterspalten / Rasterzeilen zu verweisen.

Verfügbarkeit: 2.0.0

### Beispiel: Verschneidung und Union von 2 Bändern

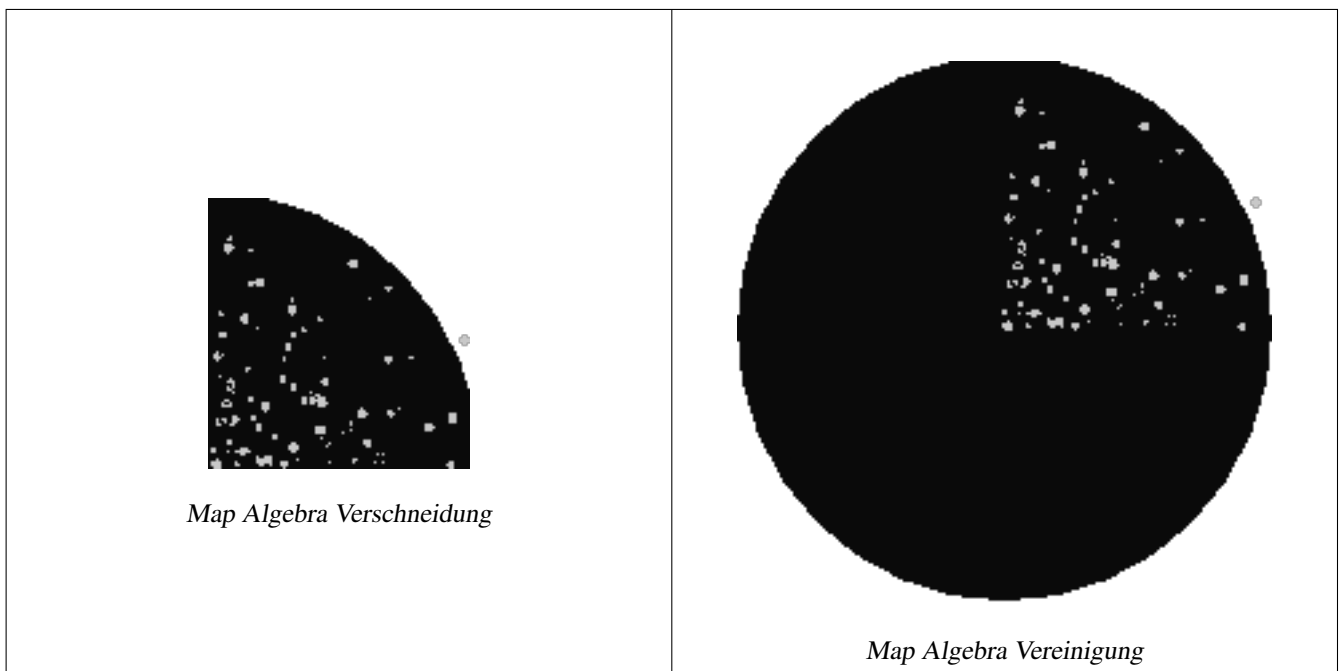
Erzeugt einen Einzelbandraster, der sich durch die Anwendung der Funktion "modulo 2" auf das ursprüngliche Rasterband ergibt.

```
--Create a cool set of rasters --
DROP TABLE IF EXISTS fun_shapes;
CREATE TABLE fun_shapes(rid serial PRIMARY KEY, fun_name text, rast raster);

-- Insert some cool shapes around Boston in Massachusetts state plane meters --
INSERT INTO fun_shapes(fun_name, rast)
VALUES ('ref', ST_AsRaster(ST_MakeEnvelope(235229, 899970, 237229, 901930,26986),200,200,'8BUI',0,0));

INSERT INTO fun_shapes(fun_name,rast)
WITH ref(rast) AS (SELECT rast FROM fun_shapes WHERE fun_name = 'ref' )
SELECT 'area' AS fun_name, ST_AsRaster(ST_Buffer(ST_SetSRID(ST_Point(236229, 900930),26986) ←
, 1000),
ref.rast,'8BUI', 10, 0) As rast
FROM ref
UNION ALL
SELECT 'rand bubbles',
ST_AsRaster(
(SELECT ST_Collect(geom)
FROM (SELECT ST_Buffer(ST_SetSRID(ST_Point(236229 + i*random()*100, 900930 + j*random() ←
*100),26986), random()*20) As geom
FROM generate_series(1,10) As i, generate_series(1,10) As j
) As foo ), ref.rast,'8BUI', 200, 0)
FROM ref;

--map them -
SELECT ST_MapAlgebraExpr(
area.rast, bub.rast, '[rast2.val]', '8BUI', 'INTERSECTION', '[rast2.val]', '[rast1. ←
val]') As interrast,
ST_MapAlgebraExpr(
area.rast, bub.rast, '[rast2.val]', '8BUI', 'UNION', '[rast2.val]', '[rast1.val ←
]') As unionrast
FROM
(SELECT rast FROM fun_shapes WHERE
fun_name = 'area') As area
CROSS JOIN (SELECT rast
FROM fun_shapes WHERE
fun_name = 'rand bubbles') As bub
```

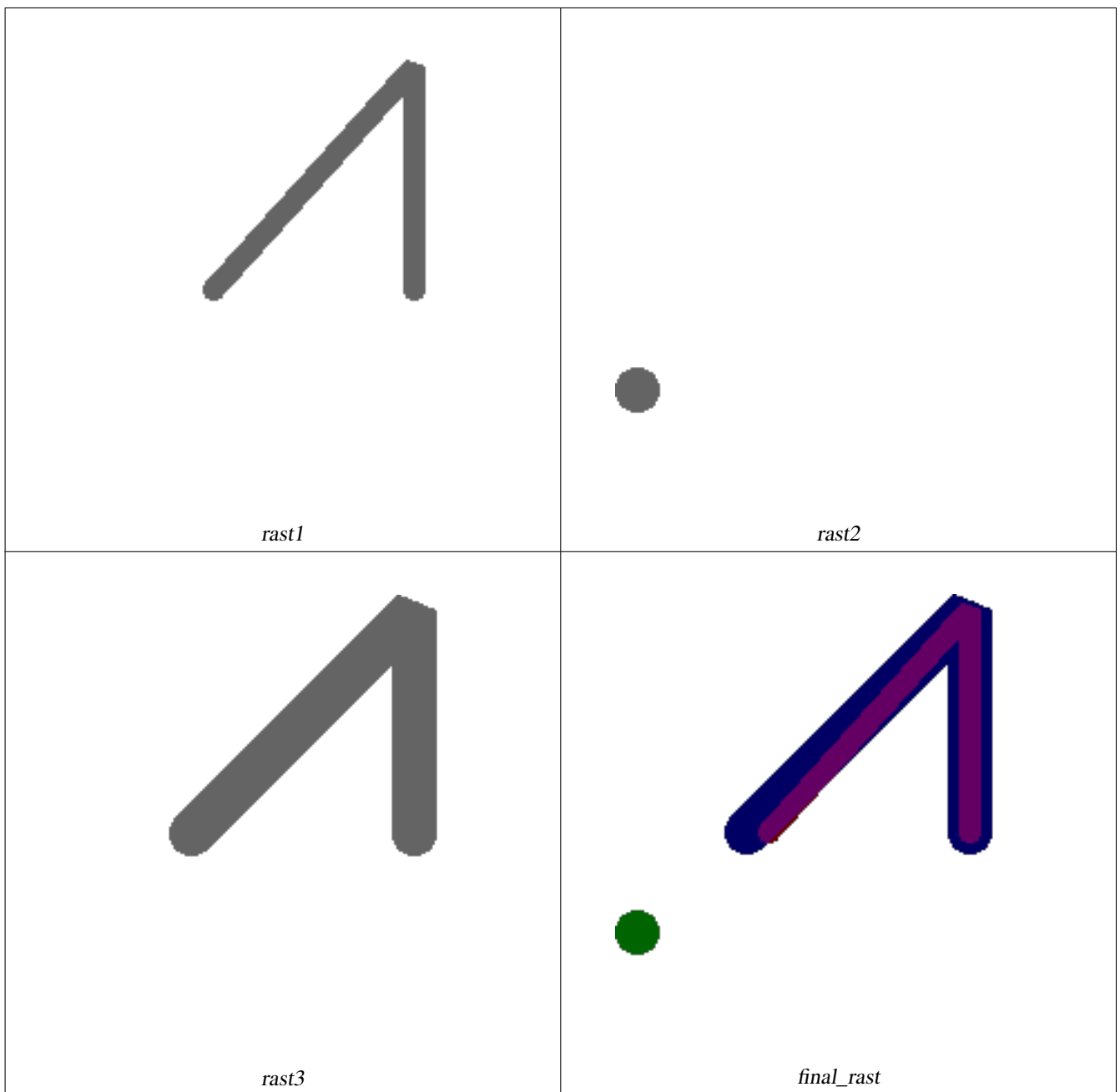


*Map Algebra Verschneidung*

*Map Algebra Vereinigung*

### Beispiel: Überlagerung von Rastern als Einzelbänder am Canvas

```
-- we use ST_AsPNG to render the image so all single band ones look grey --
WITH mygeoms
  AS ( SELECT 2 As bnum, ST_Buffer(ST_Point(1,5),10) As geom
        UNION ALL
        SELECT 3 AS bnum,
              ST_Buffer(ST_GeomFromText('LINESTRING(50 50,150 150,150 50)'), 10,'join= ↵
              bevel') As geom
        UNION ALL
        SELECT 1 As bnum,
              ST_Buffer(ST_GeomFromText('LINESTRING(60 50,150 150,150 50)'), 5,'join= ↵
              bevel') As geom
      ),
  -- define our canvas to be 1 to 1 pixel to geometry
  canvas
  AS (SELECT ST_AddBand(ST_MakeEmptyRaster(200,
    200,
    ST_XMin(e)::integer, ST_YMax(e)::integer, 1, -1, 0, 0) , '8BUI'::text,0) As rast
    FROM (SELECT ST_Extent(geom) As e,
            Max(ST_SRID(geom)) As srid
          from mygeoms
        ) As foo
  ),
  rbands AS (SELECT ARRAY(SELECT ST_MapAlgebraExpr(canvas.rast, ST_AsRaster(m.geom, canvas ↵
    .rast, '8BUI', 100),
    '[rast2.val]', '8BUI', 'FIRST', '[rast2.val]', '[rast1.val]') As rast
    FROM mygeoms AS m CROSS JOIN canvas
    ORDER BY m.bnum) As rasts
  )
  SELECT rasts[1] As rast1 , rasts[2] As rast2, rasts[3] As rast3, ST_AddBand(
    ST_AddBand(rasts[1],rasts[2]), rasts[3]) As final_rast
  FROM rbands;
```



### Beispiel: Ein Orthophoto mit einer 2 Meter breiten Begrenzung der ausgewählten Grundstücke überlagern

```
-- Create new 3 band raster composed of first 2 clipped bands, and overlay of 3rd band with ←
  our geometry
-- This query took 3.6 seconds on PostGIS windows 64-bit install
WITH pr AS
-- Note the order of operation: we clip all the rasters to dimensions of our region
(SELECT ST_Clip(rast,ST_Expand(geom,50) ) As rast, g.geom
 FROM aerials.o_2_boston AS r INNER JOIN
-- union our parcels of interest so they form a single geometry we can later intersect with
  (SELECT ST_Union(ST_Transform(geom,26986)) AS geom
   FROM landparcels WHERE pid IN('0303890000', '0303900000')) As g
   ON ST_Intersects(rast::geometry, ST_Expand(g.geom,50))
),
-- we then union the raster shards together
```

```

-- ST_Union on raster is kinda of slow but much faster the smaller you can get the rasters
-- therefore we want to clip first and then union
prunion AS
(SELECT ST_AddBand(NULL, ARRAY[ST_Union(rast,1),ST_Union(rast,2),ST_Union(rast,3)] ) As ←
  clipped,geom
FROM pr
GROUP BY geom)
-- return our final raster which is the unioned shard with
-- with the overlay of our parcel boundaries
-- add first 2 bands, then mapalgebra of 3rd band + geometry
SELECT ST_AddBand(ST_Band(clipped,ARRAY[1,2])
  , ST_MapAlgebraExpr(ST_Band(clipped,3), ST_AsRaster(ST_Buffer(ST_Boundary(geom),2), ←
  clipped, '8BUI',250),
  '[rast2.val]', '8BUI', 'FIRST', '[rast2.val]', '[rast1.val]')) ) As rast
FROM prunion;

```



Die blauen Linien sind die Ränder der ausgewählten Grundstücke.

#### Siehe auch

[ST\\_MapAlgebraExpr](#), [ST\\_AddBand](#), [ST\\_AsPNG](#), [ST\\_AsRaster](#), [ST\\_MapAlgebraFct](#), [ST\\_BandPixelType](#), [ST\\_GeoReference](#), [ST\\_Value](#), [ST\\_Union](#), [ST\\_Union](#)

### 11.12.9 ST\_MapAlgebraFct

`ST_MapAlgebraFct` — Version mit 1 Rasterband: Erzeugt ein neues Rasterband, dass über eine gültige PostgreSQL Funktion für ein gegebenes Rasterband und Pixeltyp erstellt wird. Wenn kein Band bestimmt ist, wird Band 1 angenommen.

#### Synopsis

```

raster ST_MapAlgebraFct(raster rast, regprocedure onerasteruserfunc);
raster ST_MapAlgebraFct(raster rast, regprocedure onerasteruserfunc, text[] VARIADIC args);

```

```
raster ST_MapAlgebraFct(raster rast, text pixeltype, regprocedure onerasteruserfunc);
raster ST_MapAlgebraFct(raster rast, text pixeltype, regprocedure onerasteruserfunc, text[] VARIADIC args);
raster ST_MapAlgebraFct(raster rast, integer band, regprocedure onerasteruserfunc);
raster ST_MapAlgebraFct(raster rast, integer band, regprocedure onerasteruserfunc, text[] VARIADIC args);
raster ST_MapAlgebraFct(raster rast, integer band, text pixeltype, regprocedure onerasteruserfunc);
raster ST_MapAlgebraFct(raster rast, integer band, text pixeltype, regprocedure onerasteruserfunc, text[] VARIADIC args);
```

## Beschreibung



### Warning

**ST\_MapAlgebraFct** Seit der Version 2.1.0 überholt. Benutzen Sie stattdessen bitte **ST\_MapAlgebra (callback function version)**

Erstellt einen neuen Raster mit einem Band, indem eine gültige PostgreSQL Funktion durch `tworasteruserfunc` spezifiziert und auf den gegebenen Raster (`rast`) angewendet wird. Wenn kein Band angegeben ist, wird Band 1 angenommen. Der Zielraster hat die selbe Georeferenzierung, Breite und Höhe wie der ursprüngliche Raster, hat aber nur ein Band.

Wenn `pixeltype` angegeben ist, dann hat das Band des neuen Rasters diesen Pixeltyp. Wenn für den Pixeltyp NULL übergeben wird, dann hat das neue Rasterband denselben Pixeltyp wie das gegebene Band von `rast`

Der Parameter `onerasteruserfunc` muss der Name und die Signatur einer SQL- oder PL/pgSQL-Funktion sein, die in eine `regprocedure` umgewandelt wird. Ein sehr einfaches und ziemlich nutzloses PL/pgSQL-Funktionsbeispiel ist:

```
CREATE OR REPLACE FUNCTION simple_function(pixel FLOAT, pos INTEGER[], VARIADIC args TEXT ↔
[])
RETURNS FLOAT
AS $$ BEGIN
    RETURN 0.0;
END; $$
LANGUAGE 'plpgsql' IMMUTABLE;
```

Die Benutzerfunktion kann zwei oder drei Argumente akzeptieren: einen Float-Wert, ein optionales Integer-Array und ein variables Text-Array. Das erste Argument ist der Wert einer einzelnen Rasterzelle (unabhängig vom Rasterdatentyp). Das zweite Argument ist die Position der aktuellen Verarbeitungszelle in der Form `{x,y}`. Das dritte Argument gibt an, dass alle übrigen Parameter an **ST\_MapAlgebraFct** an die Benutzerfunktion weitergegeben werden sollen.

Die Übergabe eines `regprocedure` Arguments an eine SQL-Funktion erfordert die Übergabe der vollständigen Funktionssignatur und die anschließende Umwandlung in einen `regprocedure` Typ. Um die obige Beispiel-PL/pgSQL-Funktion als Argument zu übergeben, lautet das SQL für das Argument:

```
'simple_function(float,integer[],text[])'::regprocedure
```

Beachten Sie, dass das Argument den Namen der Funktion, die Typen der Funktionsargumente, Anführungszeichen um den Namen und die Argumenttypen sowie einen Cast in eine `regprocedure` enthält.

Der dritte Übergabewert an die Funktion `userfunction` ist ein Feld vom Datentyp `variadic text`. Alle an einen Funktionsaufruf von **ST\_MapAlgebraFct** angehängten Parameter werden an die spezifizierte `userfunction` durchgereicht und sind in dem Übergabewert `args` enthalten.



### Note

Für nähere Information zu dem Schlüsselwort VARIADIC, sehen Sie bitte die PostgreSQL Dokumentation und den Abschnitt "SQL Functions with Variable Numbers of Arguments" unter [Query Language \(SQL\) Functions](#).

**Note**

Der Übergabewert `text[]` an die `userfunction` ist auch dann erforderlich, wenn Sie sonst keine Argumente für die Auswertung an an "userfunction" übergeben.

Verfügbarkeit: 2.0.0

**Beispiele**

Erzeugt einen Einzelbandraster, der sich durch die Anwendung der Funktion "modulo 2" auf das ursprüngliche Rasterband ergibt.

```
ALTER TABLE dummy_rast ADD COLUMN map_rast raster;
CREATE FUNCTION mod_fct(pixel float, pos integer[], variadic args text[])
RETURNS float
AS $$
BEGIN
    RETURN pixel::integer % 2;
END;
$$
LANGUAGE 'plpgsql' IMMUTABLE;

UPDATE dummy_rast SET map_rast = ST_MapAlgebraFct(rast,NULL,'mod_fct(float,integer[],text ←
    [])'::regprocedure) WHERE rid = 2;

SELECT ST_Value(rast,1,i,j) As origval, ST_Value(map_rast, 1, i, j) As mapval
FROM dummy_rast CROSS JOIN generate_series(1, 3) AS i CROSS JOIN generate_series(1,3) AS j
WHERE rid = 2;
```

origval	mapval
253	1
254	0
253	1
253	1
254	0
254	0
250	0
254	0
254	0

Erzeugt einen neuen Raster mit einem Band und mit dem Pixeltyp 2BUI. Der ursprüngliche Raster wird dabei neu klassifiziert und der NODATA Wert wird an die "user function" (0) übergeben.

```
ALTER TABLE dummy_rast ADD COLUMN map_rast2 raster;
CREATE FUNCTION classify_fct(pixel float, pos integer[], variadic args text[])
RETURNS float
AS
$$
DECLARE
    nodata float := 0;
BEGIN
    IF NOT args[1] IS NULL THEN
        nodata := args[1];
    END IF;
    IF pixel < 251 THEN
        RETURN 1;
    ELSIF pixel = 252 THEN
        RETURN 2;
    ELSIF pixel
> 252 THEN
```



```

        RETURN 3;
    ELSE
        RETURN nodata;
    END IF;
END;
$$
LANGUAGE 'plpgsql';
UPDATE dummy_rast SET map_rast2 = ST_MapAlgebraFct(rast,'2BUI','classify_fct(float,integer ←
    [],text[])'::regprocedure, '0') WHERE rid = 2;

SELECT DISTINCT ST_Value(rast,1,i,j) As origval, ST_Value(map_rast2, 1, i, j) As mapval
FROM dummy_rast CROSS JOIN generate_series(1, 5) AS i CROSS JOIN generate_series(1,5) AS j
WHERE rid = 2;

```

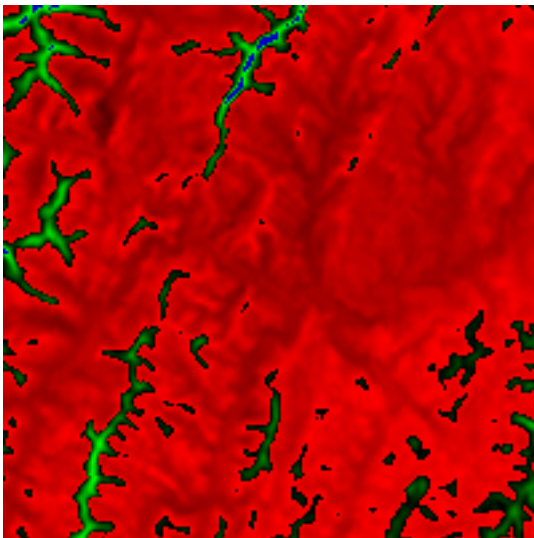
origval	mapval
249	1
250	1
251	1
252	2
253	3
254	3

```

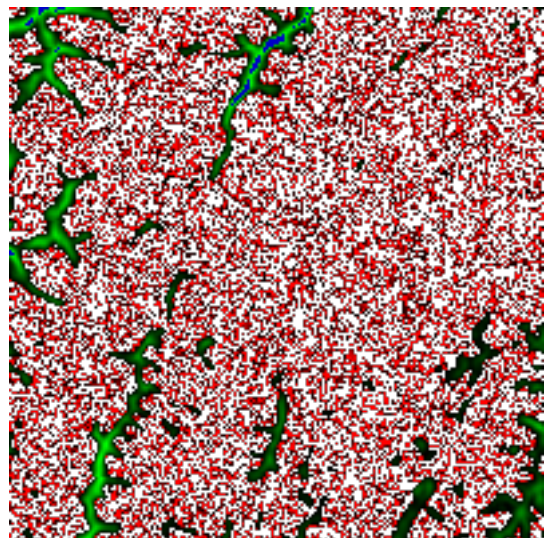
SELECT ST_BandPixelType(map_rast2) As b1pixtyp
FROM dummy_rast WHERE rid = 2;

```

b1pixtyp
2BUI



*Original (Spalte "rast\_view")*



*rast\_view\_ma*

Erzeugt einen neuen Raster mit 3 Bändern und demselben Pixeltyp als unser ursprünglicher Raster mit 3 Bändern. Das erste Band wird über einen Map Algebra Ausdruck geändert, die verbleibenden 2 Bänder sind unverändert.

```

CREATE FUNCTION rast_plus_tan(pixel float, pos integer[], variadic args text[])
RETURNS float
AS

```

```

$$
BEGIN
    RETURN tan(pixel) * pixel;
END;
$$
LANGUAGE 'plpgsql';

SELECT ST_AddBand(
    ST_AddBand(
        ST_AddBand(
            ST_MakeEmptyRaster(rast_view),
            ST_MapAlgebraFct(rast_view, 1, NULL, 'rast_plus_tan(float, integer[], text[])':: regprocedure)
        ),
        ST_Band(rast_view, 2)
    ),
    ST_Band(rast_view, 3) As rast_view_ma
)
FROM wind
WHERE rid=167;

```

**Siehe auch**

[ST\\_MapAlgebraExpr](#), [ST\\_BandPixelType](#), [ST\\_GeoReference](#), [ST\\_SetValue](#)

**11.12.10 ST\_MapAlgebraFct**

**ST\_MapAlgebraFct** — Version mit 2 Rasterbändern: Erstellt einen neuen Einzelbandraster, indem eine gültige PostgreSQL Funktion auf die 2 gegebenen Rasterbänder und den entsprechenden Pixeltyp angewendet wird. Wenn kein Band bestimmt ist, wird Band 1 angenommen. Wenn der "Extent"-Typ nicht angegeben ist, wird standardmäßig INTERSECTION angenommen.

**Synopsis**

```

raster ST_MapAlgebraFct(raster rast1, raster rast2, regprocedure tworastuserfunc, text pixeltype=same_as_rast1, text extenttype=INTERSECTION, text[] VARIADIC userargs);
raster ST_MapAlgebraFct(raster rast1, integer band1, raster rast2, integer band2, regprocedure tworastuserfunc, text pixeltype=same_as_rast1, text extenttype=INTERSECTION, text[] VARIADIC userargs);

```

**Beschreibung****Warning**

**ST\_MapAlgebraFct** Seit der Version 2.1.0 überholt. Benutzen Sie stattdessen bitte [ST\\_MapAlgebra \(callback function version\)](#)

Erstellt einen neuen Raster mit einem Band, indem eine gültige PostgreSQL Funktion (*tworastuserfunc*) auf die Ausgangsraster (*rast1*, *rast2*) angewendet wird. Wenn *band1* oder *band2* nicht angegeben ist, wird Band 1 angenommen. Der Zielraster hat die selbe Georeferenzierung, Breite und Höhe wie der ursprüngliche Raster, hat aber nur ein Band.

Wenn *pixeltype* angegeben ist, dann hat das Band des neuen Rasters diesen Pixeltyp. Wenn für den Pixeltyp NULL oder kein Wert übergeben wird, dann hat das neue Rasterband denselben Pixeltyp wie das angegebene Band von *rast1*

Der Parameter *tworastuserfunc* muss der Name und die Signatur einer SQL- oder PL/pgSQL-Funktion sein, die in eine *regprocedure* umgewandelt wird. Ein Beispiel für eine PL/pgSQL-Funktion ist:

```
CREATE OR REPLACE FUNCTION simple_function_for_two_rasters(pixel1 FLOAT, pixel2 FLOAT, pos ←
  INTEGER[], VARIADIC args TEXT[])
  RETURNS FLOAT
  AS $$ BEGIN
    RETURN 0.0;
  END; $$
LANGUAGE 'plpgsql' IMMUTABLE;
```

Die Funktion `tworastuserfunc` kann drei oder vier Argumente akzeptieren: einen Wert mit doppelter Genauigkeit, einen Wert mit doppelter Genauigkeit, ein optionales Integer-Array und ein variables Text-Array. Das erste Argument ist der Wert einer einzelnen Rasterzelle in `rast1` (unabhängig vom Rasterdatentyp). Das zweite Argument ist ein individueller Rasterzellenwert in `rast2`. Das dritte Argument ist die Position der aktuellen Verarbeitungszelle in der Form `'{x,y}'`. Das vierte Argument gibt an, dass alle übrigen Parameter von `ST_MapAlgebraFct` an die `tworastuserfunc` weitergereicht werden sollen.

Die Übergabe eines `regprocedure` Arguments an eine SQL-Funktion erfordert die Übergabe der vollständigen Funktionssignatur und die anschließende Umwandlung in einen `regprocedure` Typ. Um die obige Beispiel-PL/pgSQL-Funktion als Argument zu übergeben, lautet das SQL für das Argument:

```
'simple_function(double precision, double precision, integer[], text[])::regprocedure
```

Beachten Sie, dass das Argument den Namen der Funktion, die Typen der Funktionsargumente, Anführungszeichen um den Namen und die Argumenttypen sowie einen Cast in eine `regprocedure` enthält.

Der vierte Übergabewert an die Funktion `tworastuserfunc` ist ein Feld mit dem Datentyp `variadic text`. Alle an eine Funktion `ST_MapAlgebraFct` angehängten Parameter werden an die `tworastuserfunc` durchgereicht und sind in dem Übergabewert `userargs` enthalten.



#### Note

Für nähere Information zu dem Schlüsselwort `VARIADIC`, sehen Sie bitte die PostgreSQL Dokumentation und den Abschnitt "SQL Functions with Variable Numbers of Arguments" unter [Query Language \(SQL\) Functions](#).



#### Note

Der Übergabewert `text[]` an die `tworastuserfunc` ist auch dann erforderlich, wenn Sie sonst keine Argumente für die Auswertung an ein "userfunction" übergeben.

Verfügbarkeit: 2.0.0

### Beispiel: Überlagerung von Rastern als Einzelbänder am Canvas

```
-- define our user defined function --
CREATE OR REPLACE FUNCTION raster_mapalgebra_union(
  rast1 double precision,
  rast2 double precision,
  pos integer[],
  VARIADIC userargs text[]
)
  RETURNS double precision
  AS $$
  DECLARE
  BEGIN
    CASE
      WHEN rast1 IS NOT NULL AND rast2 IS NOT NULL THEN
        RETURN ((rast1 + rast2)/2.);
```

```

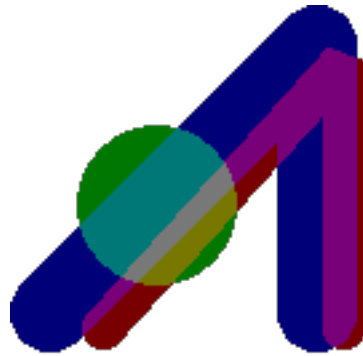
        WHEN rast1 IS NULL AND rast2 IS NULL THEN
            RETURN NULL;
        WHEN rast1 IS NULL THEN
            RETURN rast2;
        ELSE
            RETURN rast1;
    END CASE;

    RETURN NULL;
END;
$$ LANGUAGE 'plpgsql' IMMUTABLE COST 1000;

-- prep our test table of rasters
DROP TABLE IF EXISTS map_shapes;
CREATE TABLE map_shapes(rid serial PRIMARY KEY, rast raster, bnum integer, descrip text);
INSERT INTO map_shapes(rast,bnum, descrip)
WITH mygeoms
    AS ( SELECT 2 As bnum, ST_Buffer(ST_Point(90,90),30) As geom, 'circle' As descrip
        UNION ALL
        SELECT 3 AS bnum,
            ST_Buffer(ST_GeomFromText('LINESTRING(50 50,150 150,150 50)'), 15) As geom, ←
            'big road' As descrip
        UNION ALL
        SELECT 1 As bnum,
            ST_Translate(ST_Buffer(ST_GeomFromText('LINESTRING(60 50,150 150,150 50)'), ←
            8,'join=bevel'), 10,-6) As geom, 'small road' As descrip
    ),
-- define our canvas to be 1 to 1 pixel to geometry
canvas
AS ( SELECT ST_AddBand(ST_MakeEmptyRaster(250,
    250,
    ST_XMin(e)::integer, ST_YMax(e)::integer, 1, -1, 0, 0 ) , '8BUI'::text,0) As rast
    FROM (SELECT ST_Extent(geom) As e,
        Max(ST_SRID(geom)) As srid
        from mygeoms
    ) As foo
)
-- return our rasters aligned with our canvas
SELECT ST_AsRaster(m.geom, canvas.rast, '8BUI', 240) As rast, bnum, descrip
    FROM mygeoms AS m CROSS JOIN canvas
UNION ALL
SELECT canvas.rast, 4, 'canvas'
FROM canvas;

-- Map algebra on single band rasters and then collect with ST_AddBand
INSERT INTO map_shapes(rast,bnum,descrip)
SELECT ST_AddBand(ST_AddBand(rasts[1], rasts[2]),rasts[3]), 4, 'map bands overlay fct union ←
    (canvas)'
    FROM (SELECT ARRAY(SELECT ST_MapAlgebraFct(m1.rast, m2.rast,
        'raster_mapalgebra_union(double precision, double precision, integer[], text[]) ←
        '::regprocedure, '8BUI', 'FIRST')
        FROM map_shapes As m1 CROSS JOIN map_shapes As m2
        WHERE m1.descrip = 'canvas' AND m2.descrip <
    > 'canvas' ORDER BY m2.bnum) As rasts) As foo;

```



*map bands overlay (Canvas) (R: schmale Straße, G: Kreis, B: breite Straße)*

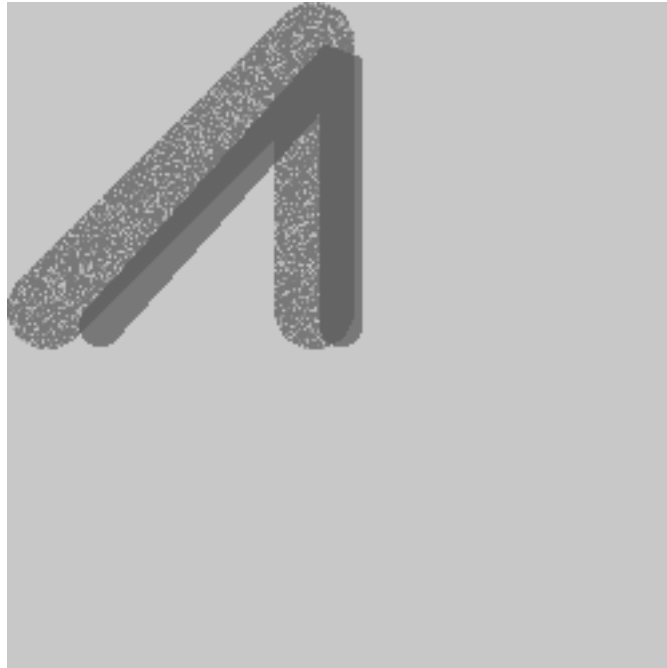
### Eine benutzerdefinierte Funktion, die zusätzliche Übergabewerte entgegennimmt

```
CREATE OR REPLACE FUNCTION raster_mapalgebra_userargs(
  rast1 double precision,
  rast2 double precision,
  pos integer[],
  VARIADIC userargs text[]
)
RETURNS double precision
AS $$
DECLARE
BEGIN
  CASE
    WHEN rast1 IS NOT NULL AND rast2 IS NOT NULL THEN
      RETURN least(userargs[1]::integer, (rast1 + rast2)/2.);
    WHEN rast1 IS NULL AND rast2 IS NULL THEN
      RETURN userargs[2]::integer;
    WHEN rast1 IS NULL THEN
      RETURN greatest(rast2, random()*userargs[3]::integer)::integer;
    ELSE
      RETURN greatest(rast1, random()*userargs[4]::integer)::integer;
  END CASE;

  RETURN NULL;
END;
$$ LANGUAGE 'plpgsql' VOLATILE COST 1000;

SELECT ST_MapAlgebraFct(m1.rast, 1, m1.rast, 3,
  'raster_mapalgebra_userargs(double precision, double precision, integer[], text ←
  [])'::regprocedure,
  '8BUI', 'INTERSECT', '100','200','200','0')
FROM map_shapes As m1
```

```
WHERE m1.descrip = 'map bands overlay fct union (canvas)';
```



*Eine benutzerdefinierte Funktion mit zusätzlichen Übergabewerten und unterschiedlichen Bändern des gleichen Raster*

#### Siehe auch

[ST\\_MapAlgebraExpr](#), [ST\\_BandPixelType](#), [ST\\_GeoReference](#), [ST\\_SetValue](#)

#### 11.12.11 ST\_MapAlgebraFctNgb

`ST_MapAlgebraFctNgb` — Version mit 1em Band: Map Algebra Nearest Neighbor mit einer benutzerdefinierten PostgreSQL Funktion. Gibt einen Raster zurück, dessen Werte sich aus einer benutzerdefinierte PL/pgsql Funktion ergeben, welche die Nachbarschaftswerte des Ausgangsrasterbandes einbezieht.

#### Synopsis

raster `ST_MapAlgebraFctNgb`(raster rast, integer band, text pixeltype, integer ngbwidth, integer ngbheight, regprocedure on-erastngbuserfunc, text nodatamode, text[] VARIADIC args);

#### Beschreibung



#### Warning

`ST_MapAlgebraFctNgb` Seit der Version 2.1.0 überholt. Benutzen Sie stattdessen bitte [ST\\_MapAlgebra \(callback function version\)](#)

(Version mit einem Raster) Gibt einen Raster zurück, dessen Werte sich aus einer benutzerdefinierte PL/pgSQL Funktion errechnen, welche die Nachbarschaftswerte des Ausgangsrasterbandes mit einbezieht. Die benutzerdefinierte Funktion nimmt die Werte der Nachbarschaftspixel als Zahlenfeld entgegen und gibt für jedes Pixel das Ergebnis der Funktion zurück, indem die aktuellen Werte der betrachteten Pixel mit dem Ergebnis der benutzerdefinierten Funktion ersetzt werden.

**rast** Raster der durch die benutzerdefinierte Funktion ausgewertet werden soll.

**band** Die Bandnummer des Raster, die ausgewertet werden soll. Die Standardeinstellung ist 1.

**pixeltype** Der resultierende Pixeltyp des Zielraster muss entweder aus **ST\_BandPixelType** sein, weggelassen oder auf NULL gesetzt werden. Wenn er nicht übergeben wird oder auf NULL gesetzt ist, wird er standardmäßig auf den Pixeltyp von **rast** gesetzt. Die Ergebnisse werden abgeschnitten, wenn sie größer als für den Pixeltyp erlaubt sind.

**ngbwidth** Die Breite der Nachbarschaft in Zellen.

**ngbheight** Die Höhe der Nachbarschaft in Zellen.

**onerastngbuserfunc** Eine benutzerdefinierte Funktion in PLPGSQL/psql, die auf die Nachbarschaftspixel in einem einzelnen Rasterband angewandt wird. Das erste Element ist ein 2-dimensionales Feld mit Zahlen, welche das Rechteck mit den Nachbarschaftspixel beschreiben

**nodatamode** Bestimmt welcher Wert an die Funktion übergeben werden soll, wenn ein Nachbarschaftspixel NODATA oder NULL ist

'ignore': NODATA Werte in der Nachbarschaft werden bei der Berechnung ignoriert -- diese Flag muss an die Rückruf-funktion gesendet werden und die benutzerdefinierte Funktion entscheidet dann, wie diese Werte ignoriert werden sollen.

'NULL': NODATA Werte in der Nachbarschaft bedingen, dass die resultierenden Pixel ebenfalls NULL annehmen - die benutzerdefinierte Rückruffunktion wird bei diesem Fall übersprungen.

'value': NODATA Werte in der Nachbarschaft werden durch den Wert des Referenzpixel (das Pixel im Zentrum der Nachbarschaft) ersetzt. Anmerkung: Wenn dieser Wert NODATA ist, dann ist die Verhaltensweise gleich wie bei 'NULL' (für die betroffene Nachbarschaft)

**args** Übergabewerte für die benutzerdefinierte Funktion.

Verfügbarkeit: 2.0.0

## Beispiele

Die Beispiele nutzen den Raster "Katrina", der als einzelne Kachel geladen wird, wie unter [http://trac.osgeo.org/gdal/wiki/frmts\\_wtkraster.html](http://trac.osgeo.org/gdal/wiki/frmts_wtkraster.html) beschrieben; und in den Beispielen von **ST\_Rescale** aufbereitet wurde.

```
--
-- A simple 'callback' user function that averages up all the values in a neighborhood.
--
CREATE OR REPLACE FUNCTION rast_avg(matrix float[][], nodatamode text, variadic args text ←
[])
RETURNS float AS
$$
DECLARE
    _matrix float[][];
    x1 integer;
    x2 integer;
    y1 integer;
    y2 integer;
    sum float;
BEGIN
    _matrix := matrix;
    sum := 0;
    FOR x in array_lower(matrix, 1)..array_upper(matrix, 1) LOOP
        FOR y in array_lower(matrix, 2)..array_upper(matrix, 2) LOOP
            sum := sum + _matrix[x][y];
        END LOOP;
    END LOOP;
    RETURN (sum*1.0/(array_upper(matrix,1)*array_upper(matrix,2) ))::integer ;
END;
$$
```

```
LANGUAGE 'plpgsql' IMMUTABLE COST 1000;

-- now we apply to our raster averaging pixels within 2 pixels of each other in X and Y ↔
-- direction --
SELECT ST_MapAlgebraFctNgb(rast, 1, '8BUI', 4,4,
    'rast_avg(float[][] , text, text[])'::regprocedure, 'NULL', NULL) As nn_with_border
FROM katrinas_rescaled
limit 1;
```



*Das erste Band Unseres Rasters*



*neuer Raster mit dem Durchschnittswert der Pixel  
innerhalb von 4x4 Pixel*

#### Siehe auch

[ST\\_MapAlgebraFct](#), [ST\\_MapAlgebraExpr](#), [ST\\_Rescale](#)

### 11.12.12 ST\_Reclass

**ST\_Reclass** — Erstellt einen neuen Raster, der aus neu klassifizierten Bändern des Originalraster besteht. Das Band "nband" ist jenes das verändert werden soll. Wenn "nband" nicht angegeben ist, wird "Band 1" angenommen. Alle anderen Bänder bleiben unverändert. Anwendungsfall: zwecks einfacherer Visualisierung ein 16BUI-Band in ein 8BUI-Band konvertieren und so weiter.

#### Synopsis

```
raster ST_Reclass(raster rast, integer nband, text reclassexpr, text pixeltype, double precision nodataval=NULL);
raster ST_Reclass(raster rast, reclassarg[] VARIADIC reclassargset);
raster ST_Reclass(raster rast, text reclassexpr, text pixeltype);
```

#### Beschreibung

Erstellt einen neuen Raster, indem eine gültige algebraische PostgreSQL Operation die durch den Ausdruck `reclassexpr` festgelegt wird auf den Ausgangsraster (`rast`) angewendet wird. Wenn `nband` nicht angegeben ist, wird Band 1 angenommen. Der Zielraster hat die selbe Georeferenzierung, Breite und Höhe wie der ursprüngliche Raster. Nicht ausgewiesene Bänder werden unverändert zurückgegeben. Siehe [reclassarg](#) für eine Beschreibung der gültigen Ausdrücke zur Neuklassifizierung.

Die Bänder des Zielraster haben den Pixeltyp `pixeltype`. Wenn `reclassargset` übergeben wird, dann bestimmt ein "regclassarg" wie das jeweilige Band erstellt werden soll.

Verfügbarkeit: 2.0.0



### Grundlegende Beispiele

Erzeugt einen neuen Raster aus dem Original, indem Band 2 von 8BUI auf 4BUI und alle Werte von 101-254 auf NODATA gesetzt werden.

```
ALTER TABLE dummy_rast ADD COLUMN reclass_rast raster;
UPDATE dummy_rast SET reclass_rast = ST_Reclass(rast,2,'0-87:1-10, 88-100:11-15, ←
    101-254:0-0', '4BUI',0) WHERE rid = 2;

SELECT i as col, j as row, ST_Value(rast,2,i,j) As origval,
    ST_Value(reclass_rast, 2, i, j) As reclassval,
    ST_Value(reclass_rast, 2, i, j, false) As reclassval_include_nodata
FROM dummy_rast CROSS JOIN generate_series(1, 3) AS i CROSS JOIN generate_series(1,3) AS j
WHERE rid = 2;
```

col	row	origval	reclassval	reclassval_include_nodata
1	1	78	9	9
2	1	98	14	14
3	1	122		0
1	2	96	14	14
2	2	118		0
3	2	180		0
1	3	99	15	15
2	3	112		0
3	3	169		0

### Beispiel: Erweiterte Verwendung mit mehreren "reclassargs"

Erstellt einen neuen Raster aus dem Ursprungsraster, wobei die Bänder 1, 2 und 3 in 1BB, 4BUI und 4BUI konvertiert und neu klassifiziert werden. Verwendet das variadische Argument reclassarg, welches eine unbegrenzte Anzahl von "reclassargs" entgegennehmen kann (theoretisch so viele wie Bänder vorhanden sind)

```
UPDATE dummy_rast SET reclass_rast =
    ST_Reclass(rast,
        ROW(2,'0-87]:1-10, (87-100]:11-15, (101-254]:0-0', '4BUI',NULL)::reclassarg,
        ROW(1,'0-253]:1, 254:0', '1BB', NULL)::reclassarg,
        ROW(3,'0-70]:1, (70-86:2, [86-150]:3, [150-255:4', '4BUI', NULL)::reclassarg
    ) WHERE rid = 2;

SELECT i as col, j as row, ST_Value(rast,1,i,j) As ov1, ST_Value(reclass_rast, 1, i, j) As ←
    rv1,
    ST_Value(rast,2,i,j) As ov2, ST_Value(reclass_rast, 2, i, j) As rv2,
    ST_Value(rast,3,i,j) As ov3, ST_Value(reclass_rast, 3, i, j) As rv3
FROM dummy_rast CROSS JOIN generate_series(1, 3) AS i CROSS JOIN generate_series(1,3) AS j
WHERE rid = 2;
```

col	row	ov1	rv1	ov2	rv2	ov3	rv3
1	1	253	1	78	9	70	1
2	1	254	0	98	14	86	3
3	1	253	1	122	0	100	3
1	2	253	1	96	14	80	2
2	2	254	0	118	0	108	3
3	2	254	0	180	0	162	4
1	3	250	1	99	15	90	3
2	3	254	0	112	0	108	3
3	3	254	0	169	0	175	4

### Beispiel: Abbildung eines Einzelbandrasters (32BF) auf mehrere darstellbare Bänder

Erstellung eines neuen Raster mit 3 Bändern (8BUI,8BUI,8BUI darstellbarer Raster) aus einem Raster mit nur einem 32bf-Band

```
ALTER TABLE wind ADD COLUMN rast_view raster;
UPDATE wind
  set rast_view = ST_AddBand( NULL,
    ARRAY[
      ST_Reclass(rast, 1, '0.1-10]:1-10,9-10]:11, (11-33:0'::text, '8BUI'::text,0),
      ST_Reclass(rast,1, '11-33):0-255, [0-32:0, (34-100:0'::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](#)

### 11.12.13 ST\_Union

**ST\_Union** — Gibt die Vereinigung mehrerer Rasterkacheln in einem einzelnen Raster mit mehreren Bändern zurück.

#### Synopsis

```
raster ST_Union(setof raster rast);
raster ST_Union(setof raster rast, unionarg[] unionargset);
raster ST_Union(setof raster rast, integer nband);
raster ST_Union(setof raster rast, text uniontype);
raster ST_Union(setof raster rast, integer nband, text uniontype);
```

#### Beschreibung

Gibt die Vereinigung von Rasterkacheln in einem einzelnen Raster zurück, der mindestens aus einem Band besteht. Die räumliche Ausdehnung des Zielraster entspricht der Gesamtausdehnung. Im Fall von Überschneidungen wird der resultierende Wert durch `uniontype` festgelegt, welcher folgende Werte annehmen kann: `LAST` (Standardwert), `FIRST`, `MIN`, `MAX`, `COUNT`, `SUM`, `MEAN`, `RANGE`.



#### Note

Damit Raster vereinigt werden können, müssen sie alle gleich ausgerichtet sein. Siehe [ST\\_SameAlignment](#) und [ST\\_NotSameAlignmentReason](#) für weitere Details und Hilfe. Eine Möglichkeit, um Probleme mit der Ausrichtung zu beheben ist [ST\\_Resample](#) und die Verwendung eines gemeinsamen Referenzraster zur Anpassung.

Verfügbarkeit: 2.0.0

Erweiterung: 2.1.0 Geschwindigkeit verbessert (zur Gänze C-basiert)

Verfügbarkeit: 2.1.0 `ST_Union(rast, unionarg)` Variante wurde eingeführt.

Erweiterung: 2.1.0 `ST_Union(rast)` (Variante 1) vereinigt alle Bänder aller Ausgangsraster. Vorherige Versionen von PostGIS setzten das erste Band voraus.

Erweiterung: 2.1.0 `ST_Union(rast, uniontype)` (Variante 4) vereinigt alle Bänder aller Ausgangsraster.

**Beispiele: Wiederherstellung eines einzelnen Bandes einer Rasterkachel**

```
-- this creates a single band from first band of raster tiles
-- that form the original file system tile
SELECT filename, ST_Union(rast,1) As file_rast
FROM sometable WHERE filename IN('dem01', 'dem02') GROUP BY filename;
```

**Beispiele: Vereinigt Kacheln, die eine Geometrie schneiden, zu einem Raster mit mehreren Bändern**

```
-- this creates a multi band raster collecting all the tiles that intersect a line
-- Note: In 2.0, this would have just returned a single band raster
-- , new union works on all bands by default
-- this is equivalent to unionarg: ARRAY[ROW(1, 'LAST'), ROW(2, 'LAST'), ROW(3, 'LAST')]:: ←
unionarg[]
SELECT ST_Union(rast)
FROM aerials.boston
WHERE ST_Intersects(rast, ST_GeomFromText('LINESTRING(230486 887771, 230500 88772)',26986) ←
);
```

**Beispiele: Vereinigt Kacheln, die eine Geometrie schneiden, zu einem Raster mit mehreren Bändern**

Um nur eine Teilmenge der Bänder zu erhalten, oder die Reihenfolge der Bänder zu ändern, können wir die längere Syntax verwenden

```
-- this creates a multi band raster collecting all the tiles that intersect a line
SELECT ST_Union(rast,ARRAY[ROW(2, 'LAST'), ROW(1, 'LAST'), ROW(3, 'LAST')]::unionarg[])
FROM aerials.boston
WHERE ST_Intersects(rast, ST_GeomFromText('LINESTRING(230486 887771, 230500 88772)',26986) ←
);
```

**Siehe auch**

[unionarg](#), [ST\\_Envelope](#), [ST\\_ConvexHull](#), [ST\\_Clip](#), [ST\\_Union](#)

## 11.13 Integrierte Map Algebra Callback Funktionen

### 11.13.1 ST\_Distinct4ma

**ST\_Distinct4ma** — Funktion zur Rasterdatenverarbeitung, welche die Anzahl der einzelnen Pixelwerte in der Nachbarschaft errechnet.

**Synopsis**

```
float8 ST_Distinct4ma(float8[][] matrix, text nodatamode, text[] VARIADIC args);
double precision ST_Distinct4ma(double precision[][][] value, integer[][] pos, text[] VARIADIC userargs);
```

## Beschreibung

Berechnet die Anzahl der einzelnen Pixelwerte in der Nachbarschaft von Pixel.



### Note

Variante 1 ist eine spezialisierte Rückruffunktion, die als Rückrufparameter für `ST_MapAlgebraFctNgb` verwendet werden kann.



### Note

Variante 2 ist eine spezialisierte Rückruffunktion, die als Rückrufparameter für `ST_MapAlgebra (callback function version)` verwendet werden kann.



### Warning

Von der Verwendung der Variante 1 wird abgeraten, da `ST_MapAlgebraFctNgb` seit 2.1.0 überholt ist.

Verfügbarkeit: 2.0.0

Erweiterung: 2.1.0 Variante 2 wurde hinzugefügt

## Beispiele

```
SELECT
  rid,
  st_value(
    st_mapalgebrafctngb(rast, 1, NULL, 1, 1, 'st_distinct4ma(float[],text,text[])':: regprocedure, 'ignore', NULL), 2, 2
  )
FROM dummy_rast
WHERE rid = 2;
  rid | st_value
-----+-----
    2 |      3
(1 row)
```

## Siehe auch

[ST\\_MapAlgebraFctNgb](#), [ST\\_MapAlgebra \(callback function version\)](#), [ST\\_Min4ma](#), [ST\\_Max4ma](#), [ST\\_Sum4ma](#), [ST\\_Mean4ma](#), [ST\\_Distinct4ma](#), [ST\\_StdDev4ma](#)

### 11.13.2 ST\_InvDistWeight4ma

`ST_InvDistWeight4ma` — Funktion zur Rasterdatenverarbeitung, die den Wert eines Pixel aus den Pixel der Nachbarschaft interpoliert.

## Synopsis

```
double precision ST_InvDistWeight4ma(double precision[][][] value, integer[][] pos, text[] VARIADIC userargs);
```

## Beschreibung

Interpoliert den Wert eines Pixel über die Methode der inversen Distanzwichtung.

Es gibt zwei optionale Parameter, die über `userargs` übergeben werden können. Der erste Parameter ist der Exponent (die Variable "k" in unterer Gleichung); dieser liegt zwischen 0 und 1 und wird in der Gleichung für die Inverse Distanzwichtung verwendet. Wenn er nicht angegeben ist, wird standardmäßig 1 angenommen. Der zweite Parameter entspricht der Gewichtung in Prozent und wird nur verwendet, wenn der Wert der betrachteten Pixel einem aus der Nachbarschaft interpolierten Wert entspricht. Wenn er nicht angegeben ist und das betrachtete Pixel einen Wert hat, so wird dieser Wert zurückgegeben.

Die grundlegende Gleichung der inversen Distanzwichtung ist:

$$\hat{z}(x_o) = \frac{\sum_{j=1}^m z(x_j) d_{ij}^{-k}}{\sum_{j=1}^m d_{ij}^{-k}}$$

*k = Exponent, eine reelle Zahl zwischen 0 und 1*



### Note

Diese Funktion ist eine spezialisierte Rückruffunktion, die als Rückrufparameter für [ST\\_MapAlgebra \(callback function version\)](#) verwendet werden kann.

Verfügbarkeit: 2.1.0

## Beispiele

```
-- NEEDS EXAMPLE
```

## Siehe auch

[ST\\_MapAlgebra \(callback function version\)](#), [ST\\_MinDist4ma](#)

### 11.13.3 ST\_Max4ma

`ST_Max4ma` — Funktion zur Rasterdatenverarbeitung, die den maximalen Zellwert in der Nachbarschaft eines Pixel errechnet.

## Synopsis

```
float8 ST_Max4ma(float8[][] matrix, text nodatamode, text[] VARIADIC args);
double precision ST_Max4ma(double precision[][][] value, integer[][] pos, text[] VARIADIC userargs);
```

## Beschreibung

Berechnet den maximalen Zellwert in der Nachbarschaft von Pixel.

Bei Variante 2 kann ein Substitutionswert für NODATA an "userargs" übergeben werden.



### Note

Variante 1 ist eine spezialisierte Rückruffunktion, die als Rückrufparameter für [ST\\_MapAlgebraFctNgb](#) verwendet werden kann.

**Note**

Variante 2 ist eine spezialisierte Rückruffunktion, die als Rückrufparameter für [ST\\_MapAlgebra \(callback function version\)](#) verwendet werden kann.

**Warning**

Von der Verwendung der Variante 1 wird abgeraten, da [ST\\_MapAlgebraFctNgb](#) seit 2.1.0 überholt ist.

Verfügbarkeit: 2.0.0

Erweiterung: 2.1.0 Variante 2 wurde hinzugefügt

**Beispiele**

```
SELECT
  rid,
  st_value(
    st_mapalgebrafctngb(rast, 1, NULL, 1, 1, 'st_max4ma(float[][],text,text[])':: ↵
    regprocedure, 'ignore', NULL), 2, 2
  )
FROM dummy_rast
WHERE rid = 2;
  rid | st_value
-----+-----
    2 |      254
(1 row)
```

**Siehe auch**

[ST\\_MapAlgebraFctNgb](#), [ST\\_MapAlgebra \(callback function version\)](#), [ST\\_Min4ma](#), [ST\\_Sum4ma](#), [ST\\_Mean4ma](#), [ST\\_Range4ma](#), [ST\\_Distinct4ma](#), [ST\\_StdDev4ma](#)

**11.13.4 ST\_Mean4ma**

[ST\\_Mean4ma](#) — Funktion zur Rasterdatenverarbeitung, die den mittleren Zellwert in der Nachbarschaft von Pixel errechnet.

**Synopsis**

```
float8 ST\_Mean4ma(float8[][] matrix, text nodatamode, text[] VARIADIC args);
double precision ST\_Mean4ma(double precision[][][] value, integer[][] pos, text[] VARIADIC userargs);
```

**Beschreibung**

Berechnet den mittleren Zellwert in der Nachbarschaft von Pixel.

Bei Variante 2 kann ein Substitutionswert für NODATA an "userargs" übergeben werden.

**Note**

Variante 1 ist eine spezialisierte Rückruffunktion, die als Rückrufparameter für [ST\\_MapAlgebraFctNgb](#) verwendet werden kann.

**Note**

Variante 2 ist eine spezialisierte Rückruffunktion, die als Rückrufparameter für `ST_MapAlgebra` (callback function version) verwendet werden kann.

**Warning**

Von der Verwendung der Variante 1 wird abgeraten, da `ST_MapAlgebraFctNgb` seit 2.1.0 überholt ist.

Verfügbarkeit: 2.0.0

Erweiterung: 2.1.0 Variante 2 wurde hinzugefügt

**Beispiele: Variante 1**

```
SELECT
  rid,
  st_value(
    st_mapalgebrafctngb(rast, 1, '32BF', 1, 1, 'st_mean4ma(float[][],text,text[])':: ↵
      regprocedure, 'ignore', NULL), 2, 2
  )
FROM dummy_rast
WHERE rid = 2;
  rid |      st_value
-----+-----
    2 | 253.222229003906
(1 row)
```

**Beispiele: Variante 2**

```
SELECT
  rid,
  st_value(
    ST_MapAlgebra(rast, 1, 'st_mean4ma(double precision[][][], integer[][], text ↵
      [])'::regprocedure,'32BF', 'FIRST', NULL, 1, 1)
    , 2, 2)
FROM dummy_rast
WHERE rid = 2;
  rid |      st_value
-----+-----
    2 | 253.222229003906
(1 row)
```

**Siehe auch**

[ST\\_MapAlgebraFctNgb](#), [ST\\_MapAlgebra \(callback function version\)](#), [ST\\_Min4ma](#), [ST\\_Max4ma](#), [ST\\_Sum4ma](#), [ST\\_Range4ma](#), [ST\\_StdDev4ma](#)

**11.13.5 ST\_Min4ma**

`ST_Min4ma` — Funktion zur Rasterdatenverarbeitung, die den minimalen Zellwert in der Nachbarschaft von Pixel errechnet.

## Synopsis

float8 **ST\_Min4ma**(float8[][] matrix, text nodatamode, text[] VARIADIC args);

double precision **ST\_Min4ma**(double precision[][][] value, integer[][] pos, text[] VARIADIC userargs);

## Beschreibung

Berechnet den minimalen Zellwert in der Nachbarschaft von Pixel.

Bei Variante 2 kann ein Substitutionswert für NODATA an "userargs" übergeben werden.



### Note

Variante 1 ist eine spezialisierte Rückruffunktion, die als Rückrufparameter für [ST\\_MapAlgebraFctNgb](#) verwendet werden kann.



### Note

Variante 2 ist eine spezialisierte Rückruffunktion, die als Rückrufparameter für [ST\\_MapAlgebra \(callback function version\)](#) verwendet werden kann.



### Warning

Von der Verwendung der Variante 1 wird abgeraten, da [ST\\_MapAlgebraFctNgb](#) seit 2.1.0 überholt ist.

Verfügbarkeit: 2.0.0

Erweiterung: 2.1.0 Variante 2 wurde hinzugefügt

## Beispiele

```
SELECT
  rid,
  st_value(
    st_mapalgebrafctngb(rast, 1, NULL, 1, 1, 'st_min4ma(float [][] ,text,text[])':: ↵
      regprocedure, 'ignore', NULL), 2, 2
  )
FROM dummy_rast
WHERE rid = 2;
  rid | st_value
-----+-----
    2 |      250
(1 row)
```

## Siehe auch

[ST\\_MapAlgebraFctNgb](#), [ST\\_MapAlgebra \(callback function version\)](#), [ST\\_Max4ma](#), [ST\\_Sum4ma](#), [ST\\_Mean4ma](#), [ST\\_Range4ma](#), [ST\\_Distinct4ma](#), [ST\\_StdDev4ma](#)

### 11.13.6 ST\_MinDist4ma

**ST\_MinDist4ma** — Funktion zur Rasterdatenverarbeitung, welche die kürzeste Entfernung (in Pixel) zwischen dem Pixel von Interesse und einem benachbarten Pixel mit Zellwert zurückgibt.



## Synopsis

double precision **ST\_MinDist4ma**(double precision[][][] value, integer[][] pos, text[] VARIADIC userargs);

## Beschreibung

Gibt die kürzeste Entfernung (Pixelanzahl) zwischen dem Pixel von Interesse und dem nächstgelegenen Pixel mit Zellwert in der Nachbarschaft zurück.



### Note

Diese Funktion soll einen Anhaltspunkt liefern, um die Sinnhaftigkeit des mittels **ST\_InvDistWeight4ma** interpolierten Wertes für das betrachtete Pixel abzuleiten. Diese Funktion ist besonders nützlich wenn die Nachbarschaft des Pixel nur spärlich besetzt ist.



### Note

Diese Funktion ist eine spezialisierte Rückruffunktion, die als Rückrufparameter für **ST\_MapAlgebra (callback function version)** verwendet werden kann.

Verfügbarkeit: 2.1.0

## Beispiele

```
-- NEEDS EXAMPLE
```

## Siehe auch

**ST\_MapAlgebra (callback function version)**, **ST\_InvDistWeight4ma**

## 11.13.7 ST\_Range4ma

**ST\_Range4ma** — Funktion zur Rasterdatenverarbeitung, die den Wertebereich der Pixel in einer Nachbarschaft errechnet.

## Synopsis

float8 **ST\_Range4ma**(float8[][] matrix, text nodatamode, text[] VARIADIC args);

double precision **ST\_Range4ma**(double precision[][][] value, integer[][] pos, text[] VARIADIC userargs);

## Beschreibung

Berechnet den Wertebereich der Pixel in einer Nachbarschaft von Pixel.

Bei Variante 2 kann ein Substitutionswert für NODATA an "userargs" übergeben werden.



### Note

Variante 1 ist eine spezialisierte Rückruffunktion, die als Rückrufparameter für **ST\_MapAlgebraFctNgb** verwendet werden kann.

**Note**

Variante 2 ist eine spezialisierte Rückruffunktion, die als Rückrufparameter für [ST\\_MapAlgebra \(callback function version\)](#) verwendet werden kann.

**Warning**

Von der Verwendung der Variante 1 wird abgeraten, da [ST\\_MapAlgebraFctNgb](#) seit 2.1.0 überholt ist.

Verfügbarkeit: 2.0.0

Erweiterung: 2.1.0 Variante 2 wurde hinzugefügt

**Beispiele**

```
SELECT
  rid,
  st_value(
    st_mapalgebrafctngb(rast, 1, NULL, 1, 1, 'st_range4ma(float[][] ,text,text[])':: ↵
      regprocedure, 'ignore', NULL), 2, 2
  )
FROM dummy_rast
WHERE rid = 2;
  rid | st_value
-----+-----
    2 |      4
(1 row)
```

**Siehe auch**

[ST\\_MapAlgebraFctNgb](#), [ST\\_MapAlgebra \(callback function version\)](#), [ST\\_Min4ma](#), [ST\\_Max4ma](#), [ST\\_Sum4ma](#), [ST\\_Mean4ma](#), [ST\\_Distinct4ma](#), [ST\\_StdDev4ma](#)

**11.13.8 ST\_StdDev4ma**

[ST\\_StdDev4ma](#) — Funktion zur Rasterdatenverarbeitung, welche die Standardabweichung der Zellwerte in der Nachbarschaft von Pixel errechnet.

**Synopsis**

float8 [ST\\_StdDev4ma](#)(float8[][] matrix, text nodatamode, text[] VARIADIC args);  
double precision [ST\\_StdDev4ma](#)(double precision[][][] value, integer[][] pos, text[] VARIADIC userargs);

**Beschreibung**

Berechnet die Standardabweichung der Zellwerte in der Nachbarschaft von Pixel.

**Note**

Variante 1 ist eine spezialisierte Rückruffunktion, die als Rückrufparameter für [ST\\_MapAlgebraFctNgb](#) verwendet werden kann.

**Note**

Variante 2 ist eine spezialisierte Rückruffunktion, die als Rückrufparameter für [ST\\_MapAlgebra \(callback function version\)](#) verwendet werden kann.

**Warning**

Von der Verwendung der Variante 1 wird abgeraten, da [ST\\_MapAlgebraFctNgb](#) seit 2.1.0 überholt ist.

Verfügbarkeit: 2.0.0

Erweiterung: 2.1.0 Variante 2 wurde hinzugefügt

**Beispiele**

```
SELECT
  rid,
  st_value(
    st_mapalgebrafctngb(rast, 1, '32BF', 1, 1, 'st_stddev4ma(float[][],text,text[])':: ←
      regprocedure, 'ignore', NULL), 2, 2
  )
FROM dummy_rast
WHERE rid = 2;
  rid |      st_value
-----+-----
    2 | 1.30170822143555
(1 row)
```

**Siehe auch**

[ST\\_MapAlgebraFctNgb](#), [ST\\_MapAlgebra \(callback function version\)](#), [ST\\_Min4ma](#), [ST\\_Max4ma](#), [ST\\_Sum4ma](#), [ST\\_Mean4ma](#), [ST\\_Distinct4ma](#), [ST\\_StdDev4ma](#)

**11.13.9 ST\_Sum4ma**

[ST\\_Sum4ma](#) — Funktion zur Rasterdatenverarbeitung, die die Summe aller Zellwerte in der Nachbarschaft von Pixel errechnet.

**Synopsis**

```
float8 ST_Sum4ma(float8[][] matrix, text nodatamode, text[] VARIADIC args);
double precision ST_Sum4ma(double precision[][][] value, integer[][] pos, text[] VARIADIC userargs);
```

**Beschreibung**

Berechnet die Summe aller Zellwerte in der Nachbarschaft von Pixel.

Bei Variante 2 kann ein Substitutionswert für NODATA an "userargs" übergeben werden.

**Note**

Variante 1 ist eine spezialisierte Rückruffunktion, die als Rückrufparameter für [ST\\_MapAlgebraFctNgb](#) verwendet werden kann.

**Note**

Variante 2 ist eine spezialisierte Rückruffunktion, die als Rückrufparameter für [ST\\_MapAlgebra \(callback function version\)](#) verwendet werden kann.

**Warning**

Von der Verwendung der Variante 1 wird abgeraten, da [ST\\_MapAlgebraFctNgb](#) seit 2.1.0 überholt ist.

Verfügbarkeit: 2.0.0

Erweiterung: 2.1.0 Variante 2 wurde hinzugefügt

**Beispiele**

```
SELECT
  rid,
  st_value(
    st_mapalgebrafctngb(rast, 1, '32BF', 1, 1, 'st_sum4ma(float[][],text,text[])'::↔
      regprocedure, 'ignore', NULL), 2, 2
  )
FROM dummy_rast
WHERE rid = 2;
  rid | st_value
-----+-----
    2 |    2279
(1 row)
```

**Siehe auch**

[ST\\_MapAlgebraFctNgb](#), [ST\\_MapAlgebra \(callback function version\)](#), [ST\\_Min4ma](#), [ST\\_Max4ma](#), [ST\\_Mean4ma](#), [ST\\_Range4ma](#), [ST\\_Distinct4ma](#), [ST\\_StdDev4ma](#)

## 11.14 Rasterverarbeitung: DEM (Elevation)

### 11.14.1 ST\_Aspect

**ST\_Aspect** — Gibt die Exposition (standardmäßig in Grad) eines Rasterbandes mit Höhen aus. Nützlich für Terrain-Analysen.

**Synopsis**

```
raster ST_Aspect(raster rast, integer band=1, text pixeltype=32BF, text units=DEGREES, boolean interpolate_nodata=FALSE);
raster ST_Aspect(raster rast, integer band, raster customextent, text pixeltype=32BF, text units=DEGREES, boolean interpolate_nodata=FALSE);
```

## Beschreibung

Gibt die Exposition (standardmäßig in Grad) eines Höhenrasterbandes zurück. Verwendet Map Algebra und wendet die Expositionsgleichung auf benachbarte Pixel an.

`units` die Einheiten für die Exposition. Mögliche Werte sind: RADIANS, DEGREES (Standardeinstellung).

Wenn `units = RADIANS`, dann liegen die Werte zwischen 0 und  $2 * \pi$  Radiant und werden im Uhrzeigersinn von Norden her angegeben.

Wenn `units = DEGREES`, dann liegen die Werte zwischen 0 und 360 Grad und sind im Uhrzeigersinn von Norden her angegeben.

Wenn die Neigung einer Zelle null ist, dann ist die Exposition des Pixel -1.



### Note

Für weitere Information über Neigung, Exposition und Schummerung siehe [ESRI - How hillshade works](#) und [ERDAS Field Guide - Aspect Images](#).

Verfügbarkeit: 2.0.0

Erweiterung: 2.1.0 Verwendet `ST_MapAlgebra()` und der optionale Funktionsparameter `interpolate_nodata` wurde hinzugefügt

Änderung: 2.1.0 In Vorgängerversionen wurden die Werte in Radiant ausgegeben. Nun werden die Werte standardmäßig in Grad ausgegeben.

## Beispiele: Variante 1

```
WITH foo AS (
  SELECT ST_SetValues(
    ST_AddBand(ST_MakeEmptyRaster(5, 5, 0, 0, 1, -1, 0, 0, 0), 1, '32BF', 0, -9999),
    1, 1, 1, ARRAY[
      [1, 1, 1, 1, 1],
      [1, 2, 2, 2, 1],
      [1, 2, 3, 2, 1],
      [1, 2, 2, 2, 1],
      [1, 1, 1, 1, 1]
    ]::double precision[]
  ) AS rast
)
SELECT
  ST_DumpValues(ST_Aspect(rast, 1, '32BF'))
FROM foo
```

---

```
(1, "{ {315,341.565063476562,0,18.4349479675293,45}, {288.434936523438,315,0,45,71.5650482177734}, {270.434936523438,315,0,45,71.5650482177734}, {227,180,161.565048217773,135}}")
(1 row)
```

## Beispiele: Variante 2

Ein vollständiges Beispiel mit Coveragekacheln. Diese Abfrage funktioniert nur mit PostgreSQL 9.1 oder höher.

```
WITH foo AS (
  SELECT ST_Tile(
    ST_SetValues(
      ST_AddBand(
        ST_MakeEmptyRaster(6, 6, 0, 0, 1, -1, 0, 0, 0),
        1, '32BF', 0, -9999
      ),
      1, 1, 1, ARRAY[
        [1, 1, 1, 1, 1, 1],
        [1, 1, 1, 1, 2, 1],
        [1, 2, 2, 3, 3, 1],
        [1, 1, 3, 2, 1, 1],
        [1, 2, 2, 1, 2, 1],
        [1, 1, 1, 1, 1, 1]
      ]::double precision[]
    ),
    2, 2
  ) AS rast
)
SELECT
  t1.rast,
  ST_Aspect(ST_Union(t2.rast), 1, t1.rast)
FROM foo t1
CROSS JOIN foo t2
WHERE ST_Intersects(t1.rast, t2.rast)
GROUP BY t1.rast;
```

### Siehe auch

[ST\\_MapAlgebra \(callback function version\)](#), [ST\\_TRI](#), [ST\\_TPI](#), [ST\\_Roughness](#), [ST\\_HillShade](#), [ST\\_Slope](#)

## 11.14.2 ST\_HillShade

**ST\_HillShade** — Gibt für gegebenen Horizontalwinkel, Höhenwinkel, Helligkeit und Maßstabsverhältnis die hypothetische Beleuchtung eines Höhenrasterbandes zurück.

### Synopsis

```
raster ST_HillShade(raster rast, integer band=1, text pixeltype=32BF, double precision azimuth=315, double precision altitude=45, double precision max_bright=255, double precision scale=1.0, boolean interpolate_nodata=FALSE);
raster ST_HillShade(raster rast, integer band, raster customextent, text pixeltype=32BF, double precision azimuth=315, double precision altitude=45, double precision max_bright=255, double precision scale=1.0, boolean interpolate_nodata=FALSE);
```

### Beschreibung

Gibt für gegebenen Horizontalwinkel, Höhenwinkel, Helligkeit und Maßstabsverhältnis die hypothetische Beleuchtung eines Höhenrasterbandes zurück. Verwendet Map Algebra und wendet die Schummerungsgleichung auf die Nachbapixel an. Die zurückgegebenen Pixelwerte liegen zwischen 0 und 255.

`azimuth` der Richtungswinkel zwischen 0 und 360 Grad, im Uhrzeigersinn von Norden gemessen.

`altitude` der Höhenwinkel zwischen 0 und 90 Grad, wobei 0 Grad am Horizont und 90 Grad direkt über Kopf liegt.

`max_bright` ein Wert zwischen 0 und 255, wobei 0 keine Helligkeit und 255 maximale Helligkeit bedeutet.

`scale` ist das Verhältnis zwischen vertikalen und horizontalen Einheiten. Für Feet:LatLon verwenden Sie bitte `scale=370400`, für Meter:LatLon `scale=111120`.

Wenn `interpolate_nodata` `TRUE` ist, dann werden die NODATA Pixel des Ausgangsraster mit `ST_InvDistWeight4ma` interpoliert, bevor die Schummerung berechnet wird.



#### Note

Für weitere Information zur Schummerung siehe [How hillshade works](#).

Verfügbarkeit: 2.0.0

Erweiterung: 2.1.0 Verwendet `ST_MapAlgebra()` und der optionale Funktionsparameter `interpolate_nodata` wurde hinzugefügt

Änderung: 2.1.0 In Vorgängerversionen wurden Richtungswinkel und Höhenwinkel in Radiant angegeben. Nun werden die Werte in Grad ausgedrückt.

#### Beispiele: Variante 1

```
WITH foo AS (
  SELECT ST_SetValues(
    ST_AddBand(ST_MakeEmptyRaster(5, 5, 0, 0, 1, -1, 0, 0, 0), 1, '32BF', 0, -9999),
    1, 1, 1, ARRAY[
      [1, 1, 1, 1, 1],
      [1, 2, 2, 2, 1],
      [1, 2, 3, 2, 1],
      [1, 2, 2, 2, 1],
      [1, 1, 1, 1, 1]
    ]::double precision[][])
  ) AS rast
)
SELECT
  ST_DumpValues(ST_Hillshade(rast, 1, '32BF'))
FROM foo
```

```
-----
(1, "{NULL,NULL,NULL,NULL,NULL},{NULL,251.32763671875,220.749786376953,147.224319458008, ←
  NULL},{NULL,220.749786376953,180.312225341797,67.7497863769531,NULL},{NULL ←
  ,147.224319458008
,67.7497863769531,43.1210060119629,NULL},{NULL,NULL,NULL,NULL,NULL}")
(1 row)
```

#### Beispiele: Variante 2

Ein vollständiges Beispiel mit Coveragekacheln. Diese Abfrage funktioniert nur mit PostgreSQL 9.1 oder höher.

```
WITH foo AS (
  SELECT ST_Tile(
    ST_SetValues(
      ST_AddBand(
        ST_MakeEmptyRaster(6, 6, 0, 0, 1, -1, 0, 0, 0),
```

```

        1, '32BF', 0, -9999
    ),
    1, 1, 1, ARRAY[
        [1, 1, 1, 1, 1, 1],
        [1, 1, 1, 1, 2, 1],
        [1, 2, 2, 3, 3, 1],
        [1, 1, 3, 2, 1, 1],
        [1, 2, 2, 1, 2, 1],
        [1, 1, 1, 1, 1, 1]
    ]::double precision[]
    ),
    2, 2
) AS rast
)
SELECT
    t1.rast,
    ST_Hillshade(ST_Union(t2.rast), 1, t1.rast)
FROM foo t1
CROSS JOIN foo t2
WHERE ST_Intersects(t1.rast, t2.rast)
GROUP BY t1.rast;

```

**Siehe auch**

[ST\\_MapAlgebra \(callback function version\)](#), [ST\\_TRI](#), [ST\\_TPI](#), [ST\\_Roughness](#), [ST\\_Aspect](#), [ST\\_Slope](#)

**11.14.3 ST\_Roughness**

`ST_Roughness` — Gibt einen Raster mit der berechneten "Rauhigkeit" des DHM zurück.

**Synopsis**

```
raster ST_Roughness(raster rast, integer nband, raster customextent, text pixeltype="32BF" , boolean interpolate_nodata=FALSE);
```

**Beschreibung**

Berechnet die "Rauhigkeit" eines DHM, indem für ein bestimmtes Gebiet das Maximum vom Minimum abgezogen wird..

Verfügbarkeit: 2.1.0

**Beispiele**

```
-- needs examples
```

**Siehe auch**

[ST\\_MapAlgebra \(callback function version\)](#), [ST\\_TRI](#), [ST\\_TPI](#), [ST\\_Slope](#), [ST\\_HillShade](#), [ST\\_Aspect](#)

**11.14.4 ST\_Slope**

`ST_Slope` — Gibt die Neigung (standardmäßig in Grad) eines Höhenrasterbandes zurück. Nützlich für Terrain-Analysen.



## Synopsis

raster **ST\_Slope**(raster rast, integer nband=1, text pixeltype=32BF, text units=DEGREES, double precision scale=1.0, boolean interpolate\_nodata=FALSE);

raster **ST\_Slope**(raster rast, integer nband, raster customextent, text pixeltype=32BF, text units=DEGREES, double precision scale=1.0, boolean interpolate\_nodata=FALSE);

## Beschreibung

Gibt die Neigung (standardmäßig in Grad) eines Höhenrasterbandes zurück. Verwendet Map Algebra und wendet die Neigungsgleichung auf benachbarte Pixel an.

`units` die Einheiten für die Neigung. Mögliche Werte sind: RADIANS, DEGREES (Standardeinstellung) und Prozent.

`scale` ist das Verhältnis zwischen vertikalen und horizontalen Einheiten. Für Feet:LatLon verwenden Sie bitte `scale=370400`, für Meter:LatLon `scale=111120`.

Wenn `interpolate_nodata` TRUE ist, dann werden die NODATA Pixel des Ausgangsraster mit **ST\_InvDistWeight4ma** interpoliert, bevor die Neigung der Oberfläche berechnet wird.



### Note

Für weitere Information über Neigung, Exposition und Schummerung siehe [ESRI - How hillshade works](#) und [ERDAS Field Guide - Slope Images](#).

Verfügbarkeit: 2.0.0

Erweiterung: 2.1.0 Verwendet `ST_MapAlgebra()` und es wurden die optionalen Funktionsparameter `units`, `scale` und `interpolate_nodata` hinzugefügt

Änderung: 2.1.0 In Vorgängerversionen wurden die Werte in Radiant ausgegeben. Nun werden die Werte standardmäßig in Grad ausgegeben.

## Beispiele: Variante 1

```
WITH foo AS (
  SELECT ST_SetValues(
    ST_AddBand(ST_MakeEmptyRaster(5, 5, 0, 0, 1, -1, 0, 0, 0), 1, '32BF', 0, -9999),
    1, 1, 1, ARRAY[
      [1, 1, 1, 1, 1],
      [1, 2, 2, 2, 1],
      [1, 2, 3, 2, 1],
      [1, 2, 2, 2, 1],
      [1, 1, 1, 1, 1]
    ]::double precision[][])
  ) AS rast
)
SELECT
  ST_DumpValues(ST_Slope(rast, 1, '32BF'))
FROM foo

          st_dumpvalues
```

```
(1, "{10.0249881744385,21.5681285858154,26.5650520324707,21.5681285858154,10.0249881744385},{21.5681285858154,26.5650520324707,36.8698959350586,0,36.8698959350586,26.5650520324707},{21.5681285858154,35.26438905681285858154,26.5650520324707,21.5681285858154,10.0249881744385}}")
(1 row)
```

## Beispiele: Variante 2

Ein vollständiges Beispiel mit Coveragekacheln. Diese Abfrage funktioniert nur mit PostgreSQL 9.1 oder höher.

```
WITH foo AS (
  SELECT ST_Tile(
    ST_SetValues(
      ST_AddBand(
        ST_MakeEmptyRaster(6, 6, 0, 0, 1, -1, 0, 0, 0),
        1, '32BF', 0, -9999
      ),
      1, 1, 1, ARRAY[
        [1, 1, 1, 1, 1, 1],
        [1, 1, 1, 1, 2, 1],
        [1, 2, 2, 3, 3, 1],
        [1, 1, 3, 2, 1, 1],
        [1, 2, 2, 1, 2, 1],
        [1, 1, 1, 1, 1, 1]
      ]::double precision[]
    ),
    2, 2
  ) AS rast
)
SELECT
  t1.rast,
  ST_Slope(ST_Union(t2.rast), 1, t1.rast)
FROM foo t1
CROSS JOIN foo t2
WHERE ST_Intersects(t1.rast, t2.rast)
GROUP BY t1.rast;
```

## Siehe auch

[ST\\_MapAlgebra \(callback function version\)](#), [ST\\_TRI](#), [ST\\_TPI](#), [ST\\_Roughness](#), [ST\\_HillShade](#), [ST\\_Aspect](#)

### 11.14.5 ST\_TPI

**ST\_TPI** — Berechnet den "Topographic Position Index" eines Raster.

#### Synopsis

raster **ST\_TPI**(raster rast, integer nband, raster customextent, text pixeltype="32BF" , boolean interpolate\_nodata=FALSE );

#### Beschreibung

Berechnet den Topographischen Lageindex, welcher als fokaler Mittelwert mit dem Radius eins, abzüglich der mittigen Zelle, definiert ist.

**Note**

Diese Funktion unterstützt lediglich einen "focalmean"-Radius von Eins.

Verfügbarkeit: 2.1.0

**Beispiele**

```
-- needs examples
```

**Siehe auch**

[ST\\_MapAlgebra \(callback function version\)](#), [ST\\_TRI](#), [ST\\_Roughness](#), [ST\\_Slope](#), [ST\\_HillShade](#), [ST\\_Aspect](#)

### 11.14.6 ST\_TRI

ST\_TRI — Gibt einen Raster mit errechneten Geländerauheitsindex aus.

**Synopsis**

```
raster ST_TRI(raster rast, integer nband, raster customextent, text pixeltype="32BF" , boolean interpolate_nodata=FALSE );
```

**Beschreibung**

Der Geländerauheitsindex wird durch den Vergleich eines zentralen Pixel mit seinen Nachbarn berechnet. Dabei werden die Differenzen der absoluten Werte (Beträge) genommen und für das Ergebnis gemittelt.

**Note**

Diese Funktion unterstützt lediglich einen "focalmean"-Radius von Eins.

Verfügbarkeit: 2.1.0

**Beispiele**

```
-- needs examples
```

**Siehe auch**

[ST\\_MapAlgebra \(callback function version\)](#), [ST\\_Roughness](#), [ST\\_TPI](#), [ST\\_Slope](#), [ST\\_HillShade](#), [ST\\_Aspect](#)

## 11.15 Rasterverarbeitung: Raster zu Geometrie

### 11.15.1 Box3D

Box3D — Stellt das umschreibende Rechteck eines Raster als Box3D dar.

## Synopsis

box3d **Box3D**(raster rast);

## Beschreibung

Gibt ein Rechteck mit der Ausdehnung des Raster zurück.

Das Polygon ist durch die Eckpunkte des umschreibenden Rechtecks ((MINX, MINY), (MAXX, MAXY)) bestimmt

Änderung: 2.0.0 In Versionen vor 2.0 war dies üblicherweise Box2D anstelle von Box3D. Da Box2D überholt ist, wurde dies zu Box3D geändert.

## Beispiele

```
SELECT
  rid,
  Box3D(rast) AS rastbox
FROM dummy_rast;
```

rid	rastbox
1	BOX3D(0.5 0.5 0,20.5 60.5 0)
2	BOX3D(3427927.75 5793243.5 0,3427928 5793244 0)

## Siehe auch

[ST\\_Envelope](#)

### 11.15.2 ST\_ConvexHull

ST\_ConvexHull — Gibt die Geometrie der konvexen Hülle des Raster, inklusive der Pixel deren Werte gleich BandNoDataValue sind. Bei regelmäßig geformten und nicht rotierten Raster ist das Ergebnis ident mit ST\_Envelope. Diese Funktion ist deshalb nur bei unregelmäßig geformten oder rotierten Raster nützlich.

## Synopsis

geometry **ST\_ConvexHull**(raster rast);

## Beschreibung

Gibt die Geometrie der konvexen Hülle des Raster, inklusive der Pixel NoDataBandValue des Bandes. Bei regelmäßig geformten und nicht rotierten Raster ist das Ergebnis mehr oder weniger das von ST\_Envelope. Diese Funktion ist deshalb nur bei unregelmäßig geformten oder rotierten Raster nützlich.



### Note

ST\_Envelope rundet die Koordinaten und fügt so einen kleinen Puffer um den Raster hinzu. Dadurch unterscheidet sich das Ergebnis gering von ST\_ConvexHull, das nicht rundet.

## Beispiele

Siehe [PostGIS Raster Specification](#) für eine entsprechende Darstellung.

```
-- Note envelope and convexhull are more or less the same
SELECT ST_AsText(ST_ConvexHull(rast)) As convhull,
       ST_AsText(ST_Envelope(rast)) As env
FROM dummy_rast WHERE rid=1;
```

```

convhull | env
-----+-----
POLYGON((0.5 0.5,20.5 0.5,20.5 60.5,0.5 60.5,0.5 0.5)) | POLYGON((0 0,20 0,20 60,0 60,0 0)
)

```

```
-- now we skew the raster
-- note how the convex hull and envelope are now different
SELECT ST_AsText(ST_ConvexHull(rast)) As convhull,
       ST_AsText(ST_Envelope(rast)) As env
FROM (SELECT ST_SetRotation(rast, 0.1, 0.1) As rast
      FROM dummy_rast WHERE rid=1) As foo;
```

```

convhull | env
-----+-----
POLYGON((0.5 0.5,20.5 1.5,22.5 61.5,2.5 60.5,0.5 0.5)) | POLYGON((0 0,22 0,22 61,0 61,0 0)
)

```

## Siehe auch

[ST\\_Envelope](#), [ST\\_MinConvexHull](#), [ST\\_ConvexHull](#), [ST\\_AsText](#)

### 11.15.3 ST\_DumpAsPolygons

`ST_DumpAsPolygons` — Gibt `geomval` (`geom, val`) Zeilen eines Rasterbandes zurück. Wenn kein Band angegeben ist, wird die Bandnummer standardmäßig auf 1 gesetzt.

#### Synopsis

```
setof geomval ST_DumpAsPolygons(raster rast, integer band_num=1, boolean exclude_nodata_value=TRUE);
```

#### Beschreibung

Dies ist eine Funktion mit Ergebnismenge (SRF von set-returning function). Sie gibt eine Menge an `geomval` Zeilen für das Band zurück, die aus einer Geometrie (`geom`) und einem Pixelwert (`val`) gebildet werden. Ein Polygon entspricht der Vereinigung der Pixel in dem Band, die denselben Pixelwert "`val`" aufweisen.

`ST_DumpAsPolygon` ist nützlich um Raster in Polygone zu vektorisieren. Im Gegensatz zu `GROUP BY` werden hier neue Zeilen erzeugt. Die Funktion kann zum Beispiel verwendet werden, um einen einzelnen Raster in mehrere Polygone/Mehrfach-Polygone zu expandieren.

Geändert 3.3.0, Validierung und Fixierung ist deaktiviert, um die Leistung zu verbessern. Kann zu ungültigen Geometrien führen.

Verfügbarkeit: Benötigt GDAL 1.7+

**Note**

Wenn das Band einen NODATA-Wert aufweist, dann werden die Pixel mit diesem Wert nicht zurückgegeben, außer wenn `exclude_nodata_value=false`.

**Note**

Wenn Sie nur die Anzahl der Pixel des Raster benötigen, die einen bestimmten Zellwert haben, dann ist `ST_ValueCount` schneller.

**Note**

Dies unterscheidet sich von `ST_PixelAsPolygons`, wo eine Geometrie für jedes Pixel zurückgegeben wird, unabhängig vom Pixelwert.

**Beispiele**

```
-- this syntax requires PostgreSQL 9.3+
SELECT val, ST_AsText(geom) As geomwkt
FROM (
SELECT dp.*
FROM dummy_rast, LATERAL ST_DumpAsPolygons(rast) AS dp
WHERE rid = 2
) As foo
WHERE val BETWEEN 249 and 251
ORDER BY val;
```

val	geomwkt
249	POLYGON((3427927.95 5793243.95,3427927.95 5793243.85,3427928 5793243.85,3427928 5793243.95,3427927.95 5793243.95))
250	POLYGON((3427927.75 5793243.9,3427927.75 5793243.85,3427927.8 5793243.85,3427927.8 5793243.9,3427927.75 5793243.9))
250	POLYGON((3427927.8 5793243.8,3427927.8 5793243.75,3427927.85 5793243.75,3427927.85 5793243.8, 3427927.8 5793243.8))
251	POLYGON((3427927.75 5793243.85,3427927.75 5793243.8,3427927.8 5793243.8,3427927.8 5793243.85,3427927.8 5793243.85))

**Siehe auch**

[geomval](#), [ST\\_Value](#), [ST\\_Polygon](#), [ST\\_ValueCount](#)

**11.15.4 ST\_Envelope**

`ST_Envelope` — Stellt die Ausdehnung des Raster als Polygon dar.

**Synopsis**

geometry `ST_Envelope`(raster rast);

## Beschreibung

Gibt eine Polygondarstellung der Rasterausdehnung zurück. Dies ist das minimale umschreibendes Rechteck in Float8, das in dem durch die SRID festgelegten Koordinatenreferenzsystem als Polygon dargestellt wird.

Das Polygon wird durch die Eckpunkte des umschreibenden Rechtecks ((MINX, MINY), (MINX, MAXY), (MAXX, MAXY), (MAXX, MINY), (MINX, MINY)) festgelegt.

## Beispiele

```
SELECT rid, ST_AsText(ST_Envelope(rast)) As envgeomwkt
FROM dummy_rast;
```

rid	envgeomwkt
1	POLYGON((0 0,20 0,20 60,0 60,0 0))
2	POLYGON((3427927 5793243,3427928 5793243, 3427928 5793244,3427927 5793244, 3427927 5793243))

## Siehe auch

[ST\\_Envelope](#), [ST\\_AsText](#), [ST\\_SRID](#)

### 11.15.5 ST\_MinConvexHull

**ST\_MinConvexHull** — Gibt die Geometrie der konvexen Hülle des Raster aus, wobei Pixel mit NODATA ausgenommen werden.

## Synopsis

geometry **ST\_MinConvexHull**(raster rast, integer nband=NULL);

## Beschreibung

Gibt die Geometrie der konvexen Hülle des Raster aus, wobei Pixel mit NODATA ausgenommen werden. Wenn nband NULL ist, dann werden alle Bänder des Raster berücksichtigt.

Verfügbarkeit: 2.1.0

## Beispiele

```
WITH foo AS (
  SELECT
    ST_SetValues(
      ST_SetValues(
        ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(9, 9, 0, 0, 1, -1, 0, 0, 0), 1, '8 ←
          BUI', 0, 0), 2, '8BUI', 1, 0),
        1, 1, 1,
        ARRAY[
          [0, 0, 0, 0, 0, 0, 0, 0, 0],
          [0, 0, 0, 0, 0, 0, 0, 0, 0],
          [0, 0, 0, 0, 0, 0, 0, 0, 0],
          [0, 0, 0, 1, 0, 0, 0, 0, 1],
          [0, 0, 0, 1, 1, 0, 0, 0, 0],
          [0, 0, 0, 1, 0, 0, 0, 0, 0],
          [0, 0, 0, 0, 0, 0, 0, 0, 0]
```

```

        [0, 0, 0, 0, 0, 0, 0, 0, 0],
        [0, 0, 0, 0, 0, 0, 0, 0, 0]
    ]::double precision[][])
),
2, 1, 1,
ARRAY[
    [0, 0, 0, 0, 0, 0, 0, 0, 0],
    [0, 0, 0, 0, 0, 0, 0, 0, 0],
    [0, 0, 0, 0, 0, 0, 0, 0, 0],
    [1, 0, 0, 0, 0, 1, 0, 0, 0],
    [0, 0, 0, 0, 1, 1, 0, 0, 0],
    [0, 0, 0, 0, 0, 1, 0, 0, 0],
    [0, 0, 0, 0, 0, 0, 0, 0, 0],
    [0, 0, 0, 0, 0, 0, 0, 0, 0],
    [0, 0, 1, 0, 0, 0, 0, 0, 0]
]::double precision[][])
) AS rast
)
SELECT
    ST_AsText(ST_ConvexHull(rast)) AS hull,
    ST_AsText(ST_MinConvexHull(rast)) AS mhull,
    ST_AsText(ST_MinConvexHull(rast, 1)) AS mhull_1,
    ST_AsText(ST_MinConvexHull(rast, 2)) AS mhull_2
FROM foo

```

hull	mhull_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\\_MinConvexHull](#), [ST\\_AsText](#)

**11.15.6 ST\_Polygon**

ST\_Polygon — Gibt eine Geometrie mit Mehrfachpolygonen zurück, die aus der Vereinigung von Pixel mit demselben Zellwert gebildet werden. Pixel mit NODATA Werten werden nicht berücksichtigt. Wenn keine Band angegeben ist, wird die Bandnummer standardmäßig auf 1 gesetzt.

**Synopsis**

geometry **ST\_Polygon**(raster rast, integer band\_num=1);

**Beschreibung**

Geändert 3.3.0, Validierung und Fixierung ist deaktiviert, um die Leistung zu verbessern. Kann zu ungültigen Geometrien führen.

Verfügbarkeit: 0.1.6 - benötigt GDAL 1.7+

Erweiterung: 2.1.0 Geschwindigkeit verbessert (zur Gänze C-basiert) und es wird sichergestellt, dass das zurückgegebenen Mehrfachpolygon valide ist.

Änderung: 2.1.0 In Vorgängerversionen wurde manchmal ein Polygon zurückgegeben; dies wurde geändert so dass jetzt immer ein Mehrfachpolygon zurückgegeben wird.



**Beispiele**

```
-- by default no data band value is 0 or not set, so polygon will return a square polygon
SELECT ST_AsText(ST_Polygon(rast)) As geomwkt
FROM dummy_rast
WHERE rid = 2;

geomwkt
-----
MULTIPOLYGON(((3427927.75 5793244,3427928 5793244,3427928 5793243.75,3427927.75  ←
  5793243.75,3427927.75 5793244)))

-- now we change the no data value of first band
UPDATE dummy_rast SET rast = ST_SetBandNoDataValue(rast,1,254)
WHERE rid = 2;
SELECT rid, ST_BandNoDataValue(rast)
from dummy_rast where rid = 2;

-- ST_Polygon excludes the pixel value 254 and returns a multipolygon
SELECT ST_AsText(ST_Polygon(rast)) As geomwkt
FROM dummy_rast
WHERE rid = 2;

geomwkt
-----
MULTIPOLYGON(((3427927.9 5793243.95,3427927.85 5793243.95,3427927.85 5793244,3427927.9  ←
  5793244,3427927.9 5793243.95)),((3427928 5793243.85,3427928 5793243.8,3427927.95  ←
  5793243.8,3427927.95 5793243.85,3427927.9 5793243.85,3427927.9 5793243.9,3427927.9  ←
  5793243.95,3427927.95 5793243.95,3427928 5793243.95,3427928 5793243.85)),((3427927.8  ←
  5793243.75,3427927.75 5793243.75,3427927.75 5793243.8,3427927.75 5793243.85,3427927.75  ←
  5793243.9,3427927.75 5793244,3427927.8 5793244,3427927.8 5793243.9,3427927.8  ←
  5793243.85,3427927.85 5793243.85,3427927.85 5793243.8,3427927.85 5793243.75,3427927.8  ←
  5793243.75)))

-- Or if you want the no data value different for just one time

SELECT ST_AsText(
  ST_Polygon(
    ST_SetBandNoDataValue(rast,1,252)
  )
) As geomwkt
FROM dummy_rast
WHERE rid =2;

geomwkt
-----
MULTIPOLYGON(((3427928 5793243.85,3427928 5793243.8,3427928 5793243.75,3427927.85  ←
  5793243.75,3427927.8 5793243.75,3427927.8 5793243.8,3427927.75 5793243.8,3427927.75  ←
  5793243.85,3427927.75 5793243.9,3427927.75 5793244,3427927.8 5793244,3427927.85  ←
  5793244,3427927.9 5793244,3427928 5793244,3427928 5793243.95,3427928 5793243.85)  ←
  ,(3427927.9 5793243.9,3427927.9 5793243.85,3427927.95 5793243.85,3427927.95  ←
  5793243.9,3427927.9 5793243.9)))
```

**Siehe auch**

[ST\\_Value](#), [ST\\_DumpAsPolygons](#)

## 11.16 Rasteroperatoren

### 11.16.1 &&

`&&` — Gibt `TRUE` zurück, wenn das umschreibende Rechteck von A das umschreibende Rechteck von B schneidet.

#### Synopsis

```
boolean &&( raster A , raster B );
boolean &&( raster A , geometry B );
boolean &&( geometry B , raster A );
```

#### Beschreibung

Der Operator `&&` gibt `TRUE` zurück, wenn das umschreibende Rechteck von Raster/Geometrie A das umschreibende Rechteck von Raster/Geometrie B schneidet.



#### Note

Dieser Operand benützt jeden Index, der für den Raster zur Verfügung stellt.

Verfügbarkeit: 2.0.0

#### Beispiele

```
SELECT A.rid As a_rid, B.rid As b_rid, A.rast && B.rast As intersect
FROM dummy_rast AS A CROSS JOIN dummy_rast AS B LIMIT 3;
```

a_rid	b_rid	intersect
2	2	t
2	3	f
2	1	f

### 11.16.2 &<

`&<` — Gibt `TRUE` zurück, wenn das umschreibende Rechteck von A links von dem von B liegt.

#### Synopsis

```
boolean &<( raster A , raster B );
```

#### Beschreibung

Der `&<` Operator gibt `TRUE` zurück, wenn das umschreibende Rechteck von Raster A das umschreibende Rechteck von Raster B überlagert oder links davon liegt, oder präziser, überlagert und NICHT rechts des umschreibenden Rechtecks von Raster B liegt.



#### Note

Dieser Operand benützt jeden Index, der für den Raster zur Verfügung stellt.

## Beispiele

```
SELECT A.rid As a_rid, B.rid As b_rid, A.rast &< B.rast As overleft
FROM dummy_rast AS A CROSS JOIN dummy_rast AS B;
```

a_rid	b_rid	overleft
2	2	t
2	3	f
2	1	f
3	2	t
3	3	t
3	1	f
1	2	t
1	3	t
1	1	t

### 11.16.3 &>

**&>** — Gibt TRUE zurück, wenn das umschreibende Rechteck von A rechts von dem von B liegt.

#### Synopsis

```
boolean &>( raster A , raster B );
```

#### Beschreibung

Der **&>** Operator gibt TRUE zurück, wenn das umschreibende Rechteck von Raster A das umschreibende Rechteck von Raster B überlagert oder rechts davon liegt, oder präziser, überlagert und NICHT links des umschreibenden Rechtecks von Raster B liegt.



#### Note

Dieser Operand benützt jeden Index, der für die Geometrie zur Verfügung stellt.

## Beispiele

```
SELECT A.rid As a_rid, B.rid As b_rid, A.rast &
> B.rast As overright
FROM dummy_rast AS A CROSS JOIN dummy_rast AS B;
```

a_rid	b_rid	overright
2	2	t
2	3	t
2	1	t
3	2	f
3	3	t
3	1	f
1	2	f
1	3	t
1	1	t

### 11.16.4 =

= — Gibt TRUE zurück, wenn die umschreibenden Rechtecke von A und B ident sind. Das umschreibende Rechteck ist in Double Precision.

#### Synopsis

```
boolean =( raster A , raster B );
```

#### Beschreibung

Der = Operator gibt TRUE zurück, wenn das umschreibende Rechteck von Raster A ident mit dem umschreibenden Rechteck von Raster B ist. PostgreSQL verwendet die =, <, und > Operatoren um die interne Sortierung und den Vergleich von Raster durchzuführen (z.B.: in einer GROUP BY oder ORDER BY Klausel).



#### Caution

Dieser Operator verwendet NICHT die für Raster verfügbaren Indizes. Verwenden Sie bitte ~= stattdessen. Dieser Operator existiert meistens, so dass man nach der Rasterspalte gruppieren kann.

---

Verfügbarkeit: 2.1.0

#### Siehe auch

~=

### 11.16.5 @

@ — Gibt TRUE zurück, wenn das umschreibende Rechteck von A in jenem von B enthalten ist. Das umschreibende Rechteck ist in Double Precision.

#### Synopsis

```
boolean @( raster A , raster B );  
boolean @( geometry A , raster B );  
boolean @( raster B , geometry A );
```

#### Beschreibung

Der @ Operator gibt TRUE zurück, wenn das umschreibende Rechteck von Raster/Geometrie A in dem umschreibenden Rechteck von Raster/Geometrie B enthalten ist.



#### Note

Dieser Operand verwendet die räumlichen Indizes der Raster.

---

Verfügbarkeit: 2.0.0 raster @ raster, und raster @ geometry eingeführt

Verfügbarkeit: 2.0.5 "geometry @ raster" eingeführt

---

**Siehe auch**

~

**11.16.6 ~=**

~= — Gibt TRUE zurück, wenn die bounding box von A ident mit jener von B ist.

**Synopsis**

```
boolean ~= ( raster A , raster B );
```

**Beschreibung**

Der Operator ~= gibt TRUE zurück, wenn die Umgebungsrechtecke der Raster "A" und "B" ident sind.

**Note**

Dieser Operand benützt jeden Index, der für den Raster zur Verfügung stellt.

Verfügbarkeit: 2.0.0

**Beispiele**

Ein sinnvoller Anwendungsfall wäre, aus zwei Einzelbandraster mit den gleichen Eigenschaften aber unterschiedlicher Thematik, einen Multi-Bandraster zu erstellen.

```
SELECT ST_AddBand(prec.rast, alt.rast) As new_rast
FROM prec INNER JOIN alt ON (prec.rast ~= alt.rast);
```

**Siehe auch**

[ST\\_AddBand](#), [=](#)

**11.16.7 ~**

~ — Gibt TRUE zurück, wenn das umschreibende Rechteck von A jenes von B enthält. Das umschreibende Rechteck ist in Double Precision.

**Synopsis**

```
boolean ~( raster A , raster B );
boolean ~( geometry A , raster B );
boolean ~( raster B , geometry A );
```

## Beschreibung

Der Operator `~` gibt `TRUE` zurück, wenn das umschreibende Rechteck von Raster/Geometrie A das umschreibende Rechteck von Raster/Geometrie B enthält.



### Note

Dieser Operand verwendet die räumlichen Indizes der Raster.

Verfügbarkeit: 2.0.0

## Siehe auch

@

## 11.17 Räumliche Beziehungen von Rastern und Rasterbändern

### 11.17.1 ST\_Contains

`ST_Contains` — Gibt `TRUE` zurück, wenn kein Punkt des Rasters "rastB" im Äußeren des Rasters "rastA" liegt und zumindest ein Punkt im Inneren von "rastB" auch im Inneren von "rastA" liegt.

## Synopsis

boolean `ST_Contains`( raster rastA , integer nbandA , raster rastB , integer nbandB );

boolean `ST_Contains`( raster rastA , raster rastB );

## Beschreibung

Raster "rastA" enthält dann und nur dann "rastB", wenn sich kein Punkt von "rastB" im Äußeren des Rasters "rastA" befindet und zumindest ein Punkt im Inneren von "rastB" auch im Inneren von "rastA" liegt. Wenn die Bandnummer nicht angegeben ist (oder auf `NULL` gesetzt), dann wird nur die konvexe Hülle des Rasters zur Überprüfung herangezogen. Wenn die Bandnummer angegeben ist, werden nur die Pixel mit einem Wert (nicht `NODATA`) bei der Überprüfung herangezogen.



### Note

Diese Funktion verwendet die für Raster verfügbaren Indizes.



### Note

Um die räumliche Beziehung zwischen einem Raster und einem geometrischen Datentyp zu überprüfen, verwenden Sie bitte `ST_Polygon` für den Raster, z.B. `ST_Contains(ST_Polygon(raster), geometry)` or `ST_Contains(geometry, ST_Polygon(raster))`.



### Note

`ST_Contains()` ist das Gegenstück zu `ST_Within()`. Daher impliziert `ST_Contains(rastA, rastB)` `ST_Within(rastB, rastA)`.

Verfügbarkeit: 2.1.0

## Beispiele

```
-- specified band numbers
SELECT r1.rid, r2.rid, ST_Contains(r1.rast, 1, r2.rast, 1) FROM dummy_rast r1 CROSS JOIN ↔
dummy_rast r2 WHERE r1.rid = 1;
```

NOTICE: The first raster provided has no bands

```
rid | rid | st_contains
-----+-----+-----
 1 | 1 |
 1 | 2 | f
```

```
-- no band numbers specified
```

```
SELECT r1.rid, r2.rid, ST_Contains(r1.rast, r2.rast) FROM dummy_rast r1 CROSS JOIN ↔
dummy_rast r2 WHERE r1.rid = 1;
```

```
rid | rid | st_contains
-----+-----+-----
 1 | 1 | t
 1 | 2 | f
```

## Siehe auch

[ST\\_Intersects](#), [ST\\_Within](#)

### 11.17.2 ST\_ContainsProperly

**ST\_ContainsProperly** — Gibt TRUE zurück, wenn "rastB" das Innere von "rastA" schneidet, aber nicht die Begrenzung oder das Äußere von "rastA".

#### Synopsis

```
boolean ST_ContainsProperly( raster rastA , integer nbandA , raster rastB , integer nbandB );
boolean ST_ContainsProperly( raster rastA , raster rastB );
```

#### Beschreibung

Raster "rastA" enthält "rastB" vollständig, wenn "rastB" nur das Innere von "rastA" schneidet, die Begrenzung und das Äußere von "rastA" jedoch nicht. Wenn die Bandnummer nicht angegeben ist (oder auf NULL gesetzt), dann wird nur die konvexe Hülle des Rasters zur Überprüfung herangezogen. Wenn die Bandnummer angegeben ist, werden nur die Pixel mit einem Wert (nicht NODATA) bei der Überprüfung herangezogen.

Raster "rastA" enthält sich selbst, aber enthält sich nicht zur Gänze selbst.



#### Note

Diese Funktion verwendet die für Raster verfügbaren Indizes.



#### Note

Um die räumliche Beziehung zwischen einem Raster und einem geometrischen Datentyp zu überprüfen, verwenden Sie bitte `ST_Polygon` für den Raster, z.B. `ST_ContainsProperly(ST_Polygon(raster), geometry)` or `ST_ContainsProperly(geometry, ST_Polygon(raster))`.

Verfügbarkeit: 2.1.0

## Beispiele

```
SELECT r1.rid, r2.rid, ST_ContainsProperly(r1.rast, 1, r2.rast, 1) FROM dummy_rast r1 CROSS JOIN
  JOIN dummy_rast r2 WHERE r1.rid = 2;
```

rid	rid	st_containsproperly
2	1	f
2	2	f

## Siehe auch

[ST\\_Intersects](#), [ST\\_Contains](#)

### 11.17.3 ST\_Covers

`ST_Covers` — Gibt TRUE zurück, wenn kein Punkt des Rasters "rastB" außerhalb des Rasters "rastA" liegt.

## Synopsis

```
boolean ST_Covers( raster rastA , integer nbandA , raster rastB , integer nbandB );
boolean ST_Covers( raster rastA , raster rastB );
```

## 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
  JOIN dummy_rast r2 WHERE r1.rid = 2;
```

rid	rid	st_covers
2	1	f
2	2	t



**Siehe auch**[ST\\_Intersects](#), [ST\\_CoveredBy](#)**11.17.4 ST\_CoveredBy**

`ST_CoveredBy` — Gibt TRUE zurück, wenn kein Punkt des Rasters "rastA" außerhalb des Rasters "rastB" liegt.

**Synopsis**

boolean `ST_CoveredBy`( raster rastA , integer nbandA , raster rastB , integer nbandB );

boolean `ST_CoveredBy`( raster rastA , raster rastB );

**Beschreibung**

Raster "rastA" wird von "rastB" dann und nur dann abgedeckt, wenn kein Punkt von "rastA" im Äußeren von "rastB" liegt. Wenn die Bandnummer nicht angegeben ist (oder auf NULL gesetzt), dann wird nur die konvexe Hülle des Rasters zur Überprüfung herangezogen. Wenn die Bandnummer angegeben ist, werden nur die Pixel mit einem Wert (nicht NODATA) bei der Überprüfung herangezogen.

**Note**

Diese Funktion verwendet die für Raster verfügbaren Indizes.

**Note**

Um die räumliche Beziehung zwischen einem Raster und einem geometrischen Datentyp zu überprüfen, verwenden Sie bitte `ST_Polygon` für den Raster, z.B. `ST_CoveredBy(ST_Polygon(raster), geometry)` or `ST_CoveredBy(geometry, ST_Polygon(raster))`.

Verfügbarkeit: 2.1.0

**Beispiele**

```
SELECT r1.rid, r2.rid, ST_CoveredBy(r1.rast, 1, r2.rast, 1) FROM dummy_rast r1 CROSS JOIN ↔
dummy_rast r2 WHERE r1.rid = 2;
```

```
rid | rid | st_coveredby
-----+-----+-----
  2 |  1 | f
  2 |  2 | t
```

**Siehe auch**[ST\\_Intersects](#), [ST\\_Covers](#)**11.17.5 ST\_Disjoint**

`ST_Disjoint` — Gibt TRUE zurück, wenn sich die Raster "rastA" und "rastB" räumlich nicht überschneiden.

## Synopsis

```
boolean ST_Disjoint( raster rastA , integer nbandA , raster rastB , integer nbandB );
boolean ST_Disjoint( raster rastA , raster rastB );
```

## Beschreibung

Raster "rastA" und "rastB" sind getrennt voneinander (disjoint) wenn sie sich keinen gemeinsamen Raum teilen. Wenn die Bandnummer nicht angegeben ist (oder auf NULL gesetzt), dann wird nur die konvexe Hülle des Rasters zur Überprüfung herangezogen. Wenn die Bandnummer angegeben ist, werden nur die Pixel mit einem Wert (nicht NODATA) bei der Überprüfung herangezogen.



### Note

Diese Funktion verwendet keine Indizes.



### Note

Um die räumliche Beziehung zwischen einem Raster und einem geometrischen Datentyp zu überprüfen, verwenden Sie bitte ST\_Polygon für den Raster, z.B. ST\_Disjoint(ST\_Polygon(raster), geometry).

Verfügbarkeit: 2.1.0

## Beispiele

```
-- rid = 1 has no bands, hence the NOTICE and the NULL value for st_disjoint
SELECT r1.rid, r2.rid, ST_Disjoint(r1.rast, 1, r2.rast, 1) FROM dummy_rast r1 CROSS JOIN ←
    dummy_rast r2 WHERE r1.rid = 2;
```

NOTICE: The second raster provided has no bands

```
rid | rid | st_disjoint
-----+-----+-----
 2 |  1 |
 2 |  2 | f
```

```
-- this time, without specifying band numbers
```

```
SELECT r1.rid, r2.rid, ST_Disjoint(r1.rast, r2.rast) FROM dummy_rast r1 CROSS JOIN ←
    dummy_rast r2 WHERE r1.rid = 2;
```

```
rid | rid | st_disjoint
-----+-----+-----
 2 |  1 | t
 2 |  2 | f
```

## Siehe auch

[ST\\_Intersects](#)

### 11.17.6 ST\_Intersects

ST\_Intersects — Gibt TRUE zurück, wenn sich die Raster "rastA" und "rastB" nicht räumlich überschneiden.

## Synopsis

```
boolean ST_Intersects( raster rastA , integer nbandA , raster rastB , integer nbandB );
boolean ST_Intersects( raster rastA , raster rastB );
boolean ST_Intersects( raster rast , integer nband , geometry geommin );
boolean ST_Intersects( raster rast , geometry geommin , integer nband=NULL );
boolean ST_Intersects( geometry geommin , raster rast , integer nband=NULL );
```

## Beschreibung

Gibt TRUE zurück, wenn der Raster "rastA" den Raster "rastB" räumlich schneidet. Wenn die Bandnummer nicht angegeben ist (oder auf NULL gesetzt), dann wird nur die konvexe Hülle des Rasters zur Überprüfung herangezogen. Wenn die Bandnummer angegeben ist, werden nur die Pixel mit einem Wert (nicht NODATA) bei der Überprüfung herangezogen.



### Note

Diese Funktion verwendet die für Raster verfügbaren Indizes.

Erweiterung: 2.0.0 Unterstützung für Raster/Raster Verschneidungen eingeführt.



### Warning

Änderung: 2.1.0 Die Verhaltensweise der Varianten von ST\_Intersects(raster, geometry) wurde geändert um mit dem von ST\_Intersects(geometry, raster) übereinzustimmen.

## Beispiele

```
-- different bands of same raster
SELECT ST_Intersects(rast, 2, rast, 3) FROM dummy_rast WHERE rid = 2;

st_intersects
-----
t
```

## Siehe auch

[ST\\_Intersection](#), [ST\\_Disjoint](#)

### 11.17.7 ST\_Overlaps

ST\_Overlaps — Gibt TRUE zurück, wenn sich die Raster "rastA" und "rastB" schneiden, aber ein Raster den anderen nicht zur Gänze enthält.

## Synopsis

```
boolean ST_Overlaps( raster rastA , integer nbandA , raster rastB , integer nbandB );
boolean ST_Overlaps( raster rastA , raster rastB );
```

## Beschreibung

Gibt TRUE zurück, wenn der Raster "rastA" den Raster "rastB" überlagert. Dies bedeutet, dass rastA und rastB sich schneiden, aber einer den anderen nicht zur Gänze enthält. Wenn die Bandnummer nicht angegeben ist (oder auf NULL gesetzt), dann wird nur die konvexe Hülle des Rasters zur Überprüfung herangezogen. Wenn die Bandnummer angegeben ist, werden nur die Pixel mit einem Wert (nicht NODATA) bei der Überprüfung herangezogen.



### Note

Diese Funktion verwendet die für Raster verfügbaren Indizes.



### Note

Um die räumliche Beziehung zwischen einem Raster und einem geometrischen Datentyp zu überprüfen, verwenden Sie bitte ST\_Polygon für den Raster, z.B. ST\_Overlaps(ST\_Polygon(raster), geometry).

Verfügbarkeit: 2.1.0

## Beispiele

```
-- comparing different bands of same raster
SELECT ST_Overlaps(rast, 1, rast, 2) FROM dummy_rast WHERE rid = 2;

st_overlaps
-----
f
```

## Siehe auch

[ST\\_Intersects](#)

### 11.17.8 ST\_Touches

ST\_Touches — Gibt TRUE zurück, wenn rastA und rastB zumindest einen Punkt gemeinsam haben, sich aber nicht überschneiden.

## Synopsis

```
boolean ST_Touches( raster rastA , integer nbandA , raster rastB , integer nbandB );
boolean ST_Touches( raster rastA , raster rastB );
```

## Beschreibung

Gibt TRUE zurück, wenn der Raster "rastA" den Raster "rastB" räumlich berührt. Dies bedeutet, dass rastA und rastB zumindest einen Punkt gemeinsam haben, ihr Inneres sich aber nicht überschneidet. Wenn die Bandnummer nicht angegeben ist (oder auf NULL gesetzt), dann wird nur die konvexe Hülle des Rasters zur Überprüfung herangezogen. Wenn die Bandnummer angegeben ist, werden nur die Pixel mit einem Wert (nicht NODATA) bei der Überprüfung herangezogen.

**Note**

Diese Funktion verwendet die für Raster verfügbaren Indizes.

**Note**

Um die räumliche Beziehung zwischen einem Raster und einem geometrischen Datentyp zu überprüfen, verwenden Sie bitte `ST_Polygon` für den Raster, z.B. `ST_Touches(ST_Polygon(raster), geometry)`.

Verfügbarkeit: 2.1.0

**Beispiele**

```
SELECT r1.rid, r2.rid, ST_Touches(r1.rast, 1, r2.rast, 1) FROM dummy_rast r1 CROSS JOIN ↵
    dummy_rast r2 WHERE r1.rid = 2;
```

```
rid | rid | st_touches
-----+-----+-----
  2 |  1 | f
  2 |  2 | f
```

**Siehe auch**

[ST\\_Intersects](#)

**11.17.9 ST\_SameAlignment**

`ST_SameAlignment` — Gibt TRUE zurück, wenn die Raster die selbe Rotation, Skalierung, Koordinatenreferenzsystem und Versatz (Pixel können auf dasselbe Gitter gelegt werden, ohne dass die Gitterlinien durch die Pixel schneiden) aufweisen. Wenn nicht, wird FALSE und eine Beschreibung des Problems ausgegeben.

**Synopsis**

```
boolean ST_SameAlignment( raster rastA , raster rastB );
boolean ST_SameAlignment( double precision ulx1 , double precision uly1 , double precision scalex1 , double precision scaley1
, double precision skewx1 , double precision skewy1 , double precision ulx2 , double precision uly2 , double precision scalex2 ,
double precision scaley2 , double precision skewx2 , double precision skewy2 );
boolean ST_SameAlignment( raster set rastfield );
```

**Beschreibung**

Nicht-Aggregat Version (Versionen 1 und 2): Gibt TRUE zurück, wenn die zwei Raster (die entweder direkt übergeben oder mit Werten für UpperLeft, Scale, Skew und SRID erstellt werden) dieselbe Skalierung, Rotation und SRID haben und zumindest eine der vier Ecken eines Pixel des einen Raster auf eine Ecke des anderen Rastergitters fällt. Wenn nicht, wird FALSE und eine Problembeschreibung ausgegeben.

Aggregat Version (Variante 3): Gibt TRUE zurück, wenn alle übergebenen Raster gleich ausgerichtet sind. Die Funktion `ST_SameAlignment()` ist in der Terminologie von PostgreSQL eine Aggregatfunktion. Dies bedeutet, dass sie so wie die Funktionen `SUM()` und `AVG()` mit Datenzeilen arbeitet.

Verfügbarkeit: 2.0.0

Erweiterung: 2.1.0 die Variante mit der Aggregatfunktion hinzugefügt

**Beispiele: Raster**

```
SELECT ST_SameAlignment(
  ST_MakeEmptyRaster(1, 1, 0, 0, 1, 1, 0, 0),
  ST_MakeEmptyRaster(1, 1, 0, 0, 1, 1, 0, 0)
) as sm;
```

```
sm
----
t
```

```
SELECT ST_SameAlignment(A.rast,b.rast)
FROM dummy_rast AS A CROSS JOIN dummy_rast AS B;
```

```
NOTICE: The two rasters provided have different SRIDs
NOTICE: The two rasters provided have different SRIDs
st_samealignment
```

```
-----
t
f
f
f
```

**Siehe auch**

Section [10.1](#), [ST\\_NotSameAlignmentReason](#), [ST\\_MakeEmptyRaster](#)

**11.17.10 ST\_NotSameAlignmentReason**

`ST_NotSameAlignmentReason` — Gibt eine Meldung aus, die angibt ob die Raster untereinander ausgerichtet sind oder nicht und warum wenn nicht.

**Synopsis**

text `ST_NotSameAlignmentReason`(raster rastA, raster rastB);

**Beschreibung**

Gibt eine Meldung aus, die angibt ob die Raster untereinander ausgerichtet sind oder nicht und warum wenn nicht.

**Note**

Falls es mehrere Gründe gibt, warum die Raster nicht untereinander ausgerichtet sind, so wird nur der erste Grund (die erste fehlgeschlagene Überprüfung) ausgegeben.

Verfügbarkeit: 2.1.0

**Beispiele**

```

SELECT
  ST_SameAlignment (
    ST_MakeEmptyRaster(1, 1, 0, 0, 1, 1, 0, 0),
    ST_MakeEmptyRaster(1, 1, 0, 0, 1.1, 1.1, 0, 0)
  ),
  ST_NotSameAlignmentReason (
    ST_MakeEmptyRaster(1, 1, 0, 0, 1, 1, 0, 0),
    ST_MakeEmptyRaster(1, 1, 0, 0, 1.1, 1.1, 0, 0)
  )
;

st_samealignment |          st_notsamealignmentreason
-----+-----
f                | The rasters have different scales on the X axis
(1 row)

```

**Siehe auch**

Section [10.1, ST\\_SameAlignment](#)

**11.17.11 ST\_Within**

**ST\_Within** — Gibt TRUE zurück, wenn kein Punkt des Rasters "rastA" außerhalb des Rasters "rastB" liegt und zumindest ein Punkt im Inneren von "rastA" auch im Inneren von "rastB" liegt.

**Synopsis**

boolean **ST\_Within**( raster rastA , integer nbandA , raster rastB , integer nbandB );  
 boolean **ST\_Within**( raster rastA , raster rastB );

**Beschreibung**

Raster "rastA" liegt dann und nur dann "within"/innerhalb von "rastB", wenn sich kein Punkt von "rastA" im Äußeren des Rasters "rastB" befindet und zumindest ein Punkt im Inneren von "rastA" auch im Inneren von "rastB" liegt. Wenn die Bandnummer nicht angegeben ist (oder auf NULL gesetzt), dann wird nur die konvexe Hülle des Rasters zur Überprüfung herangezogen. Wenn die Bandnummer angegeben ist, werden nur die Pixel mit einem Wert (nicht NODATA) bei der Überprüfung herangezogen.

**Note**

Dieser Operand benützt jeden Index, der für den Raster zur Verfügung stellt.

**Note**

Um die räumliche Beziehung zwischen einem Raster und einer Geometrie zu überprüfen, verwenden Sie bitte `ST_Polygon` für den Raster, z.B. `ST_Within(ST_Polygon(raster), geometry)` or `ST_Within(geometry, ST_Polygon(raster))`.

**Note**

`ST_Within()` ist die Umkehrfunktion von `ST_Contains()`. Es gilt daher: `ST_Within(rastA, rastB)` impliziert `ST_Contains(rastB, rastA)`.

Verfügbarkeit: 2.1.0

## Beispiele

```
SELECT r1.rid, r2.rid, ST_Within(r1.rast, 1, r2.rast, 1) FROM dummy_rast r1 CROSS JOIN ↵
    dummy_rast r2 WHERE r1.rid = 2;
```

```
rid | rid | st_within
-----+-----+-----
  2 |  1 | f
  2 |  2 | t
```

## Siehe auch

[ST\\_Intersects](#), [ST\\_Contains](#), [ST\\_DWithin](#), [ST\\_DFullyWithin](#)

### 11.17.12 ST\_DWithin

**ST\_DWithin** — Gibt TRUE zurück, wenn die Raster "rastA" und "rastB" innerhalb der angegebenen Entfernung voneinander liegen.

## Synopsis

```
boolean ST_DWithin( raster rastA , integer nbandA , raster rastB , integer nbandB , double precision distance_of_srid );
boolean ST_DWithin( raster rastA , raster rastB , double precision distance_of_srid );
```

## Beschreibung

Gibt TRUE zurück, wenn die Raster "rastA" und "rastB" innerhalb der angegebenen Distanz zueinander liegen. Wenn die Bandnummer nicht angegeben ist (oder auf NULL gesetzt), dann wird nur die konvexe Hülle des Rasters zur Überprüfung herangezogen. Wenn die Bandnummer angegeben ist, werden nur die Pixel mit einem Wert (nicht NODATA) bei der Überprüfung herangezogen.

Die Distanz wird in den Einheiten des Koordinatenreferenzsystems des Rasters angegeben. Damit diese Funktion Sinn hat, müssen die Quellraster dieselbe Projektion und die gleiche SRID haben.



### Note

Dieser Operand benützt jeden Index, der für den Raster zur Verfügung stellt.



### Note

Um die räumliche Beziehung zwischen einem Raster und einer Geometrie zu untersuchen, wenden Sie bitte `ST_Polygon` auf den Raster an, z.B.: `ST_DWithin(ST_Polygon(raster), geometry)`.

Verfügbarkeit: 2.1.0

## Beispiele

```
SELECT r1.rid, r2.rid, ST_DWithin(r1.rast, 1, r2.rast, 1, 3.14) FROM dummy_rast r1 CROSS ↵
    JOIN dummy_rast r2 WHERE r1.rid = 2;
```

```
rid | rid | st_dwithin
-----+-----+-----
  2 |  1 | f
  2 |  2 | t
```



**Siehe auch**[ST\\_Within](#), [ST\\_DFullyWithin](#)**11.17.13 ST\_DFullyWithin**

`ST_DFullyWithin` — Gibt TRUE zurück, wenn die Raster "rastA" und "rastB" zur Gänze innerhalb der angegebenen Distanz zueinander liegen.

**Synopsis**

```
boolean ST_DFullyWithin( raster rastA , integer nbandA , raster rastB , integer nbandB , double precision distance_of_srid );
boolean ST_DFullyWithin( raster rastA , raster rastB , double precision distance_of_srid );
```

**Beschreibung**

Gibt TRUE zurück, wenn die Raster "rastA" und "rastB" zur Gänze innerhalb der angegebenen Distanz zueinander liegen. Wenn die Bandnummer nicht angegeben ist (oder auf NULL gesetzt), dann wird nur die konvexe Hülle des Rasters zur Überprüfung herangezogen. Wenn die Bandnummer angegeben ist, werden nur die Pixel mit einem Wert (nicht NODATA) bei der Überprüfung herangezogen.

Die Distanz wird in den Einheiten des Koordinatenreferenzsystems des Rasters angegeben. Damit diese Funktion Sinn hat, müssen die Quellraster dieselbe Projektion und die gleiche SRID haben.

**Note**

Dieser Operand benützt jeden Index, der für den Raster zur Verfügung stellt.

**Note**

Um die räumliche Relation zwischen einem Raster und einer Geometrie zu untersuchen, wenden Sie bitte `ST_Polygon` auf den Raster an, z.B.: `ST_DFullyWithin(ST_Polygon(raster), geometry)`.

Verfügbarkeit: 2.1.0

**Beispiele**

```
SELECT r1.rid, r2.rid, ST_DFullyWithin(r1.rast, 1, r2.rast, 1, 3.14) FROM dummy_rast r1 ↔
      CROSS JOIN dummy_rast r2 WHERE r1.rid = 2;
```

```
rid | rid | st_dfullywithin
-----+-----+-----
  2 |  1 | f
  2 |  2 | t
```

**Siehe auch**[ST\\_Within](#), [ST\\_DWithin](#)

## 11.18 Raster Tipps

### 11.18.1 Out-DB Raster

#### 11.18.1.1 Das Verzeichnis enthält eine Vielzahl an Dateien

Wenn GDAL eine Datei öffnet, dann liest es eifrig das gesamte Verzeichnis in dem sich die Datei befindet um einen Katalog mit den weiteren Dateien zu erstellen. Wenn dieses Verzeichnis viele Dateien (z.B.: Tausende, Millionen) enthält, kann das Öffnen dieser Datei extrem lange dauern (insbesondere wenn sich die Datei auf einem Netzlaufwerk, wie einem NFS befindet).

Dieses Verhalten kann durch folgende Umgebungsvariable von GDAL beeinflusst werden: **GDAL\_DISABLE\_READDIR\_ON\_OPEN**. Setzen Sie **GDAL\_DISABLE\_READDIR\_ON\_OPEN** auf **TRUE** um das Scannen von Verzeichnissen zu verhindern.

Auf Ubuntu (angenommen Sie verwenden ein PostgreSQL Paket für Ubuntu), kann **GDAL\_DISABLE\_READDIR\_ON\_OPEN** in `/etc/postgresql/POSTGRESQL_VERSION/CLUSTER_NAME/environment` gesetzt werden (wobei **POSTGRESQL\_VERSION** der Version von PostgreSQL entspricht, z.B. 9.6 und **CLUSTER\_NAME** der Bezeichnung des Datenbankclusters, z.B. maindb). Sie können hier ebenso die Umgebungsvariablen von PostGIS setzen.

```
# environment variables for postmaster process
# This file has the same syntax as postgresql.conf:
# VARIABLE = simple_value
# VARIABLE2 = 'any value!'
# I. e. you need to enclose any value which does not only consist of letters,
# numbers, and '-', '_', '.' in single quotes. Shell commands are not
# evaluated.
POSTGIS_GDAL_ENABLED_DRIVERS = 'ENABLE_ALL'

POSTGIS_ENABLE_OUTDB_RASTERS = 1

GDAL_DISABLE_READDIR_ON_OPEN = 'TRUE'
```

#### 11.18.1.2 Die maximale Anzahl geöffneter Dateien

Die Einstellungen von Linux und PostgreSQL bezüglich der maximal erlaubten Anzahl von offenen Dateien sind üblicherweise sehr konservativ (normalerweise 1024 offene Dateien pro Prozess), da sie unter der Annahme getroffen wurden, dass das System von Menschen genutzt wird. Bei Out-DB Rastern kann eine einzelne Abfrage spielend dieses Limit überschreiten (z.B. ein Datensatz mit Rasterwerten über 10 Jahre, wobei ein Raster die Tageswerte der niedrigsten und höchsten Temperaturwerte enthält und wir den absoluten Mindest- und Höchstwert des Datensatzes abfragen wollen).

Am einfachsten kann dies über die PostgreSQL Einstellung **max\_files\_per\_process** geändert werden. Der Standardwert von 1000 ist für Out-DB Raster viel zu niedrig. Ein zuverlässiger Anfangswert könnte 65536 sein, wobei dies jedoch sehr stark von den verwendeten Datensätzen abhängt und den Abfragen die Sie auf diese ausführen wollen. Diese Einstellung muss vor dem Starten des Servers gesetzt werden und kann vermutlich nur in der Konfigurationsdatei von PostgreSQL (z.B. `/etc/postgresql/POSTGRESQL_VERSION/CLUSTER_NAME/postgresql.conf` auf Ubuntu) vorgenommen werden.

```
...
# - Kernel Resource Usage -

max_files_per_process = 65536          # min 25
                                       # (change requires restart)

...
```

Die wesentliche Änderung muss an den Limits des Linux Kernels für offene Dateien vorgenommen werden. Dies umfasst zwei Teile:

- Die maximale Anzahl geöffneter Dateien für das ganze System
- Die maximale Anzahl geöffneter Dateien pro Prozess

### 11.18.1.2.1 Die maximale Anzahl geöffneter Dateien für das ganze System

Das folgende Beispiel zeigt, wie Sie die aktuelle Einstellung zu der maximalen Anzahl geöffneter Dateien für das ganze System anzeigen können:

```
$ sysctl -a | grep fs.file-max
fs.file-max = 131072
```

Wenn der ausgegebene Wert zu niedrig ist, können Sie wie im folgenden Beispiel gezeigt wird, eine Datei zu */etc/sysctl.d/* hinzufügen:

```
$ echo "fs.file-max = 6145324" >> /etc/sysctl.d/fs.conf

$ cat /etc/sysctl.d/fs.conf
fs.file-max = 6145324

$ sysctl -p --system
* Applying /etc/sysctl.d/fs.conf ...
fs.file-max = 2097152
* Applying /etc/sysctl.conf ...

$ sysctl -a | grep fs.file-max
fs.file-max = 6145324
```

### 11.18.1.2.2 Die maximale Anzahl geöffneter Dateien pro Prozess

Die maximale Anzahl geöffneter Dateien pro Process für die Serverprozesse von PostgreSQL sollten geändert werden.

Um die maximale Anzahl geöffneter Dateien herauszufinden, welche von den Prozessen des PostgreSQL Dienstes genutzt werden, können Sie folgendes ausführen (stellen Sie sicher, dass PostgreSQL läuft):

```
$ ps aux | grep postgres
postgres 31713  0.0  0.4 179012 17564 pts/0    S   Dec26   0:03 /home/dustymugs/devel/ ↵
    postgresql/sandbox/10/usr/local/bin/postgres -D /home/dustymugs/devel/postgresql/sandbox ↵
    /10/pgdata
postgres 31716  0.0  0.8 179776 33632 ?        Ss   Dec26   0:01 postgres: checkpointer ↵
    process
postgres 31717  0.0  0.2 179144  9416 ?        Ss   Dec26   0:05 postgres: writer process
postgres 31718  0.0  0.2 179012  8708 ?        Ss   Dec26   0:06 postgres: wal writer ↵
    process
postgres 31719  0.0  0.1 179568  7252 ?        Ss   Dec26   0:03 postgres: autovacuum ↵
    launcher process
postgres 31720  0.0  0.1  34228  4124 ?        Ss   Dec26   0:09 postgres: stats collector ↵
    process
postgres 31721  0.0  0.1 179308  6052 ?        Ss   Dec26   0:00 postgres: bgworker: ↵
    logical replication launcher

$ cat /proc/31718/limits
Limit                Soft Limit           Hard Limit           Units
Max cpu time         unlimited            unlimited            seconds
Max file size        unlimited            unlimited            bytes
Max data size        unlimited            unlimited            bytes
Max stack size       8388608              unlimited            bytes
Max core file size   0                    unlimited            bytes
Max resident set     unlimited            unlimited            bytes
Max processes        15738                15738                processes
Max open files      1024                4096                files
Max locked memory    65536                65536                bytes
Max address space    unlimited            unlimited            bytes
Max file locks       unlimited            unlimited            locks
Max pending signals  15738                15738                signals
```

Max msgqueue size	819200	819200	bytes
Max nice priority	0	0	
Max realtime priority	0	0	
Max realtime timeout	unlimited	unlimited	us

Im oberen Beispiel haben wir die Limits für geöffnete Dateien für den Prozess 31718 ausgelesen. Es spielt dabei keine Rolle welcher Prozess von PostgreSQL, es genügt jeder. Von Interesse ist dabei die Rückmeldung von *Max open files*.

Wir wollen das *Soft Limit* und das *Hard Limit* für die *Max open files* so erhöhen, dass sie über dem Wert liegen den wir in der Einstellung von PostgreSQL für `max_files_per_process` angegeben haben. In unserem Beispiel haben wir `max_files_per_pro` mit 65536 angegeben.

Auf Ubuntu (angenommen Sie verwenden ein PostgreSQL Paket für Ubuntu), kann das *Soft Limit* und das *Hard Limit* am einfachsten durch editieren von `/etc/init.d/postgresql` (SysV) oder `/lib/systemd/system/postgresql*.service` (systemd) geändert werden.

Befassen wir uns zuerst mit dem Fall SysV auf Ubuntu, bei dem wir `ulimit -H -n 262144` und `ulimit -n 131072` zu `/etc/init.d/postgresql` hinzufügen.

```
...
case "$1" in
  start|stop|restart|reload)
    if [ "$1" = "start" ]; then
      create_socket_directory
    fi
    if [ -z "`pg_lsclusters -h`" ]; then
      log_warning_msg 'No PostgreSQL clusters exist; see "man pg_createcluster"'
      exit 0
    fi

    ulimit -H -n 262144
    ulimit -n 131072

    for v in $versions; do
      $1 $v || EXIT=$?
    done
    exit ${EXIT:-0}
    ;;
  status)
...

```

Nun der Fall mit systemd unter Ubuntu. Wir fügen `LimitNOFILE=131072` in jeder Datei `/lib/systemd/system/postgresql*.service` in dem Abschnitt `[Service]` ein.

```
...
[Service]

LimitNOFILE=131072

...

[Install]
WantedBy=multi-user.target
...

```

Nach den erforderlichen systemd Änderungen müssen Sie den Dämon neu laden

```
systemctl daemon-reload
```

## Chapter 12

# PostGIS Extras

Dieses Kapitel beschreibt Funktionen, die sich in dem Verzeichnis "extras" des PostGIS Quellcodes (Tarball oder Repository) befinden. Diese sind nicht immer mit der binären PostGIS Release paketierte, es handelt sich dabei aber üblicherweise um Pl/Pgsql- oder Shell-Skripts, die direkt aufgerufen werden können.

### 12.1 Adressennormierer

Dies ist ein Entwicklungszweig des **PAGC Adressennormierers** (der Code für diesen Teilbereich beruht auf dem **PAGC Adressennormierer für PostgreSQL**).

Der Adressennormierer ist ein Parser für einzeilige Adressen. Eine gegebene Adresse wird anhand von in einer Tabelle abgelegten Regeln und den Hilfstabellen "lex" und "gaz" normiert.

Der Code befindet sich in einer einzelnen PostgreSQL Erweiterungsbibliothek mit der Bezeichnung `address_standardizer` und kann mittels `CREATE EXTENSION address_standardizer;` installiert werden. Zusätzlich zu der Erweiterung "address\_standardizer" gibt es auch die Erweiterung `address_standardizer_data_us`, welche die Tabellen "gaz", "lex" und "rules" für Daten der USA enthält. Diese Erweiterung kann mittels `CREATE EXTENSION address_standardizer_data_us;` installiert werden.

Der Code für diese Erweiterung befindet sich unter PostGIS in `extensions/address_standardizer` und ist zurzeit self-contained ("unabhängig").

Für eine Installationsanleitung siehe: Section 2.3.

#### 12.1.1 Funktionsweise des Parsers

Der Parser arbeitet von rechts nach links und betrachtet zunächst die Makroelemente Postleitzahl, Staat/Provinz, Stadt. Anschließend werden die Mikroelemente untersucht, um festzustellen ob es sich um eine Hausnummer, eine Kreuzung oder eine Wegmarkierung handelt. Zur Zeit schaut der Parser nicht auf die Landeskenntzahl oder -namen, dies kann aber möglicherweise noch implementiert werden.

**Country code** Wird als US oder CA basiert angenommen: Postleitzahl als US oder Kanada, state/province als US oder Kanada, sonst US

**Postcode/zipcode** Diese werden über Perl-kompatible reguläre Ausdrücke erkannt. Die Regexp befinden sich in "parseaddress-api.c" und können bei Bedarf relativ leicht angepasst werden.

**State/province** Diese werden über Perl-kompatible reguläre Ausdrücke erkannt. Die Regexp befinden sich zurzeit in "parseaddress-api.c", könnten zukünftig aber zwecks leichter Wartbarkeit in die "includes" verschoben werden.

## 12.1.2 Adressennormierer Datentypen

### 12.1.2.1 stdaddr

`stdaddr` — Ein zusammengesetzter Datentyp, der aus den Elementen einer Adresse besteht. Dies ist der zurückgegebene Datentyp der `standardize_address` Funktion.

#### Beschreibung

Ein zusammengesetzter Datentyp, der aus den Elementen einer Adresse besteht. Dies ist der Datentyp, der von `standardize_address` zurückgegeben wird. Einige Elementbeschreibungen wurden von **PAGC Postal Attributes** übernommen.

Die Token-Nummern geben die Referenznummer der Ausgabe in der **rules table** an.



Diese Methode benötigt die Erweiterung `address_standardizer`.

**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 folgt. Beispiel *WEST* in *3715 TENTH AVENUE WEST*.

**ruralroute** ist ein Text (Token-Nummer 8): RURAL ROUTE . Beispiel: *7* in *RR 7*.

**extra** ist ein Text: Zusätzliche Information, wie die Geschosnummer/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*.

## 12.1.3 Adressennormierer Tabellen

### 12.1.3.1 rules table

`rules table` — Die Tabelle "rules" enthält die Regeln, nach denen die Token der Eingabesequenz der Adresse in eine standardisierte Ausgabesequenz abgebildet werden. Eine Regel besteht aus einem Satz Eingabetoken, gefolgt von -1 (Terminator), gefolgt von einem Satz Ausgabtoken, gefolgt von -1, gefolgt von einer Zahl zur Kennzeichnung des Regeltyps, gefolgt von der Rangordnung der Regel.

## Beschreibung

Eine "rules" Tabelle muss mindestens die folgenden Spalten aufweisen, es können aber zusätzliche Spalten für den Eigenbedarf hinzugefügt werden.

**id** Der Primärschlüssel der Tabelle

**rule** Ein Textfeld, das die Regel festlegt. Details unter [PAGC Address Standardizer Rule records](#).

Eine Regel besteht aus positiven ganzen Zahlen, den Eingabetoken, die durch ein -1 abgeschlossen werden, gefolgt von der gleichen Anzahl an positiven ganzen Zahlen, den Postattributen, die ebenfalls mit -1 abgeschlossen werden, gefolgt von einer ganzen Zahl, die den Regeltyp kennzeichnet, gefolgt von einer ganzen Zahl, welche die Rangordnung der Regel festlegt. Die Regeln werden von 0 (niedrigster Rang) bis 17 (höchster) gereiht.

So wird zum Beispiel durch die Regel 2 0 2 22 3 -1 5 5 6 7 3 -1 2 6 die Abfolge von Ausgabemarkern *TYPE NUMBER TYPE DIRECT QUALIF* auf die Ausgabeabfolge *STREET STREET SUFTYP SUFDIR QUALIF* abgebildet. Dies ist eine ARC\_C Regel vom Rang 6.

Die Nummern der entsprechenden Ausgabe-Token sind unter [stdaddr](#) aufgeführt.

## Eingabe-Token

Jede Regel beginnt mit einer Menge an Eingabetoken, gefolgt bei der Abschlussanweisung -1. Im Folgenden ein Auszug von gültigen Eingabetoken aus [PAGC Input Tokens](#):

### Formbasierte Eingabezeichen

**AMPERS** (13). Das kaufmännische Und (&) 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.

### Funktionsbasierte Eingabezeichen

**BOXH** (14). Ein Text zur Kennzeichnung von Postfächern. Zum Beispiel *Box* oder *PO Box*.

**BUILDH** (19). Wörter zur Bezeichnung von Gebäuden und Gebäudekomplexen - üblicherweise als Präfix. Zum Beispiel: *Tower* in *Tower 7A*.

**BUILDT** (24). Wörter und Abkürzungen zur Bezeichnung von Gebäuden und Gebäudekomplexen - üblicherweise als Suffix. Zum Beispiel: *Shopping Centre*.

**DIRECT** (22). Text zur Richtungsangabe, zum Beispiel *North*.

**MILE** (20). Wörter zur Bezeichnung von Milepost Adressen.

**ROAD** (6). Wörter und Abkürzungen für die Bezeichnung von Autobahnen und Straßen. Zum Beispiel *Interstate* in *Interstate 5*.

**RR** (8). Wörter und Abkürzungen für Postwege im ländlichen Gebiet - "Rural Routes". *RR*.

**TYPE** (2). Begriffe und Abkürzungen für Straßentypen. Zum Beispiel: *ST* oder *AVE*.

**UNITH** (16). Begriffe und Abkürzungen für zusätzliche Adressangaben. Zum Beispiel *APT* oder *UNIT*.

### Eingabezeichen für den Postleitzahltyp

**QUINT** (28). Eine 5-stellige Nummer. Gibt den Zip Code an

**QUAD** (29). Eine 4-stellige Nummer. Gibt den ZIP4 Code an.

**PCH** (27). Eine 3 Zeichen lange Abfolge von Buchstabe - Zahl - Buchstabe. Kennzeichnet eine FSA, die ersten 3 Zeichen des kanadischen Postleitzahl.

**PCT** (26). Eine 3 Zeichen lange Abfolge von Zahl -Buchstabe - Zahl. Kennzeichnet eine LDU, die letzten 3 Zeichen des kanadischen Postleitzahl.

### Stoppwörter

Stoppwörter werden mit Wörtern kombiniert. In den Regeln wird eine Zeichenkette aus mehreren Wörtern und Stoppwörtern durch einen einzelnen WORD-Token dargestellt.

**STOPWORD** (7). Ein Wort mit geringer semantischer Bedeutung, das bei der Analyse weggelassen werden kann. Zum Beispiel: *THE*.

### Ausgabe-Token

Nach dem ersten -1 (Abschlussanweisung) folgen die Ausgabetoken und deren Reihenfolge, gefolgt bei einer Abschlussanweisung -1. Die Nummern der entsprechenden Ausgabetoken sind unter **stdaddr** aufgeführt. Welche Token zulässig sind hängt von der Art der Regel ab. Die gültigen Ausgabetoken für die jeweiligen Regeln sind unter the section called "**Regel Typen und Rang**" aufgelistet.

### Regel Typen und Rang

Den Schlussteil der Regel bildet der Regeltyp. Dieser wird, gefolgt von einem Rang für die Regel, durch eines der folgenden Wörter angegeben. Die Regeln sind von 0 (niedrigster Rang) bis 17 (höchster Rang) gereiht.

#### MACRO\_C

(Token-Nummer = "0"). Die Klassenregeln um MACRO Klauseln, wie *PLACE STATE ZIP*, zu parsen.

**MACRO\_C output tokens** (Auszug aus <http://www.pgcgeo.org/docs/html/pagc-12.html#-r-ty->).

**CITY** (Token-Nummer "10"). Beispiel "Albanien"

**STATE** (Token-Nummer "11"). Beispiel "NY"

**NATION** (Token Nummer "12"). Dieses Attribut wird in den meisten Referenzdateien nicht verwendet. Beispiel "USA"

**POSTAL** (Token Nummer "13"). (SADS Elemente "ZIP CODE", "PLUS 4"). Dieses Attribut wird für die Postleitzahlen-Codes der USA (ZIP-Code) und Kanada (Postal Code) verwendet.

#### MICRO\_C

(Token Nummer = "1"). Die Regelklasse zum Parsen ganzer MICRO Klauseln (wie House, street, sufdir, predir, pretyp, suftype, qualif) (insbesondere ARC\_C plus CIVIC\_C). Diese Regeln werden bei der Aufbauphase nicht benutzt.

**MICRO\_C output tokens** (Auszug aus <http://www.pgcgeo.org/docs/html/pagc-12.html#-r-ty->).

**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 Nummer = "2"). Die Regelklasse zum Parsen von MICRO Klauseln ausgenommen dem Attribut "HOUSE". Verwendet dieselben Ausgabetoken wie MICRO\_C, abzüglich dem HOUSE Token.

### **CIVIC\_C**

(Token-Nummer = "3"). Die Klassenregeln zum Parsen des HOUSE Attributs.

### **EXTRA\_C**

(token number = "4"). Die Regelklasse zum Parsen von zusätzlichen Attributen - Attribute die von der Geokodierung ausgeschlossen sind. Diese Regeln werden bei der Aufbauphase nicht benutzt.

**EXTRA\_C output tokens** (Auszug aus <http://www.pagcgeo.org/docs/html/pagc-12.html#--r-typ-->).

**BLDNG** (Token Nummer 0): Ungeparste Gebäudeidentifikatoren und Gebäudetypen.

**BOXH** (Token-Nummer 14): Die **BOX** in BOX 3B

**BOXT** (Token-Nummer 15): **3B** in BOX 3B

**RR** (Token-Nummer 8): **RR** in RR 7

**UNITH** (Token-Nummer 16): **APT** in APT 3B

**UNITT** (Token-Nummer 17): **3B** in APT 3B

**UNKNWN** (Token-Nummer 9): Eine nicht näher klassifizierte Ausgabe.

### **12.1.3.2 lex table**

lex table — Eine "lex" Tabelle wird verwendet, um eine alphanumerische Eingabe einzustufen und mit (a) Eingabe-Tokens (siehe the section called "**Eingabe-Token**") und (b) normierten Darstellungen zu verbinden.

#### **Beschreibung**

Eine lex (abgekürzt für Lexikon) Tabelle wird verwendet um alphanumerische Eingaben zu gliedern, und die Eingabe mit the section called "**Eingabe-Token**" und (b) genormten Darstellungen zu verbinden. In diesen Tabellen finden Sie Dinge wie ONE abgebildet auf stdword: 1.

Eine "lex" Tabelle muss zumindest die folgenden Spalten aufweisen.

**id** Der Primärschlüssel der Tabelle

**seq** Integer: Definitionsnummer?

**word** text: das Eingabewort

**stdword** text: das normierte Ersatzwort

**token** Integer: die Art des Wortes. Wird nur in diesem Zusammenhang ersetzt. Siehe [PAGC Tokens](#).

### 12.1.3.3 gaz table

gaz table — Eine "gaz" Tabelle wird verwendet, um Ortsnamen zu normieren und um diese mit (a) Eingabe-Token (siehe the section called “**Eingabe-Token**”) und (b) normierten Darstellungen zu verbinden.

#### Beschreibung

Eine "gaz" (Abkürzung für Gazeteer) Tabelle wird verwendet, um Ortsnamen zu normieren und um diese mit the section called “**Eingabe-Token**” und (b) normierten Darstellungen zu verbinden. Wenn Sie zum Beispiel in der USA sind, können Sie die Namen der Bundesstaaten und die zugehörigen Abkürzungen in diese Tabelle laden.

Eine "gaz" Tabelle muss zumindest die folgenden Spalten aufweisen, es können aber zusätzliche Spalten für den Eigenbedarf hinzugefügt werden.

**id** Der Primärschlüssel der Tabelle

**seq** Integer: Kennzahl? - Kennung die für diese Instanz des Wortes verwendet wird.

**word** text: das Eingabewort

**stdword** text: das normierte Ersatzwort

**token** Integer: die Art des Wortes. Wird nur in diesem Zusammenhang ersetzt. Siehe **PAGC Tokens**.

## 12.1.4 Adressennormierer Funktionen

### 12.1.4.1 debug\_standardize\_address

debug\_standardize\_address — Gibt einen json-formatierten Text zurück, der die Parse-Token und Standardisierungen auflistet

#### Synopsis

text **debug\_standardize\_address**(text lextab, text gaztab, text rultab, text micro, text macro=NULL);

#### Beschreibung

Dies ist eine Funktion zum Debuggen von Adresstandardisierungsregeln und lex/gaz-Zuordnungen. Sie gibt einen json-formatierten Text zurück, der die Abgleichsregeln, die Zuordnung von Token und die beste standardisierte Adresse **stdaddr** Form einer Eingabeadresse unter Verwendung von **lex table** Tabellennamen, **gaz table** und **rules table** Tabellennamen und einer Adresse enthält.

Für einzeilige Adressen verwenden Sie einfach **micro**

Für eine zweizeilige Adresse **A micro** bestehend aus der ersten Zeile einer Standard-Postanschrift, z. B. **house\_num street**, und einem Makro bestehend aus der zweiten Zeile einer Standard-Postanschrift, z. B. **city, state postal\_code country**.

Die im json-Dokument zurückgegebenen Elemente sind

**input\_tokens** Gibt für jedes Wort in der Eingabeadresse die Position des Wortes, die Token-Kategorisierung des Wortes und das Standardwort, dem es zugeordnet ist, zurück. Beachten Sie, dass Sie für einige Eingabewörter möglicherweise mehrere Datensätze zurückerhalten, da einige Eingaben als mehr als eine Sache kategorisiert werden können.

**rules** Die Menge der Regeln, die der Eingabe entsprechen, und die entsprechende Punktzahl für jede Regel. Die erste Regel (mit der höchsten Punktzahl) wird für die Standardisierung verwendet

**stdaddr** Die standardisierten Adresselemente **stdaddr**, die bei der Ausführung von **standardize\_address**

Verfügbarkeit: 3.4.0



Diese Methode benötigt die Erweiterung **address\_standardizer**.

## Beispiele

### Verwendung der address\_standardizer\_data\_us Erweiterung

```
CREATE EXTENSION address_standardizer_data_us; -- only needs to be done once
```

#### Variante 1: Einzeilige Adresse und Rückgabe der Eingabe-Token

```
SELECT it->>'pos' AS position, it->>'word' AS word, it->>'stdword' AS standardized_word,
       it->>'token' AS token, it->>'token-code' AS token_code
FROM jsonb(
  debug_standardize_address('us_lex',
    'us_gaz', 'us_rules', 'One Devonshire Place, PH 301, Boston, MA 02109')
  ) AS s, jsonb_array_elements(s->'input_tokens') AS it;
```

position	word	standardized_word	token	token_code
0	ONE	1	NUMBER	0
0	ONE	1	WORD	1
1	DEVONSHIRE	DEVONSHIRE	WORD	1
2	PLACE	PLACE	TYPE	2
3	PH	PATH	TYPE	2
3	PH	PENTHOUSE	UNITT	17
4	301	301	NUMBER	0

(7 rows)

#### Variante 2: Mehrzeilige Adresse und Rückgabe der ersten Regeleingabe-Mappings und der Punktzahl

```
SELECT (s->'rules'->0->>'score')::numeric AS score, it->>'pos' AS position,
       it->>'input-word' AS word, it->>'input-token' AS input_token, it->>'mapped-word' AS ↵
       standardized_word,
       it->>'output-token' AS output_token
FROM jsonb(
  debug_standardize_address('us_lex',
    'us_gaz', 'us_rules', 'One Devonshire Place, PH 301', 'Boston, MA 02109')
  ) AS s, jsonb_array_elements(s->'rules'->0->'rule_tokens') AS it;
```

score	position	word	input_token	standardized_word	output_token
0.876250	0	ONE	NUMBER	1	HOUSE
0.876250	1	DEVONSHIRE	WORD	DEVONSHIRE	STREET
0.876250	2	PLACE	TYPE	PLACE	SUFTYP
0.876250	3	PH	UNITT	PENTHOUSE	UNITT
0.876250	4	301	NUMBER	301	UNITT

(5 rows)

### Siehe auch

[stdaddr](#), [rules table](#), [lex table](#), [gaz table](#), [Pagc\\_Normalize\\_Address](#)

#### 12.1.4.2 parse\_address

parse\_address — Nimmt eine 1-zeilige Adresse entgegen und zerlegt sie in die Einzelteile

### Synopsis

```
record parse_address(text address);
```

## Beschreibung

Nimmt eine Adresse entgegen und gibt einen Datensatz mit den folgenden Attributen zurück: *num*, *street*, *street2*, *address1*, *city*, *state*, *zip*, *zipplus* und *country*.

Verfügbarkeit: 2.2.0



Diese Methode benötigt die Erweiterung `address_standardizer`.

## 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

### 12.1.4.3 standardize\_address

`standardize_address` — Gibt eine gegebene Adresse in der Form "stdaddr" zurück. Verwendet die Tabellen "lex", "gaz" und "rule".

## Synopsis

```
stdaddr standardize_address(text lextab, text gaztab, text rultab, text address);
stdaddr standardize_address(text lextab, text gaztab, text rultab, text micro, text macro);
```

## Beschreibung

Gibt eine gegebene Adresse in der Form `stdaddr` zurück. Verwendet die Tabellennamen `lex table`, `gaz table` und `rules table` und eine Adresse.

Variante 1: Nimmt eine einzeilige Adresse entgegen.

Variante 2: Nimmt eine Adresse in 2 Teilen entgegen. Ein `micro` Teil, der aus der normierten ersten Zeile einer Postadresse besteht; z.B. `house_num street`. Ein "macro"-Teil, der aus der normierten zweiten Zeile einer Adresse besteht; z.B. `city, state postal_code country`.

Verfügbarkeit: 2.2.0



Diese Methode benötigt die Erweiterung `address_standardizer`.

## Beispiele

Verwendung der `address_standardizer_data_us` Erweiterung

```
CREATE EXTENSION address_standardizer_data_us; -- only needs to be done once
```

Variante 1: Einzeilige Adresse. Dies funktioniert nicht gut mit Adressen außerhalb der US

```
SELECT house_num, name, suftype, city, country, state, unit FROM standardize_address(' ←
    us_lex',
                                           'us_gaz', 'us_rules', 'One Devonshire Place, PH 301, Boston, MA ←
                                           02109');
```

house_num	name	suftype	city	country	state	unit
1	DEVONSHIRE	PLACE	BOSTON	USA	MASSACHUSETTS	# PENTHOUSE 301

Verwendung der Tabellen, die mit dem Tiger Geokodierer paketiert sind. Dieses Beispiel funktioniert nur, wenn Sie `postgis_tiger` installiert haben.

```
SELECT * FROM standardize_address('tiger.pagc_lex',
    'tiger.pagc_gaz', 'tiger.pagc_rules', 'One Devonshire Place, PH 301, Boston, MA ←
    02109-1234');
```

Die Ausgabe über einen Dump mit der Erweiterung "hstore" ist leichter lesbar. Die Erweiterung `hstore` muss mittels "CREATE EXTENSION hstore;" installiert sein.

```
SELECT (each(hstore(p))).*
FROM standardize_address('tiger.pagc_lex', 'tiger.pagc_gaz',
    'tiger.pagc_rules', 'One Devonshire Place, PH 301, Boston, MA 02109') As p;
```

key	value
box	
city	BOSTON
name	DEVONSHIRE
qual	
unit	# PENTHOUSE 301
extra	
state	MA
predir	
sufdir	
country	USA
pretype	
suftype	PL
building	

```

postcode | 02109
house_num | 1
ruralroute |
(16 rows)

```

### Variante 2: Adresse aus zwei Teilen.

```

SELECT (each(hstore(p))).*
FROM standardize_address('tiger.pagc_lex', 'tiger.pagc_gaz',
    'tiger.pagc_rules', 'One Devonshire Place, PH 301', 'Boston, MA 02109, US') As p;

```

```

key      | value
-----+-----
box      |
city     | BOSTON
name     | DEVONSHIRE
qual     |
unit     | # PENTHOUSE 301
extra    |
state    | MA
preidir  |
sufdir   |
country  | USA
pretype  |
suftype  | PL
building |
postcode | 02109
house_num | 1
ruralroute |
(16 rows)

```

### Siehe auch

[stdaddr](#), [rules table](#), [lex table](#), [gaz table](#), [Pagc\\_Normalize\\_Address](#)

## 12.2 Tiger Geokoder

Es existieren eine Reihe weiterer Open Source Geokodierer für PostGIS, welche im Gegensatz zu dem Tiger Geokodierer den Vorteil haben, dass sie mehrere Länder unterstützen

- **Nominatim** verwendet Daten von OpenStreetMap, die mittels Gazetteer formatiert werden. Es benötigt osm2pgsql zum Laden der Daten, PostgreSQL 8.4+ und PostGIS 1.5+ um zu funktionieren. Es ist als Webinterface paketiert und wird vermutlich als Webservice aufgerufen. So wie der Tiger Geokodierer besteht es aus einem Geokodierer und einer inversen Komponente des Geokodierers. Aus der Dokumentation ist nicht ersichtlich, ob es wie der Tiger Geokodierer auch eine reine SQL Schnittstelle aufweist, oder ob ein größerer Anteil der Logik in das Webinterface implementiert wurde.
- **GIS Graphy** nützt ebenfalls PostGIS und arbeitet so wie Nominatim ebenfalls mit Daten von OpenStreetMap (OSM). Es beinhaltet einen Loader, um OSM-Daten zu importieren. Ähnlich wie Nominatim kann es auch für die Geokodierung außerhalb der USA verwendet werden. So wie Nominatim läuft es als Webservice und benötigt Java 1.5, Servlet Apps und Solr. GisGraphy kann plattformübergreifend genutzt werden und hat ebenfalls einen invertierten Geokodierer zusammen mit anderen geschickten Funktionen.

### 12.2.1 Drop\_Indexes\_Generate\_Script

`Drop_Indexes_Generate_Script` — Erzeugt ein Skript, welches alle Indizes aus dem Datenbankschema "Tiger" oder aus einem vom Anwender angegebenen Schema löscht, wenn die Indizes nicht auf den Primärschlüssel gelegt und nicht "unique" sind. Wenn kein Schema angegeben ist wird standardmäßig auf das `tiger_data` Schema zugegriffen.

## Synopsis

```
text Drop_Indexes_Generate_Script(text param_schema=tiger_data);
```

## Beschreibung

Erzeugt ein Skript, welches alle Indizes aus dem Datenbankschema "Tiger" oder aus einem vom Anwender angegebenen Schema löscht, wenn die Indizes nicht auf den Primärschlüssel gelegt und nicht "unique" sind. Wenn kein Schema angegeben ist wird standardmäßig auf das `tiger_data` Schema zugegriffen.

Dies kann verwendet werden, damit sich die Indizes nicht aufblähen und dadurch den Anfrageoptimierer irritieren oder unnötigen Speicherplatz belegen. Sie können das Skript in Verbindung mit [Install\\_Missing\\_Indexes](#) verwenden um nur jene Indizes zu erstellen die der Gekodierer benötigt.

Verfügbarkeit: 2.0.0

## Beispiele

```
SELECT drop_indexes_generate_script() As actionsql;
actionsql
-----
DROP INDEX tiger.idx_tiger_countysub_lookup_lower_name;
DROP INDEX tiger.idx_tiger_edges_countyfp;
DROP INDEX tiger.idx_tiger_faces_countyfp;
DROP INDEX tiger.tiger_place_the_geom_gist;
DROP INDEX tiger.tiger_edges_the_geom_gist;
DROP INDEX tiger.tiger_state_the_geom_gist;
DROP INDEX tiger.idx_tiger_addr_least_address;
DROP INDEX tiger.idx_tiger_addr_tlid;
DROP INDEX tiger.idx_tiger_addr_zip;
DROP INDEX tiger.idx_tiger_county_countyfp;
DROP INDEX tiger.idx_tiger_county_lookup_lower_name;
DROP INDEX tiger.idx_tiger_county_lookup_snd_name;
DROP INDEX tiger.idx_tiger_county_lower_name;
DROP INDEX tiger.idx_tiger_county_snd_name;
DROP INDEX tiger.idx_tiger_county_the_geom_gist;
DROP INDEX tiger.idx_tiger_countysub_lookup_snd_name;
DROP INDEX tiger.idx_tiger_cousub_countyfp;
DROP INDEX tiger.idx_tiger_cousub_cousubfp;
DROP INDEX tiger.idx_tiger_cousub_lower_name;
DROP INDEX tiger.idx_tiger_cousub_snd_name;
DROP INDEX tiger.idx_tiger_cousub_the_geom_gist;
DROP INDEX tiger_data.idx_tiger_data_ma_addr_least_address;
DROP INDEX tiger_data.idx_tiger_data_ma_addr_tlid;
DROP INDEX tiger_data.idx_tiger_data_ma_addr_zip;
DROP INDEX tiger_data.idx_tiger_data_ma_county_countyfp;
DROP INDEX tiger_data.idx_tiger_data_ma_county_lookup_lower_name;
DROP INDEX tiger_data.idx_tiger_data_ma_county_lookup_snd_name;
DROP INDEX tiger_data.idx_tiger_data_ma_county_lower_name;
DROP INDEX tiger_data.idx_tiger_data_ma_county_snd_name;
:
:
```

## Siehe auch

[Install\\_Missing\\_Indexes](#), [Missing\\_Indexes\\_Generate\\_Script](#)

### 12.2.2 Drop\_Nation\_Tables\_Generate\_Script

`Drop_Nation_Tables_Generate_Script` — Erzeugt ein Skript, welches alle Tabellen in dem angegebenen Schema löscht, die mit `county_all`, `state_all` oder dem Ländercode gefolgt von `county` oder `state` beginnen.

#### Synopsis

```
text Drop_Nation_Tables_Generate_Script(text param_schema=tiger_data);
```

#### Beschreibung

Erzeugt ein Skript, welches alle Tabellen in dem angegebenen Schema löscht, die mit `county_all`, `state_all` oder dem Ländercode gefolgt von `county` oder `state` beginnen. Dies ist dann notwendig, wenn Sie von `tiger_2010` auf `tiger_2011` Daten upgraden.

Verfügbarkeit: 2.1.0

#### Beispiele

```
SELECT drop_nation_tables_generate_script();  
DROP TABLE tiger_data.county_all;  
DROP TABLE tiger_data.county_all_lookup;  
DROP TABLE tiger_data.state_all;  
DROP TABLE tiger_data.ma_county;  
DROP TABLE tiger_data.ma_state;
```

#### Siehe auch

[Loader\\_Generate\\_Nation\\_Script](#)

### 12.2.3 Drop\_State\_Tables\_Generate\_Script

`Drop_State_Tables_Generate_Script` — Erzeugt ein Skript, dass alle Tabellen in dem angegebenen Schema löscht, die als Präfix einen Ländercode haben. Wenn kein Schema angegeben ist wird standardmäßig auf das `tiger_data` Schema zugegriffen.

#### Synopsis

```
text Drop_State_Tables_Generate_Script(text param_state, text param_schema=tiger_data);
```

#### Beschreibung

Erzeugt ein Skript, dass alle Tabellen in dem angegebenen Schema löscht, die als Präfix einen Ländercode haben. Wenn kein Schema angegeben ist wird standardmäßig auf das `tiger_data` Schema zugegriffen. Wenn beim Import etwas schiefgegangen ist, können mit dieser Funktion die Tabellen eines Staates unmittelbar vor dem erneuten Import, gelöscht werden.

Verfügbarkeit: 2.0.0



## Beispiele

```
SELECT drop_state_tables_generate_script('PA');
DROP TABLE tiger_data.pa_addr;
DROP TABLE tiger_data.pa_county;
DROP TABLE tiger_data.pa_county_lookup;
DROP TABLE tiger_data.pa_cousub;
DROP TABLE tiger_data.pa_edges;
DROP TABLE tiger_data.pa_faces;
DROP TABLE tiger_data.pa_featnames;
DROP TABLE tiger_data.pa_place;
DROP TABLE tiger_data.pa_state;
DROP TABLE tiger_data.pa_zip_lookup_base;
DROP TABLE tiger_data.pa_zip_state;
DROP TABLE tiger_data.pa_zip_state_loc;
```

## Siehe auch

[Loader\\_Generate\\_Script](#)

### 12.2.4 Geocode

Geocode — Nimmt eine Adresse als Zeichenkette (oder eine bereits standardisierte Adresse) entgegen und gibt die möglichen Punktlagen zurück. Die Ausgabe beinhaltet eine Punktgeometrie in NAD 83 Länge/Breite, eine standardisierte Adresse und eine Rangfolge (Rating) für jede Punktlage. Umso niedriger die Rangfolge ist, um so wahrscheinlicher ist die Übereinstimmung. Die Ergebnisse werden mit aufsteigender Rangfolge sortiert - dar niedrigste Rang zuerst. Optional kann die maximale Anzahl der Ergebnisse angegeben werden (Standardeinstellung ist 10) und der Bereich mit `restrict_region` beschränkt werden (Standardeinstellung ist NULL)

## Synopsis

```
setof record geocode(varchar address, integer max_results=10, geometry restrict_region=NULL, norm_addy OUT addy, geometry OUT geomout, integer OUT rating);
setof record geocode(norm_addy in_addy, integer max_results=10, geometry restrict_region=NULL, norm_addy OUT addy, geometry OUT geomout, integer OUT rating);
```

## Beschreibung

Nimmt eine Adresse als Zeichenkette (oder eine bereits standardisierte Adresse) entgegen und gibt die möglichen Punktlagen zurück. Die Ausgabe beinhaltet eine Punktgeometrie in NAD 83 Länge/Breite, eine standardisierte Adresse `normalized_address` (`addy`) und eine Rangfolge (Rating) für jede Punktlage. Umso niedriger die Rangfolge ist, um so wahrscheinlicher ist die Übereinstimmung. Die Ergebnisse werden nach dem Rating aufsteigend sortiert - das niedrigste Rating zuerst. Verwendet Tiger Daten (Kanten, Maschen, Adressen), Fuzzy String Matching (`soundex`, `levenshtein`) von PostgreSQL und PostGIS Funktionen zur Interpolation entlang von Linien, um die Adressen entlang der Kanten von TIGER zu interpolieren. Umso höher das Rating, umso unwahrscheinlicher ist es, dass die Geokodierung richtig liegt. Der geokodierte Punkt wird dort, wo sich die Adresse befindet, standardmäßig um 10 Meter von der Mittellinie auf die Seite (L/R) versetzt. Optional kann die maximale Anzahl der Ergebnisse angegeben werden (Standardeinstellung ist 10) und der Bereich mit `restrict_region` beschränkt werden (Standardeinstellung ist NULL)

Erweiterung: 2.0.0 Unterstützung von strukturierten Daten von TIGER 2010. Weiters wurde die Logik überarbeitet, um die Rechengeschwindigkeit und die Genauigkeit der Geokodierung zu erhöhen, und den Versatz von der Mittellinie auf die Straßenseite zu ermöglichen. Der neue Parameter `max_results` kann verwendet werden, um die Anzahl der besten Ergebnisse zu beschränken oder um nur das beste Ergebnis zu erhalten.

## Beispiele: Grundlagen

Die Zeitangaben für die unteren Beispiele beziehen sich auf einen 3.0 GHZ Prozessor mit Windows 7, 2GB RAM, PostgreSQL 9.1rc1/PostGIS 2.0 und den geladenen TIGER-Daten der Staaten MA, MN, CA und RI.

Genauere Übereinstimmungen haben eine kürzere Rechenzeit (61ms)

```
SELECT g.rating, ST_X(g.geomout) As lon, ST_Y(g.geomout) As lat,
       (addy).address As stno, (addy).streetname As street,
       (addy).streettypeabbrev As styp, (addy).location As city, (addy).stateabbrev As st, ( ←
       addy).zip
FROM geocode('75 State Street, Boston MA 02109', 1) As g;
rating |          lon          |          lat          | stno | street | styp | city  | st | zip
-----+-----+-----+-----+-----+-----+-----+-----+-----
      0 | -71.0557505845646 | 42.35897920691 |   75 | State  | St   | Boston | MA | 02109
```

Sogar wenn der Zip-Code nicht übergeben wird, kann ihn der Geokodierer erraten (dauerte ca. 122-150ms)

```
SELECT g.rating, ST_AsText(ST_SnapToGrid(g.geomout,0.00001)) As 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
```

Kann Rechtschreibfehler behandeln und liefert mehr mögliche Lösungen mit Einstufungen, hat allerdings eine längere Laufzeit (500ms).

```
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('31 - 37 Stewart Street, Boston, MA 02116',1) As g;
rating |          wktmlonlat          | stno | street | styp | city  | st | zip
-----+-----+-----+-----+-----+-----+-----+-----
      70 | POINT(-71.06466 42.35114) |    31 | Stuart | St   | Boston | MA | 02116
```

Verwendet um die Adresskodierung in enier Stapelverarbeitung auszuführen. Am einfachsten ist es `max_results=1` zu setzen. Berechnet nur die Fälle, die noch nicht geokodiert wurden (keine Einstufung haben).

```
CREATE TABLE addresses_to_geocode(addid serial PRIMARY KEY, address text,
                                   lon numeric, lat numeric, new_address text, rating integer);

INSERT INTO addresses_to_geocode(address)
VALUES ('529 Main Street, Boston MA, 02129'),
       ('77 Massachusetts Avenue, Cambridge, MA 02139'),
       ('25 Wizard of Oz, Walaford, KS 99912323'),
       ('26 Capen Street, Medford, MA'),
       ('124 Mount Auburn St, Cambridge, Massachusetts 02138'),
       ('950 Main Street, Worcester, MA 01610');

-- only update the first 3 addresses (323-704 ms - there are caching and shared memory ←
-- effects so first geocode you do is always slower) --
-- for large numbers of addresses you don't want to update all at once
-- since the whole geocode must commit at once
-- For this example we rejoin with LEFT JOIN
-- and set to rating to -1 rating if no match
-- to ensure we don't regeocode a bad address
UPDATE addresses_to_geocode
SET (rating, new_address, lon, lat)
```

```

= ( COALESCE(g.rating,-1), pprint_addy(g.addy),
    ST_X(g.geomout)::numeric(8,5), ST_Y(g.geomout)::numeric(8,5) )
FROM (SELECT addid, address
      FROM addresses_to_geocode
      WHERE rating IS NULL ORDER BY addid LIMIT 3) As a
LEFT JOIN LATERAL geocode(a.address,1) As g ON true
WHERE a.addid = addresses_to_geocode.addid;

```

result  
-----

Query returned successfully: 3 rows affected, 480 ms execution time.

```
SELECT * FROM addresses_to_geocode WHERE rating is not null;
```

addid	address new_address	lon	lat	↔
1	529 Main Street, Boston MA, 02129 Boston, MA 02129	-71.07177	42.38357	529 Main St, ↔
2	77 Massachusetts Avenue, Cambridge, MA 02139 Massachusetts Ave, Cambridge, MA 02139	-71.09396	42.35961	77 ↔
3	25 Wizard of Oz, Walaford, KS 99912323 KS 67502	-97.92913	38.12717	Willowbrook, ↔

(3 rows)

**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
7	POINT(-70.96796 42.4659)	100	Federal	St	Lynn	MA	01905
16	POINT(-70.96786 42.46853)	NULL	Federal	St	Lynn	MA	01905

(2 rows)

Time: 622.939 ms

**Siehe auch**

[Normalize\\_Address](#), [Pprint\\_Addy](#), [ST\\_AsText](#), [ST\\_SnapToGrid](#), [ST\\_X](#), [ST\\_Y](#)

**12.2.5 Geocode\_Intersection**

Geocode\_Intersection — Nimmt 2 sich kreuzende Straßen, einen Bundesstaat, eine Stadt und einen ZIP-Code entgegen und gibt die möglichen Punktlagen an der ersten Querstraße an der Kreuzung zurück. Die Ausgabe beinhaltet auch die Geometrie "geomout" in NAD 83 Länge/Breite, eine standardisierte Adresse `normalized_address` (`addy`) für jede Punktage, sowie die Rangfolge. Umso niedriger die Rangfolge ist, um so wahrscheinlicher ist die Übereinstimmung. Die Ergebnisse werden mit

aufsteigender Rangfolge sortiert - dar niedrigste Rang zuerst. Optional kann die maximale Anzahl der Ergebnisse angegeben werden (Standardeinstellung ist 10). Verwendet TIGER Daten (Kanten, Maschen, Adressen) und Fuzzy String Matching (soundex, levenshtein) von PostgreSQL.

### Synopsis

setof record **geocode\_intersection**(text roadway1, text roadway2, text in\_state, text in\_city, text in\_zip, integer max\_results=10, norm\_addy OUT addy, geometry OUT geomout, integer OUT rating);

### Beschreibung

Nimmt 2 sich kreuzende Straßen, einen Bundesstaat, eine Stadt und einen ZIP-Code entgegen und gibt die möglichen Punktlagen an der ersten Querstraße bei der Kreuzung zurück. Die Ausgabe beinhaltet auch eine Punktgeometrie in NAD 83 Länge/Breite, eine standardisierte Adresse für jede Punktage, sowie die Rangfolge. Umso niedriger die Rangfolge ist, um so wahrscheinlicher ist die Übereinstimmung. Die Ergebnisse werden mit aufsteigender Rangfolge sortiert - dar niedrigste Rang zuerst. Optional kann die maximale Anzahl der Ergebnisse angegeben werden (Standardeinstellung ist 10). Gibt für jede Punktage die standardisierte Adresse `normalized_address` (`addy`), die Punktgeometrie "geomout" in NAD 83 Länge/Breite und ein Rating zurück. Verwendet TIGER Daten (Kanten, Maschen, Adressen) und Fuzzy String Matching (soundex, levenshtein) von PostgreSQL.

Verfügbarkeit: 2.0.0

### Beispiele: Grundlagen

Die Zeitangaben für die unteren Beispiele beziehen sich auf einen 3.0 GHZ Prozessor mit Windows 7, 2GB RAM, PostgreSQL 9.0/PostGIS 1.5 und den geladenen TIGER-Daten des Staates MA. Zurzeit ein bißchen langsam (3000ms)

Testlauf auf Windows 2003 64-bit 8GB mit PostGIS 2.0, PostgreSQL 64-bit und geladenen TIGER-Daten von 2011 -- (41ms)

```
SELECT pprint_addy(addy), st_astext(geomout), rating
      FROM geocode_intersection('Haverford St', 'Germania St', 'MA', 'Boston', '02130', 1);
```

pprint_addy	st_astext	rating
98 Haverford St, Boston, MA 02130	POINT(-71.101375 42.31376)	0

Sogar wenn der Zip-Code nicht angegeben ist, kann der Geokodierer diesen erraten (benötigte 3500ms auf einem Windows 7 Rechner, 741 ms auf Windows 2003 64-bit)

```
SELECT pprint_addy(addy), st_astext(geomout), rating
      FROM geocode_intersection('Weld', 'School', 'MA', 'Boston');
```

pprint_addy	st_astext	rating
98 Weld Ave, Boston, MA 02119	POINT(-71.099 42.314234)	3
99 Weld Ave, Boston, MA 02119	POINT(-71.099 42.314234)	3

### Siehe auch

[Geocode](#), [Pprint\\_Addy](#), [ST\\_AsText](#)

## 12.2.6 Get\_Geocode\_Setting

`Get_Geocode_Setting` — Gibt die in der Tabelle "tiger.geocode\_settings" gespeicherten Einstellungen zurück.

### Synopsis

text **Get\_Geocode\_Setting**(text setting\_name);

**Beschreibung**

Gibt die in der Tabelle "tiger.geocode\_settings" gespeicherten Einstellungen zurück. Die Einstellungen erlauben auf "debugging" der Funktionen umzuschalten. Für später ist geplant auch die Ratings über die Einstellungen zu kontrollieren. Die aktuellen Einstellungen sind wie folgt:

name	setting	unit	category	↔	short_desc
debug_geocode_address	false		boolean	debug	outputs debug information in notice log such as queries when geocode_address is called if true
debug_geocode_intersection	false		boolean	debug	outputs debug information in notice log such as queries when geocode_intersection is called if true
debug_normalize_address	false		boolean	debug	outputs debug information in notice log such as queries and intermediate expressions when normalize_address is called if true
debug_reverse_geocode	false		boolean	debug	if true, outputs debug information in notice log such as queries and intermediate expressions when reverse_geocode
reverse_geocode_numbered_roads_highways,	0		integer	rating	For state and county 1 - prefer the numbered highway name, 2 - prefer local state/county name
use_pagc_address_parser	false		boolean	normalize	If set to true, will try to use the address_standardizer extension (via pagc_normalize_address) instead of tiger normalize_address built one

Änderung: 2.2.0 : die Standardeinstellungen befinden sich nun in der Tabelle "geocode\_settings\_default". Die vom Anwender angepassten Einstellungen - und nur diese - befinden sich in der Tabelle "geocode\_settings".

Verfügbarkeit: 2.1.0

**Das Beispiel gibt die "debugging" Einstellungen aus**

```
SELECT get_geocode_setting('debug_geocode_address) As result;
result
-----
false
```

**Siehe auch**

[Set\\_Geocode\\_Setting](#)

**12.2.7 Get\_Tract**

Get\_Tract — Gibt für die Lage einer Geometrie die Census Area oder ein Feld der tract-Tabelle zurück. Standardmäßig wird die Kurzbezeichnung der Census Area ausgegeben.

**Synopsis**

```
text get_tract(geometry loc_geom, text output_field=name);
```

## Beschreibung

Für eine gegebene Geometrie wird der Zählsprengel zurückgeben, in dem sich die Geometrie befindet. Wenn das Koordinatenreferenzsystem unbestimmt ist, dann wird NAD 83 in Länge und Breite angenommen.

### Note

Diese Funktion verwendet den Census `tract`, welcher standardmäßig nicht geladen wird. Wenn Sie bereits die Tabelle mit den Bundesstaaten geladen haben, können Sie mit dem Skript [Loader\\_Generate\\_Census\\_Script](#) sowohl "tract" als auch "bg" und "tabblock" laden.



Wenn Sie die Daten der Bundesstaaten noch nicht geladen haben, können Sie diese zusätzlichen Tabellen wie folgt laden

```
UPDATE tiger.loader_lookuptables SET load = true WHERE load = false AND lookup_name <->
    IN('tract', 'bg', 'tabblock');
```

dann werden diese in dem [Loader\\_Generate\\_Script](#) mit einbezogen.

Verfügbarkeit: 2.0.0

## Beispiele: Grundlagen

```
SELECT get_tract(ST_Point(-71.101375, 42.31376) ) As tract_name;
tract_name
-----
1203.01
```

```
--this one returns the tiger geoid
SELECT get_tract(ST_Point(-71.101375, 42.31376), 'tract_id' ) As tract_id;
tract_id
-----
25025120301
```

## Siehe auch

[Geocode](#)>

## 12.2.8 Install\_Missing\_Indexes

`Install_Missing_Indexes` — Findet alle Tabellen mit Schlüsselspalten, die für JOINS und Filterbedingungen vom Geokodierer verwendet werden und keinen Index aufweisen; die fehlenden Indizes werden hinzugefügt.

## Synopsis

```
boolean Install_Missing_Indexes();
```

## Beschreibung

Findet alle Tabellen in den Schemata `tiger` und `tiger_data`, bei denen die Schlüsselspalten, die für Joins und Filter vom Geokodierer verwendet werden keine Indizes aufweisen. Weiters wird ein SQL-DDL Skript ausgegeben, das die Indizes für diese Tabellen festlegt und anschließend ausgeführt wird. Dabei handelt es sich um eine Hilfsfunktion, die Abfragen schneller macht, indem die benötigten Indizes, die während des Imports gefehlt haben, neu hinzufügt. Diese Funktion ist verwandt mit [Missing\\_Indexes\\_Generate\\_Script](#), wobei zusätzlich zur Erstellung des "CREATE INDEX"-Skripts dieses auch ausgeführt wird. Es wird als Teil des Upgrade-Skripts `update_geocode.sql` aufgerufen.

Verfügbarkeit: 2.0.0

## Beispiele

```
SELECT install_missing_indexes();
       install_missing_indexes
-----
t
```

## Siehe auch

[Loader\\_Generate\\_Script](#), [Missing\\_Indexes\\_Generate\\_Script](#)

### 12.2.9 Loader\_Generate\_Census\_Script

`Loader_Generate_Census_Script` — Erzeugt für gegebene Plattform und Bundesstaaten ein Shellskript, das die TIGER Datentabellen "tract", "bg" und "tabblocks" herunterlädt, bereitstellt und in das Schema `tiger_data` importiert. Jedes Bundesstaat-Skript wird in einem eigenen Datensatz ausgegeben.

## Synopsis

```
setof text loader_generate_census_script(text[] param_states, text os);
```

## Beschreibung

Erzeugt für gegebene Plattform und Bundesstaaten ein Shellskript, das die TIGER Datentabellen Census Area `tract`, block groups `bg` und `tabblocks` herunterlädt, bereitstellt und in das Schema `tiger_data` importiert. Jedes Bundesstaat-Skript wird in einem eigenen Datensatz ausgegeben.

Zum Herunterladen wird auf Linux `unzip` (auf Windows standardmäßig 7-zip) und `wget` verwendet. Es verwendet Section [4.7.2](#) zum Laden der Daten. Die kleinste Einheit, die bearbeitet wird ist ein ganzer Bundesstaat. Es werden nur die Dateien in den Ordnern "staging" und "temp" bearbeitet.

Verwendet die folgenden Kontrolltabellen, um den Verarbeitungsprozess und die verschiedenen Variationen der Betriebssysteme in Bezug auf den Shellsyntax zu überprüfen.

1. `loader_variables` behält den Überblick über verschiedenen Variablen, wie Census Site, Jahr, Daten- und "staging"-Schemata
2. `loader_platform` Profile von verschiedenen Plattformen und Speicherplätze der ausführbaren Programme. Beinhaltet Windows und Linux. Weitere können hinzugefügt werden.
3. `loader_lookuptables` jeder Datensatz definiert einen bestimmten Tabellentyp (`state`, `county`), um Datensätze zu bearbeiten und zu importieren. Legt die Schritte fest, die notwendig sind, um Daten zu importieren, bereitzustellen und hinzuzufügen, und um Spalten, Indizes und Constraints zu löschen. Jede Tabelle wird mit einem Präfix des Bundesstaates versehen und erbt von einer Tabelle in dem Schema TIGER. z.B. wird die Tabelle `tiger_data.ma_faces` erstellt, welche von `tiger.faces` erbt

Verfügbarkeit: 2.0.0



### Note

`Loader_Generate_Script` beinhaltet diese Logik; wenn Sie aber den TIGER Geokodierer vor PostGIS 2.0.0 alpha5 installiert haben, müssen Sie dies für bereits importierte Bundesstaaten ausführen, um die zusätzlichen Tabellen zu erhalten.

## Beispiele

Erzeugt ein Skript um Daten für die ausgewählten Länder im Windows Shell Script Format zu laden.

```
SELECT loader_generate_census_script (ARRAY['MA'], 'windows');
-- result --
set STATEDIR="\gisdata\www2.census.gov\geo\pvs\tiger2010st\25_Massachusetts"
set TMPDIR=\gisdata\temp\
set UNZIPTOOL="C:\Program Files\7-Zip\7z.exe"
set WGETTOOL="C:\wget\wget.exe"
set PGBIN=C:\projects\pg\pg91win\bin\
set PGPORT=5432
set PGHOST=localhost
set PGUSER=postgres
set PGPASSWORD=yourpasswordhere
set PGDATABASE=tiger_postgis20
set PSQL="%PGBIN%psql"
set SHP2PGSQL="%PGBIN%shp2pgsql"
cd \gisdata

%WGETTOOL% http://www2.census.gov/geo/pvs/tiger2010st/25_Massachusetts/25/ --no-parent -- ←
  relative --accept=*bg10.zip,*tract10.zip,*tabblock10.zip --mirror --reject=html
del %TMPDIR%\*.* /Q
%PSQL% -c "DROP SCHEMA tiger_staging CASCADE;"
%PSQL% -c "CREATE SCHEMA tiger_staging;"
cd %STATEDIR%
for /r %%z in (*.zip) do %UNZIPTOOL% e %%z -o%TMPDIR%
cd %TMPDIR%
%PSQL% -c "CREATE TABLE tiger_data.MA_tract (CONSTRAINT pk_MA_tract PRIMARY KEY (tract_id) ) ←
  INHERITS (tiger.tract); "
%SHP2PGSQL% -c -s 4269 -g the_geom -W "latin1" tl_2010_25_tract10.dbf tiger_staging. ←
  ma_tract10 | %PSQL%
%PSQL% -c "ALTER TABLE tiger_staging.MA_tract10 RENAME geoid10 TO tract_id; SELECT ←
  loader_load_staged_data (lower('MA_tract10'), lower('MA_tract')); "
%PSQL% -c "CREATE INDEX tiger_data_MA_tract_the_geom_gist ON tiger_data.MA_tract USING gist ←
  (the_geom); "
%PSQL% -c "VACUUM ANALYZE tiger_data.MA_tract;"
%PSQL% -c "ALTER TABLE tiger_data.MA_tract ADD CONSTRAINT chk_statefp CHECK (statefp = ←
  '25');"
:
```

Erzeugt ein Shell-Skript

```
STATEDIR="/gisdata/www2.census.gov/geo/pvs/tiger2010st/25_Massachusetts"
TMPDIR="/gisdata/temp/"
UNZIPTOOL=unzip
WGETTOOL="/usr/bin/wget"
export PGBIN=/usr/pgsql-9.0/bin
export PGPORT=5432
export PGHOST=localhost
export PGUSER=postgres
export PGPASSWORD=yourpasswordhere
export PGDATABASE=geocoder
PSQL=${PGBIN}/psql
SHP2PGSQL=${PGBIN}/shp2pgsql
cd /gisdata

wget http://www2.census.gov/geo/pvs/tiger2010st/25_Massachusetts/25/ --no-parent --relative ←
  --accept=*bg10.zip,*tract10.zip,*tabblock10.zip --mirror --reject=html
rm -f ${TMPDIR}/*.*
${PSQL} -c "DROP SCHEMA tiger_staging CASCADE;"
${PSQL} -c "CREATE SCHEMA tiger_staging;"
```



```
cd $STATEDIR
for z in *.zip; do $UNZIPTOOL -o -d $TMPDIR $z; done
:
:
```

## Siehe auch

[Loader\\_Generate\\_Script](#)

### 12.2.10 Loader\_Generate\_Script

`Loader_Generate_Script` — Erzeugt für gegebene Plattform und Bundesstaaten ein Shellskript, das die TIGER Daten herunterlädt, bereitstellt und in das Schema `tiger_data` importiert. Jedes Bundesstaat-Skript wird in einem eigenen Datensatz ausgegeben. Die neueste Version unterstützt die geänderte Struktur von Tiger 2010 und lädt ebenfalls die Census Tract, Block Groups und Blocks Tabellen.

## Synopsis

```
setof text loader_generate_script(text[] param_states, text os);
```

## Beschreibung

Erzeugt für gegebene Plattform und Bundesstaaten ein Shellskript, das die TIGER Daten herunterlädt, bereitstellt und in das Schema `tiger_data` importiert. Jedes Bundesstaat-Skript wird in einem eigenen Datensatz ausgegeben.

Zum Herunterladen wird auf Linux `unzip` (auf Windows standardmäßig `7-zip`) und `wget` verwendet. Es verwendet Section [4.7.2](#) um die Daten zu laden. Die kleinste Einheit, die bearbeitet wird ist ein ganzer Bundesstaat. Es werden nur die Dateien in den Ordnern "staging" und "temp" bearbeitet.

Verwendet die folgenden Kontrolltabellen, um den Verarbeitungsprozess und die verschiedenen Variationen der Betriebssysteme in Bezug auf den Shellsyntax zu überprüfen.

1. `loader_variables` behält den Überblick über verschiedenen Variablen, wie Census Site, Jahr, Daten- und "staging"-Schemata
2. `loader_platform` Profile von verschiedenen Plattformen und Speicherplätze der ausführbaren Programme. Beinhaltet Windows und Linux. Weitere können hinzugefügt werden.
3. `loader_lookuptables` jeder Datensatz definiert einen bestimmten Tabellentyp (`state`, `county`), um Datensätze zu bearbeiten und zu importieren. Legt die Schritte fest, die notwendig sind, um Daten zu importieren, bereitzustellen und hinzuzufügen, und um Spalten, Indizes und Constraints zu löschen. Jede Tabelle wird mit einem Präfix des Bundesstaates versehen und erbt von einer Tabelle in dem Schema TIGER. z.B. wird die Tabelle `tiger_data.ma_faces` erstellt, welche von `tiger.faces` erbt

Verfügbarkeit: 2.0.0 unterstützt die strukturierten Daten von Tiger 2010 und ladet die Tabellen "tract" (Census Area), "bg" (Census Block Groups) und "tabblocks" (Census Blocks).



## Note

Wenn Sie `pgAdmin3` verwenden, dann seien Sie gewarnt, dass `pgAdmin3` langen Text abschneidet. Um dies zu beheben, können Sie `File -> Options -> Query Tool -> Query Editor -> Max. characters per column` auf mehr als 50000 Zeichen setzen.

## Beispiele

Verwendung von psql; die Datenbank ist "gistest" und /gisdata/data\_load.sh ist die Datei mit den Shell-Befehlen die ausgeführt werden sollen.

```
psql -U postgres -h localhost -d gistest -A -t \  
-c "SELECT Loader_Generate_Script (ARRAY['MA'], 'gistest') " > /gisdata/data_load.sh;
```

Erzeugt ein Skript, das Daten von 2 Staaten im Windows Shell Script Format ladet.

```
SELECT loader_generate_script (ARRAY['MA','RI'], 'windows') AS result;  
-- result --  
set TMPDIR=\gisdata\temp\  
set UNZIPTOOL="C:\Program Files\7-Zip\7z.exe"  
set WGETTOOL="C:\wget\wget.exe"  
set PGBIN=C:\Program Files\PostgreSQL\9.4\bin\  
set PGPORT=5432  
set PGHOST=localhost  
set PGUSER=postgres  
set PGPASSWORD=yourpasswordhere  
set PGDATABASE=geocoder  
set PSQL="%PGBIN%psql"  
set SHP2PGSQL="%PGBIN%shp2pgsql"  
cd \gisdata  
  
cd \gisdata  
%WGETTOOL% ftp://ftp2.census.gov/geo/tiger/TIGER2015/PLACE/tl*_25_* --no-parent --relative ←  
--recursive --level=2 --accept=zip --mirror --reject=html  
cd \gisdata/ftp2.census.gov/geo/tiger/TIGER2015/PLACE  
:  
:
```

Erzeugt ein Shell-Skript

```
SELECT loader_generate_script (ARRAY['MA','RI'], 'sh') AS result;  
-- result --  
TMPDIR="/gisdata/temp/"  
UNZIPTOOL=unzip  
WGETTOOL="/usr/bin/wget"  
export PGBIN=/usr/lib/postgresql/9.4/bin  
-- variables used by psql: https://www.postgresql.org/docs/current/static/libpq-envars.html  
export PGPORT=5432  
export PGHOST=localhost  
export PGUSER=postgres  
export PGPASSWORD=yourpasswordhere  
export PGDATABASE=geocoder  
PSQL=${PGBIN}/psql  
SHP2PGSQL=${PGBIN}/shp2pgsql  
cd /gisdata  
  
cd /gisdata  
wget ftp://ftp2.census.gov/geo/tiger/TIGER2015/PLACE/tl*_25_* --no-parent --relative -- ←  
recursive --level=2 --accept=zip --mirror --reject=html  
cd /gisdata/ftp2.census.gov/geo/tiger/TIGER2015/PLACE  
rm -f ${TMPDIR}/*. *  
:  
:
```

**Siehe auch**

Section 2.4.1, [Loader\\_Generate\\_Nation\\_Script](#)

## 12.2.11 Loader\_Generate\_Nation\_Script

Loader\_Generate\_Nation\_Script — Erzeugt für die angegebene Plattform ein Shell-Skript, welches die County und State Lookup Tabellen ladet.

### Synopsis

```
text loader_generate_nation_script(text os);
```

### Beschreibung

Erstellt für die gegebene Plattform ein Shell Skript, dass die Tabellen `county_all`, `county_all_lookup` und `state_all` in das Schema `tiger_data` lädt. Diese Tabellen erben jeweils von den Tabellen `county`, `county_lookup` und `state`, die sich im Schema `tiger` befinden.

Verwendet `unzip` auf Linux (auf Windows standardmäßig `7--zip`) und `wget` zum Herunterladen. Verwendet Section 4.7.2 um die Daten zu laden.

Verwendet die Kontrolltabellen `tiger.loader_platform`, `tiger.loader_variables` und `tiger.loader_lookuptab` um den Verarbeitungsprozess zu überprüfen, sowie unterschiedliche Varianten für die Syntax verschiedener Betriebssysteme.

1. `loader_variables` behält den Überblick über verschiedenen Variablen, wie Census Site, Jahr, Daten- und "staging"-Schemata
2. `loader_platform` Profile von verschiedenen Plattformen und Speicherplätze der ausführbaren Programme. Beinhaltet Windows und Linux/Unix. Weitere können hinzugefügt werden.
3. `loader_lookuptables` jeder Datensatz definiert einen bestimmten Tabellentyp (`state`, `county`), um Datensätze zu bearbeiten und zu importieren. Legt die Schritte fest, die notwendig sind, um Daten zu importieren, bereitzustellen und hinzuzufügen, und um Spalten, Indizes und Constraints zu löschen. Jede Tabelle wird mit einem Präfix des Bundesstaates versehen und erbt von einer Tabelle in dem Schema TIGER. z.B. wird die Tabelle `tiger_data.ma_faces` erstellt, welche von `tiger.faces` erbt

Verbessert: 2.4.1 zip code 5 tabulation area (zcta5) Ladeschritt wurde korrigiert und wenn aktiviert, werden zcta5 Daten als eine einzige Tabelle namens `zcta5_all` als Teil des Nationenskripts geladen.

Verfügbarkeit: 2.1.0

#### Note



Wenn Sie möchten, dass der Tabellierungsbereich der Postleitzahl 5 (zcta5) in Ihr Nationenskript geladen wird, gehen Sie wie folgt vor:

```
UPDATE tiger.loader_lookuptables SET load = true WHERE table_name = 'zcta510';
```

#### Note



Falls Sie die Version `tiger_2010` haben und `tiger_2011` laden wollen, dann müssen Sie als allererstes das Skript "loader\_generate\_nation\_script" und die Löschanweisungen [Drop\\_Nation\\_Tables\\_Generate\\_Script](#) ausführen, bevor Sie dieses Skript laufen lassen.

### Beispiele

Erzeugt ein Script um die Daten einer Nation in Windows zu laden.

```
SELECT loader_generate_nation_script('windows');
```

Erzeugt ein Script um die Daten auf Linux/Unix Systemen zu laden.

```
SELECT loader_generate_nation_script('sh');
```

**Siehe auch**

[Loader\\_Generate\\_Script](#), [Missing\\_Indexes\\_Generate\\_Script](#)

**12.2.12 Missing\_Indexes\_Generate\_Script**

`Missing_Indexes_Generate_Script` — Findet alle Tabellen mit Schlüsselspalten, die für JOINS vom Geokodierer verwendet werden und keinen Index aufweisen; gibt ein DDL (SQL) aus, das die Indizes für diese Tabellen festlegt.

**Synopsis**

```
text Missing_Indexes_Generate_Script();
```

**Beschreibung**

Findet alle Tabellen in den Schemata `tiger` und `tiger_data`, bei denen die Schlüsselspalten, die für Joins vom Geokodierer verwendet werden keine Indizes aufweisen. Weiters wird ein SQL-DDL Skript ausgegeben, das die Indizes für diese Tabellen festlegt. Dabei handelt es sich um eine Hilfsfunktion, die Abfragen schneller macht, indem die benötigten Indizes, die während des Imports gefehlt haben, neu hinzugefügt werden. Wenn der Geokodierer verbessert wird, dann wird diese Funktion aktualisiert um sie für die Verwendung neuer Indizes anzupassen. Wenn diese Funktion nichts zurückgibt, so bedeutet dies, dass alle Tabellen entsprechend befüllt und die Schlüsselindizes bereits vorhanden sind.

Verfügbarkeit: 2.0.0

**Beispiele**

```
SELECT missing_indexes_generate_script();
-- output: This was run on a database that was created before many corrections were made to ←
the loading script ---
CREATE INDEX idx_tiger_county_countyfp ON tiger.county USING btree(countyfp);
CREATE INDEX idx_tiger_cousub_countyfp ON tiger.cousub USING btree(countyfp);
CREATE INDEX idx_tiger_edges_tfidr ON tiger.edges USING btree(tfidr);
CREATE INDEX idx_tiger_edges_tfidl ON tiger.edges USING btree(tfidl);
CREATE INDEX idx_tiger_zip_lookup_all_zip ON tiger.zip_lookup_all USING btree(zip);
CREATE INDEX idx_tiger_data_ma_county_countyfp ON tiger_data.ma_county USING btree(countyfp ←
);
CREATE INDEX idx_tiger_data_ma_cousub_countyfp ON tiger_data.ma_cousub USING btree(countyfp ←
);
CREATE INDEX idx_tiger_data_ma_edges_countyfp ON tiger_data.ma_edges USING btree(countyfp);
CREATE INDEX idx_tiger_data_ma_faces_countyfp ON tiger_data.ma_faces USING btree(countyfp);
```

**Siehe auch**

[Loader\\_Generate\\_Script](#), [Install\\_Missing\\_Indexes](#)

**12.2.13 Normalize\_Address**

`Normalize_Address` — Für einen gegebenen Adressentext wird der zusammengesetzte Datentyp `norm_addy` zurückgeben, der ein Suffix und ein Präfix für die Straße, einen normierten Datentyp, die Straße, den Straßennamen etc. enthält und diese einzelnen Attributen zuweist. Diese Funktion benötigt lediglich die "lookup data", die mit dem Tiger Geokodierer paketiert sind (Tiger Census Daten werden nicht benötigt).

## Synopsis

```
norm_addy normalize_address(varchar in_address);
```

## Beschreibung

Für einen gegebenen Adresstext wird der zusammengesetzte Datentyp `norm_addy` zurückgeben, der ein Suffix und ein Präfix für die Straße, einen normierten Datentyp, die Straße, den Straßennamen etc. enthält und diese einzelnen Attributen zuweist. Dies ist der erste Schritt beim Geokodieren, der alle Adressen in eine standardisierte Postform bringt. Es werden keine anderen Daten, außer jenen die mit dem Geokodierer paketiert sind, benötigt.

Diese Funktion verwendet lediglich die verschiedenen Lookup-Tabellen "direction/state/suffix/", die mit `tiger_geocoder` vorinstalliert wurden und sich im Schema `tiger` befinden. Es ist deshalb nicht nötig Tiger Census Daten oder sonstige zusätzliche Daten herunterzuladen um diese Funktion zu verwenden.

Verwendet verschiedene Kontrolltabellen im Schema `tiger` zum Normalisieren der Eingabeadressen.

Die Attribute des Objekttyps `norm_addy`, die in dieser Reihenfolge von der Funktion zurückgegeben werden, wobei () für ein verbindliches Attribut und [] für ein optionales Attribut steht:

```
(address) [predirAbbrev] (streetName) [streetTypeAbbrev] [postdirAbbrev] [internal] [location] [stateAbbrev] [zip] [parsed] [zip4] [address_alphanumeric]
```

Erweiterung: 2.4.0 die zusätzlichen Felder "zip4" und "address\_alphanumeric" wurden zum Objekt "norm\_addy" hinzugefügt.

1. `address` ist eine Ganzzahl: Die Hausnummer
2. `predirAbbrev` ist ein Textfeld variabler Länge: Ein Präfix für die Straßenrichtung, wie N, S, E, W etc. Wird durch die `direction_lookup` Tabelle gesteuert.
3. `streetName` varchar
4. Das Textfeld `streetTypeAbbrev` mit variabler Länge beinhaltet eine abgekürzte Version der Straßentypen: z.B. St, Ave, Cir. Diese werden über die Tabelle `street_type_lookup` kontrolliert.
5. Das Textfeld `postdirAbbrev` mit variabler Länge beinhaltet Suffixe für die Abkürzungen der Straßenrichtung; N, S, E, W etc. Diese werden über die Tabelle `direction_lookup` kontrolliert.
6. `internal` ein Textfeld variabler Länge mit einer zusätzlichen Adressangabe, wie Apartment- oder Suitennummer.
7. `location` ein Textfeld variabler Länge, üblicherweise eine Stadt oder eine autonome Provinz.
8. `stateAbbrev` ein Textfeld variabler Länge mit dem zwei Zeichen langen Bundesstaat der USA. z.B. MA, NY, MI. Diese werden durch die Tabelle `state_lookup` beschränkt.
9. Das Textfeld `zip` mit variabler Länge enthält den 5-Zeichen langen Zip-Code. z.B. 02109
10. `parsed` boolesche Variable - zeigt an, ob eine Adresse durch Normalisierung erstellt wurde. Die Funktion "normalize\_address" setzt diese Variable auf TRUE, bevor die Adresse zurückgegeben wird.
11. `zip4` die letzten 4 Zeichen des 9 Zeichen langen Zip-Codes. Verfügbarkeit: PostGIS 2.4.0.
12. `address_alphanumeric` Vollständige Hausnummer, auch wenn sie Buchstaben wie bei "17R" enthält. Kann mit der Funktion [PgC\\_Normalize\\_Address](#) besser geparkt werden. Verfügbarkeit: PostGIS 2.4.0.

## Beispiele

Gibt die ausgewählten Felder aus. Verwenden Sie bitte [Pprint\\_Addy](#) wenn Sie einen sauber formatierten Ausgabertext 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](#)

### 12.2.14 Pagc\_Normalize\_Address

`Pagc_Normalize_Address` — Für einen gegebenen Adressentext wird der zusammengesetzte Datentyp `norm_addy` zurückgeben, der ein Suffix und ein Präfix für die Straße, einen normierten Datentyp, die Straße, den Straßennamen etc. enthält und diese einzelnen Attributen zuweist. Diese Funktion benötigt lediglich die "lookup data", die mit dem Tiger Geokodierer paketiert sind (Tiger Census Daten werden nicht benötigt). Benötigt die Erweiterung "address\_standardizer".

## Synopsis

```
norm_addy pagc_normalize_address(varchar in_address);
```

## Beschreibung

Für einen gegebenen Adressentext wird der zusammengesetzte Datentyp `norm_addy` zurückgeben, der ein Suffix und ein Präfix für die Straße, einen normierten Datentyp, die Straße, den Straßennamen etc. enthält und diese einzelnen Attributen zuweist. Dies ist der erste Schritt beim Geokodieren, der alle Adressen in eine standardisierte Postform bringt. Es werden keine anderen Daten, außer jenen die mit dem Geokodierer paketiert sind, benötigt.

Diese Funktion verwendet lediglich die verschiedenen Lookup-Tabellen "pagc\_\*", die mit `tiger_geocoder` vorinstalliert wurden und sich im Schema `tiger` befinden. Es ist deshalb nicht nötig Tiger Census Daten oder sonstige zusätzliche Daten herunterzuladen um diese Funktion zu verwenden. Möglicherweise stellt sich heraus, dass Sie Lookup-Tabellen im Schema `tiger` um weitere Abkürzungen und alternative Namensgebungen erweitern müssen.

Verwendet verschiedene Kontrolltabellen im Schema `tiger` zum Normalisieren der Eingabeadressen.

Die Attribute des Objekttyps `norm_addy`, die in dieser Reihenfolge von der Funktion zurückgegeben werden, wobei () für ein verbindliches Attribut und [] für ein optionales Attribut steht:

Bei [Normalize\\_Address](#) gibt es geringfügige Abweichungen beim Format und bei der Groß- und Kleinschreibung.

Verfügbarkeit: 2.1.0



Diese Methode benötigt die Erweiterung `address_standardizer`.

```
(address) [predirAbbrev] (streetName) [streetTypeAbbrev] [postdirAbbrev] [internal] [location] [stateAbbrev] [zip]
```

Die native "standardaddr" der Erweiterung "address\_standardizer" ist ein bisschen reichhaltiger als "norm\_addy", da es für die Unterstützung internationaler Adressen (inklusive Länder) entworfen wurde. Die entsprechenden Attribute von "standardaddr" sind:

```
house_num, predir, name, suftype, sufdire, unit, city, state, postcode
```

Erweiterung: 2.4.0 die zusätzlichen Felder "zip4" und "address\_alphanumeric" wurden zum Objekt "norm\_addy" hinzugefügt.

1. address ist eine Ganzzahl: Die Hausnummer
2. predirAbbrev ist ein Textfeld variabler Länge: Ein Präfix für die Straßenrichtung, wie N, S, E, W etc. Wird durch die direction\_lookup Tabelle gesteuert.
3. streetName varchar
4. Das Textfeld streetTypeAbbrev mit variabler Länge beinhaltet eine abgekürzte Version der Straßentypen: z.B. St, Ave, Cir. Diese werden über die Tabelle street\_type\_lookup kontrolliert.
5. Das Textfeld postdirAbbrev mit variabler Länge beinhaltet Suffixe für die Abkürzungen der Straßenrichtung; N, S, E, W etc. Diese werden über die Tabelle direction\_lookup kontrolliert.
6. internal ein Textfeld variabler Länge mit einer zusätzlichen Adressangabe, wie Apartment- oder Suitennummer.
7. location ein Textfeld variabler Länge, üblicherweise eine Stadt oder eine autonome Provinz.
8. stateAbbrev ein Textfeld variabler Länge mit dem zwei Zeichen langen Bundesstaat der USA. z.B. MA, NY, MI. Diese werden durch die Tabelle state\_lookup beschränkt.
9. Das Textfeld zip mit variabler Länge enthält den 5-Zeichen langen Zip-Code. z.B. 02109
10. parsed boolesche Variable - zeigt an, ob eine Adresse durch Normalisierung erstellt wurde. Die Funktion "normalize\_address" setzt diese Variable auf TRUE, bevor die Adresse zurückgegeben wird.
11. zip4 die letzten 4 Zeichen des 9 Zeichen langen Zip-Codes. Verfügbarkeit: PostGIS 2.4.0.
12. address\_alphanumeric Vollständige Hausnummer, auch wenn sie Buchstaben wie bei "17R" enthält. Kann mit der Funktion [Pagc\\_Normalize\\_Address](#) besser geparkt werden. Verfügbarkeit: PostGIS 2.4.0.

## Beispiele

### Beispiel für einen Einzelaufruf

```
SELECT addy.*
FROM pagc_normalize_address('9000 E ROO ST STE 999, Springfield, CO') AS addy;
```

address	predirabbrev	streetname	streettypeabbrev	postdirabbrev	internal	location	stateabbrev	zip	parsed
9000	E	ROO	ST			SUITE 999	CO		t

Batch call (Stapelverarbeitung). Zurzeit gibt es Geschwindigkeitsprobleme bezüglich des Wrappers des `postgis_tiger_geocoder` für den `address_standardizer`. Dies wird hoffentlich in späteren Versionen behoben. Wenn Sie Geschwindigkeit für den Aufruf einer Stapelverarbeitung zur Erstellung von `normaddy` benötigen, können Sie diese Probleme umgehen, indem Sie die Funktion `"standardize_address"` des `address_standardizer` direkt aufrufen. Dies wird nachfolgend gezeigt und entspricht ungefähr der Aufgabe unter [Normalize\\_Address](#) wo Daten verwendet werden, die unter [Geocode](#) erstellt wurden.

```
WITH g AS (SELECT address, ROW((sa).house_num, (sa).predir, (sa).name
, (sa).suftype, (sa).sufdir, (sa).unit, (sa).city, (sa).state, (sa).postcode, true)::
normaddy 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](#)

**12.2.15 Pprint\_Addy**

**Pprint\_Addy** — Für einen zusammengesetzten Objekttyp `norm_addy` wird eine formatierte Darstellung zurückgegeben. Wird üblicherweise in Verbindung mit `normalize_address` verwendet.

**Synopsis**

```
varchar pprint_addy(norm_addy in_addy);
```

**Beschreibung**

Für einen zusammengesetzten Objekttyp `norm_addy` wird eine formatierte Darstellung zurückgegeben. Außer den Daten die mit dem Geokodierer paketiert sind, werden keine weiteren Daten benötigt.

Wird üblicherweise in Verbindung mit [Normalize\\_Address](#) verwendet.

**Beispiele**

**Formatiert eine einzelne Adresse**

```
SELECT pprint_addy(normalize_address('202 East Fremont Street, Las Vegas, Nevada 89101')) ←
  As pretty_address;
      pretty_address
-----
202 E Fremont St, Las Vegas, NV 89101
```

**Adressen einer Tabelle formatieren**

```
SELECT address As orig, pprint_addy(normalize_address(address)) As pretty_address
  FROM addresses_to_geocode;
      orig | pretty_address
-----|-----
529 Main Street, Boston MA, 02129 | 529 Main St, Boston MA, 02129
77 Massachusetts Avenue, Cambridge, MA 02139 | 77 Massachusetts Ave, Cambridge, MA ←
28 Capen Street, Medford, MA | 28 Capen St, Medford, MA
124 Mount Auburn St, Cambridge, Massachusetts 02138 | 124 Mount Auburn St, Cambridge, MA ←
950 Main Street, Worcester, MA 01610 | 950 Main St, Worcester, MA 01610
```



**Siehe auch**

[Normalize\\_Address](#)

**12.2.16 Reverse\_Geocode**

**Reverse\_Geocode** — Nimmt einen geometrischen Punkt in einem bekannten Koordinatenreferenzsystem entgegen und gibt einen Datensatz zurück, das ein Feld mit theoretisch möglichen Adressen und ein Feld mit Straßenkreuzungen beinhaltet. Wenn `include_strnum_range = true`, dann beinhalten die Straßenkreuzungen den "Street Range" (Kennung des Straßenabschnitts).

**Synopsis**

```
record Reverse_Geocode(geometry pt, boolean include_strnum_range=false, geometry[] OUT intpt, norm_addy[] OUT addy, varchar[] OUT street);
```

**Beschreibung**

Nimmt einen geometrischen Punkt in einem bekannten Koordinatenreferenzsystem entgegen und gibt einen Datensatz zurück, das ein Feld mit theoretisch möglichen Adressen und ein Feld mit Straßenkreuzungen beinhaltet. Wenn `include_strnum_range = true`, dann beinhalten die Straßenkreuzungen den "Street Range" (Kennung des Straßenabschnitts). Die Standardeinstellung für `include_strnum_range` ist `FALSE`. Die Adressen werden nach dem Abstand des Punktes zu den jeweiligen Straßen gereiht, so dass die erste Adresse höchstwahrscheinlich die Richtige ist.

Warum sprechen wir von theoretischen anstatt von tatsächlichen Adressen. Die Daten von TIGER haben keine echten Adressen, sondern nur Straßenabschnitte (Street Ranges). Daher ist die theoretische Adresse eine anhand der Straßenabschnitte interpolierte Adresse. So gibt zum Beispiel die Interpolation einer Adresse "26 Court St." und "26 Court Sq." zurück, obwohl keine Adresse mit der Bezeichnung "26 Court Sq." existiert. Dies beruht darauf, dass ein Punkt an einem Eckpunkt von 2 Straßen liegen kann und die Logik entlang beider Straßen interpoliert. Die Logik nimmt auch an, dass sich die Adressen in einem regelmäßigen Abstand entlang der Straße befinden, was natürlich falsch ist, da ein öffentliches Gebäude einen großen Bereich eines Straßenabschnitts (street range) einnehmen kann und der Rest der Gebäude sich lediglich am Ende befinden.

Anmerkung: diese Funktion benötigt TIGER-Daten. Wenn Sie für diesen Bereich keine Daten geladen haben, dann bekommen Sie einen Datensatz mit lauter NULLs zurück.

Die zurückgelieferten Elemente des Datensatzes lauten wie folgt:

1. `intpt` ist ein Feld mit Punkten: Dies sind Punkte auf der Mittellinie der Straße, die dem gegebenen Punkt am nächsten liegt. Es beinhaltet soviele Punkte wie es Adressen gibt.
2. `addy` ist ein Feld mit normalisierten Adressen (`norm_addy`): Diese sind ein Feld mit möglichen Adressen die zu dem gegebenen Punkt passen. Die erste in dem Feld ist die wahrscheinlichste Adresse. Üblicherweise sollte dies nur eine Adresse sein, ausgenommen dem Fall wo ein Punkt an der Ecke von 2 oder 3 Straßen liegt, oder wenn der Punkt irgendwo auf der Straße und nicht auf der Seite liegt.
3. `street` ein Feld mit `varchar` (Textfeld variabler Länge): Dies sind Querstraßen (oder die Straße) (sich kreuzende Straßen oder die Straße auf der der Punkt liegt).

Verbessert: 2.4.1 Wenn der optionale `zcta5`-Datensatz geladen ist, kann die Funktion `reverse_geocode` nach `state` und `zip` auflösen, auch wenn die spezifischen Statusdaten nicht geladen sind. Siehe [Loader\\_Generate\\_Nation\\_Script](#) für Einzelheiten zum Laden von `zcta5`-Daten.

Verfügbarkeit: 2.0.0

**Beispiele**

Beispiel für eine Position an der Ecke von zwei Straßen, aber näher bei einer. Näherungsweise die Lage des MIT: 77 Massachusetts Ave, Cambridge, MA 02139. Beachten Sie, dass obwohl wir keine 3 Straßen haben, PostgreSQL nur NULL für die Einträge oberhalb unserer oberen Begrenzung zurückgibt; kann also gefahrlos verwendet werden. Dieses Beispiel verwendet Adressbereiche

```
SELECT pprint_addy(r.addy[1]) As st1, pprint_addy(r.addy[2]) As st2, pprint_addy(r.addy[3]) ←
  As st3,
  array_to_string(r.street, ',') As cross_streets
  FROM reverse_geocode(ST_GeomFromText('POINT(-71.093902 42.359446)',4269),true) As r ←
  ;
```

```
result
-----
      st1                                     | st2 | st3 |                                     cross_streets
-----+-----+-----+-----
67 Massachusetts Ave, Cambridge, MA 02139 |     |     | 67 - 127 Massachusetts Ave,32 - 88 ←
  Vassar St
```

Hier haben wir die Adressbereiche für die Querstraßen nicht inkludiert und eine Position ausgewählt, die sehr sehr nahe an einer Ecke von 2 Straßen liegt, damit diese (Position) von zwei unterschiedlichen Adressen erkannt werden könnte.

```
SELECT pprint_addy(r.addy[1]) As st1, pprint_addy(r.addy[2]) As st2,
pprint_addy(r.addy[3]) As st3, array_to_string(r.street, ',') As cross_str
FROM reverse_geocode(ST_GeomFromText('POINT(-71.06941 42.34225)',4269)) As r;
```

```
result
-----
      st1                                     |          st2                                     | st3 | cross_str
-----+-----+-----+-----
5 Bradford St, Boston, MA 02118 | 49 Waltham St, Boston, MA 02118 |     | Waltham St
```

Bei diesem Beispiel wird das Beispiel von **Geocode** wiederverwendet und wir wollen nur die primäre Adresse und höchstens 2 Straßenkreuzungen.

```
SELECT actual_addr, lon, lat, pprint_addy((rg).addy[1]) As int_addr1,
  (rg).street[1] As cross1, (rg).street[2] As cross2
FROM (SELECT address As actual_addr, lon, lat,
  reverse_geocode( ST_SetSRID(ST_Point(lon,lat),4326) ) As rg
  FROM addresses_to_geocode WHERE rating
> -1) As foo;
```

```
      actual_addr                                     | lon | lat | ←
      cross2                                     | int_addr1 | cross1 | ←
-----+-----+-----+-----
529 Main Street, Boston MA, 02129 | -71.07181 | 42.38359 | 527 Main St, ←
  Boston, MA 02129 | Medford St |
77 Massachusetts Avenue, Cambridge, MA 02139 | -71.09428 | 42.35988 | 77 ←
  Massachusetts Ave, Cambridge, MA 02139 | Vassar St |
26 Capen Street, Medford, MA | -71.12377 | 42.41101 | 9 Edison Ave, ←
  Medford, MA 02155 | Capen St | Tesla Ave
124 Mount Auburn St, Cambridge, Massachusetts 02138 | -71.12304 | 42.37328 | 3 University ←
  Rd, Cambridge, MA 02138 | Mount Auburn St |
950 Main Street, Worcester, MA 01610 | -71.82368 | 42.24956 | 3 Maywood St, ←
  Worcester, MA 01603 | Main St | Maywood Pl
```

## Siehe auch

[Pprint\\_Addy](#), [Loader\\_Generate\\_Nation\\_Script](#)

### 12.2.17 Topology\_Load\_Tiger

`Topology_Load_Tiger` — Lädt die Tiger-Daten einer bestimmte Region in die PostGIS Topologie, transformiert sie in das Koordinatenreferenzsystem der Topologie und fängt sie entsprechend der Genauigkeitstoleranz der Topologie.

#### Synopsis

```
text Topology_Load_Tiger(varchar topo_name, varchar region_type, varchar region_id);
```

#### Beschreibung

Lädt die Tiger Daten einer bestimmten Region in die PostGIS Topologie. Die Maschen, Knoten und Kanten werden in das Koordinatenreferenzsystem der ZIELtopologie transformiert und die Knoten werden entsprechend der Toleranz der ZIELtopologie gefangen. Die erzeugten Maschen, Knoten und Kanten behalten die ids der ursprünglichen Maschen, Knoten und Kanten der Tiger Daten, wodurch die Daten zukünftig leichter mit neuen Tiger Daten abgeglichen werden können. Gibt eine Zusammenfassung des Prozessablaufs aus.

Dies ist zum Beispiel nützlich, wenn Daten neu strukturiert werden müssen, bei denen die neu ausgebildeten Polygone an den Mittellinien der Strassen ausgerichtet sein sollen und die erzeugten Polygone sich nicht überlappen dürfen.



#### Note

Diese Funktion benötigt Tiger Daten und das Topologiemodul von PostGIS. Für weiterführende Information siehe Chapter 9 und Section 2.2.3. Wenn keine Daten für den betreffenden Bereich geladen wurden, dann werden auch keine topologischen Datensätze erstellt. Diese Funktion versagt auch dann, wenn keine Topologie mit den topologischen Funktionen aufgebaut wurde.



#### Note

Die meisten topologischen Überprüfungsfehler entstehen durch Toleranzprobleme, wenn nach der Transformation die Kanten nicht genau abgeglichen sind oder sich überlappen. Um dies zu beheben, können Sie bei topologischen Überprüfungsfehlern die Präzision erhöhen oder erniedrigen.

Benötigte Parameter:

1. `topo_name` Die Bezeichnung einer bestehenden PostGIS Topologie, in die Daten geladen werden.
2. `region_type` Der Typ des begrenzten Bereichs. Zurzeit wird nur `place` und `county` unterstützt. Geplant sind noch einige weitere Typen. In dieser Tabelle werden die Begrenzungen definiert. z.B. `tiger.place`, `tiger.county`
3. `region_id` Entspricht der "geoid" von TIGER und ist ein eindeutige Identifikator für die Region in der Tabelle. Für Plätze ist es die Spalte `plcidfp` in `tiger.place`. Für "county" ist es die Spalte `cntyidfp` in `tiger.county`

Verfügbarkeit: 2.0.0

**Beispiel: Boston, Massachusetts Topologie**

Erstellt eine Topologie für Boston, Massachusetts in "Mass State Plane Feet" (2249) mit einer Toleranz von 0.25 Meter und lädt anschließend die Maschen, Kanten und Knoten von Tiger für Boston City.

```
SELECT topology.CreateTopology('topo_boston', 2249, 0.25);
createtopology
-----
      15
-- 60,902 ms ~ 1 minute on windows 7 desktop running 9.1 (with 5 states tiger data loaded)
SELECT tiger.topology_load_tiger('topo_boston', 'place', '2507000');
-- topology_loader_tiger --
29722 edges holding in temporary. 11108 faces added. 1875 edges of faces added. 20576 ←
      nodes added.
19962 nodes contained in a face. 0 edge start end corrected. 31597 edges added.

-- 41 ms --
SELECT topology.TopologySummary('topo_boston');
-- topologysummary--
Topology topo_boston (15), SRID 2249, precision 0.25
20576 nodes, 31597 edges, 11109 faces, 0 topogeoms in 0 layers

-- 28,797 ms to validate yeh returned no errors --
SELECT * FROM
      topology.ValidateTopology('topo_boston');

      error          |      id1      |      id2
-----+-----+-----
```

**Beispiel: Suffolk, Massachusetts Topologie**

Erstellt eine Topologie für Suffolk, Massachusetts in "Mass State Plane Meters" (26986) mit einer Toleranz von 0.25 Meter und lädt anschließend die Maschen, Kanten und Knoten von Tiger für Suffolk County.

```
SELECT topology.CreateTopology('topo_suffolk', 26986, 0.25);
-- this took 56,275 ms ~ 1 minute on Windows 7 32-bit with 5 states of tiger loaded
-- must have been warmed up after loading boston
SELECT tiger.topology_load_tiger('topo_suffolk', 'county', '25025');
-- topology_loader_tiger --
36003 edges holding in temporary. 13518 faces added. 2172 edges of faces added.
24761 nodes added. 24075 nodes contained in a face. 0 edge start end corrected. 38175 ←
      edges added.
-- 31 ms --
SELECT topology.TopologySummary('topo_suffolk');
-- topologysummary--
Topology topo_suffolk (14), SRID 26986, precision 0.25
24761 nodes, 38175 edges, 13519 faces, 0 topogeoms in 0 layers

-- 33,606 ms to validate --
SELECT * FROM
      topology.ValidateTopology('topo_suffolk');

      error          |      id1      |      id2
-----+-----+-----
coincident nodes   | 81045651 | 81064553
edge crosses node  | 81045651 | 85737793
edge crosses node  | 81045651 | 85742215
edge crosses node  | 81045651 | 620628939
edge crosses node  | 81064553 | 85697815
edge crosses node  | 81064553 | 85728168
edge crosses node  | 81064553 | 85733413
```

**Siehe auch**

[CreateTopology](#), [CreateTopoGeom](#), [TopologySummary](#), [ValidateTopology](#)

**12.2.18 Set\_Geocode\_Setting**

Set\_Geocode\_Setting — Setzt die Einstellungen, welche das Verhalten der Funktionen des Geokodierers beeinflussen.

**Synopsis**

```
text Set_Geocode_Setting(text setting_name, text setting_value);
```

**Beschreibung**

Setzt bestimmte Einstellwerte, die in der Tabelle "tiger.geocode\_settings" gespeichert werden. Die Einstellungen erlauben auf "debugging" der Funktionen umzuschalten. Für später ist geplant auch die Rangordnung über die Einstellungen zu kontrollieren. Eine aktuelle Liste der Einstellungen ist unter [Get\\_Geocode\\_Setting](#) aufgeführt.

Verfügbarkeit: 2.1.0

**Das Beispiel gibt die "debugging" Einstellungen aus**

Wenn Sie [Geocode](#) ausführen und diese Funktion TRUE ist, dann werden die Abfragen und die Zeitmessung von dem Log "NOTICE" ausgegeben.

```
SELECT set_geocode_setting('debug_geocode_address', 'true') As result;
result
-----
true
```

**Siehe auch**

[Get\\_Geocode\\_Setting](#)

## Chapter 13

# PostGIS Spezialfunktionen Index

### 13.1 PostGIS-Aggregat-Funktionen

Bei den folgenden Funktionen handelt es sich um räumliche Aggregatfunktionen, die auf die gleiche Weise wie SQL-Aggregatfunktionen wie `sum` und `average` verwendet werden.

- **CG\_3DUnion** - Perform 3D union using `postgis_sfcgal`.
  - **ST\_3DExtent** - Aggregatfunktion, die den 3D-Begrenzungsrahmen von Geometrien zurückgibt.
  - **ST\_3DUnion** - 3D-Vereinigung durchführen.
  - **ST\_AsFlatGeobuf** - Rückgabe einer FlatGeobuf-Darstellung einer Reihe von Zeilen.
  - **ST\_AsGeobuf** - Gibt eine Menge an Zeilen in der Geobuf Darstellung aus.
  - **ST\_AsMVT** - Aggregatfunktion, die eine MVT-Darstellung einer Reihe von Zeilen zurückgibt.
  - **ST\_ClusterDBSCAN** - Fensterfunktion, die eine Cluster-ID für jede Eingabegeometrie unter Verwendung des DBSCAN-Algorithmus zurückgibt.
  - **ST\_ClusterIntersecting** - Aggregatfunktion, die Eingabegeometrien zu zusammenhängenden Mengen clustert.
  - **ST\_ClusterIntersectingWin** - Fensterfunktion, die für jede Eingabegeometrie eine Cluster-ID zurückgibt und die Eingabegeometrien in zusammenhängende Gruppen clustert.
  - **ST\_ClusterKMeans** - Fensterfunktion, die eine Cluster-ID für jede Eingabegeometrie unter Verwendung des K-Means-Algorithmus zurückgibt.
  - **ST\_ClusterWithin** - Aggregatfunktion, die Geometrien nach Trennungsabstand gruppiert.
  - **ST\_ClusterWithinWin** - Fensterfunktion, die eine Cluster-ID für jede Eingabegeometrie zurückgibt, Clustering anhand des Trennungsabstands.
  - **ST\_Collect** - Erzeugt eine `GeometryCollection` oder `Multi*-Geometrie` aus einer Reihe von Geometrien.
  - **ST\_CoverageInvalidEdges** - Fensterfunktion, die Stellen findet, an denen die Polygone keine gültige Abdeckung bilden.
  - **ST\_CoverageSimplify** - Fensterfunktion, die die Kanten einer polygonalen Abdeckung vereinfacht.
  - **ST\_CoverageUnion** - Berechnet die Vereinigung einer Menge von Polygonen, die eine Abdeckung bilden, indem gemeinsame Kanten entfernt werden.
  - **ST\_Extent** - Aggregatfunktion, die den Begrenzungsrahmen von Geometrien zurückgibt.
  - **ST\_MakeLine** - Erzeugt einen Linienzug aus einer Punkt-, Mehrfachpunkt- oder Liniengeometrie.
-

- **ST\_MemUnion** - Aggregatfunktion, die Geometrien auf eine speichereffiziente, aber langsamere Weise zusammenfasst
- **ST\_Polygonize** - Berechnet eine Sammlung von Polygonen, die aus dem Linienwerk einer Reihe von Geometrien gebildet werden.
- **ST\_SameAlignment** - Gibt TRUE zurück, wenn die Raster die selbe Rotation, Skalierung, Koordinatenreferenzsystem und Versatz (Pixel können auf dasselbe Gitter gelegt werden, ohne dass die Gitterlinien durch die Pixel schneiden) aufweisen. Wenn nicht, wird FALSE und eine Beschreibung des Problems ausgegeben.
- **ST\_Union** - Berechnet eine Geometrie, die die Punktmengenvereinigung der Eingabegeometrien darstellt.
- **ST\_Union** - Gibt die Vereinigung mehrerer Rasterkacheln in einem einzelnen Raster mit mehreren Bändern zurück.
- **TopoElementArray\_Agg** - Gibt für eine Menge an element\_id, type Feldern (topoelements) ein topoelementarray zurück.

## 13.2 PostGIS-Fenster-Funktionen

Bei den folgenden Funktionen handelt es sich um räumliche Fensterfunktionen, die auf die gleiche Weise wie SQL-Fensterfunktionen wie `row_number()`, `lead()` und `lag()` verwendet werden. Sie müssen von einer `OVER()` Klausel gefolgt werden.

- **ST\_ClusterDBSCAN** - Fensterfunktion, die eine Cluster-ID für jede Eingabegeometrie unter Verwendung des DBSCAN-Algorithmus zurückgibt.
- **ST\_ClusterIntersectingWin** - Fensterfunktion, die für jede Eingabegeometrie eine Cluster-ID zurückgibt und die Eingabegeometrien in zusammenhängende Gruppen clustert.
- **ST\_ClusterKMeans** - Fensterfunktion, die eine Cluster-ID für jede Eingabegeometrie unter Verwendung des K-Means-Algorithmus zurückgibt.
- **ST\_ClusterWithinWin** - Fensterfunktion, die eine Cluster-ID für jede Eingabegeometrie zurückgibt, Clustering anhand des Trennungsabstands.
- **ST\_CoverageInvalidEdges** - Fensterfunktion, die Stellen findet, an denen die Polygone keine gültige Abdeckung bilden.
- **ST\_CoverageSimplify** - Fensterfunktion, die die Kanten einer polygonalen Abdeckung vereinfacht.

## 13.3 PostGIS SQL-MM-kompatible Funktionen

Die folgenden Funktionen sind PostGIS-Funktionen, die dem SQL/MM 3-Standard entsprechen

- **CG\_3DArea** - Berechnet die Fläche von 3D-Oberflächengeometrien. Gibt 0 für Solids zurück.
- **CG\_3DDifference** - 3D-Differenz durchführen
- **CG\_3DIntersection** - 3D-Schnitte durchführen
- **CG\_3DUnion** - Perform 3D union using `postgis_sfcgal`.
- **CG\_Volume** - Berechnet das Volumen eines 3D-Volumens. Bei Anwendung auf (auch geschlossene) Flächengeometrien wird 0 zurückgegeben.
- **ST\_3DArea** - Berechnet die Fläche von 3D-Oberflächengeometrien. Gibt 0 für Solids zurück.
- **ST\_3DDWithin** - Prüft, ob zwei 3D-Geometrien innerhalb eines bestimmten 3D-Abstands liegen
- **ST\_3DDifference** - 3D-Differenz durchführen
- **ST\_3DDistance** - Für den geometrischen Datentyp. Es wird der geringste 3-dimensionale kartesische Abstand (basierend auf dem Koordinatenreferenzsystem) zwischen zwei geometrischen Objekten in projizierten Einheiten zurückgegeben.

- **ST\_3DIntersection** - 3D-Schnitte durchführen
  - **ST\_3DIntersects** - Prüft, ob sich zwei Geometrien in 3D räumlich schneiden - nur für Punkte, Linienzüge, Polygone, polyedrische Flächen (Bereich)
  - **ST\_3DLength** - Gibt den geometrischen Schwerpunkt einer Geometrie zurück.
  - **ST\_3DPerimeter** - Gibt den geometrischen Schwerpunkt einer Geometrie zurück.
  - **ST\_3DUnion** - 3D-Vereinigung durchführen.
  - **ST\_AddEdgeModFace** - Fügt eine Kante hinzu. Falls dabei eine Masche aufgetrennt wird, so wird die ursprüngliche Masche angepasst und eine weitere Masche hinzugefügt.
  - **ST\_AddEdgeNewFaces** - Fügt eine Kante hinzu. Falls dabei eine Masche aufgetrennt wird, so wird die ursprüngliche Masche gelöscht und durch zwei neue Maschen ersetzt.
  - **ST\_AddIsoEdge** - Fügt eine isolierte Kante, die durch die Geometrie alinestring festgelegt wird zu einer Topologie hinzu, indem zwei bestehende isolierte Knoten anode und anothernode verbunden werden. Gibt die "edgeid" der neuen Kante aus.
  - **ST\_AddIsoNode** - Fügt einen isolierten Knoten zu einer Masche in einer Topologie hinzu und gibt die "nodeid" des neuen Knotens aus. Falls die Masche NULL ist, wird der Knoten dennoch erstellt.
  - **ST\_Area** - Gibt den geometrischen Schwerpunkt einer Geometrie zurück.
  - **ST\_AsBinary** - Rückgabe der OGC/ISO Well-Known Binary (WKB)-Darstellung der Geometrie/Geografie ohne SRID-Metadaten.
  - **ST\_AsGML** - Gibt die Geometrie als GML-Element - Version 2 oder 3 - zurück.
  - **ST\_AsText** - Gibt die Well-known-Text(WKT) Darstellung der Geometrie/Geographie ohne die SRID Metadaten zurück.
  - **ST\_Boundary** - Gibt die abgeschlossene Hülle aus der kombinierten Begrenzung der Geometrie zurück.
  - **ST\_Buffer** - Berechnet eine Geometrie, die alle Punkte innerhalb eines bestimmten Abstands zu einer Geometrie umfasst.
  - **ST\_Centroid** - Gibt den geometrischen Schwerpunkt einer Geometrie zurück.
  - **ST\_ChangeEdgeGeom** - Ändert die geometrische Form einer Kante, ohne sich auf die topologische Struktur auszuwirken.
  - **ST\_Contains** - Tests, wenn jeder Punkt von B in A liegt und ihre Innenräume einen gemeinsamen Punkt haben
  - **ST\_ConvexHull** - Berechnet die konvexe Hülle einer Geometrie.
  - **ST\_CoordDim** - Gibt die Dimension der Koordinaten für den Wert von ST\_Geometry zurück.
  - **ST\_CreateTopoGeo** - Fügt eine Sammlung von Geometrien an eine leere Topologie an und gibt eine Bestätigungsmeldung aus.
  - **ST\_Crosses** - Prüft, ob zwei Geometrien einige, aber nicht alle, innere Punkte gemeinsam haben
  - **ST\_CurveN** - Returns the Nth component curve geometry of a CompoundCurve.
  - **ST\_CurveToLine** - Konvertiert eine Geometrie mit Kurven in eine lineare Geometrie.
  - **ST\_Difference** - Berechnet eine Geometrie, die den Teil der Geometrie A darstellt, der die Geometrie B nicht schneidet.
  - **ST\_Dimension** - Gibt die Dimension der Koordinaten für den Wert von ST\_Geometry zurück.
  - **ST\_Disjoint** - Prüft, ob zwei Geometrien keine gemeinsamen Punkte haben
  - **ST\_Distance** - Gibt die größte 3-dimensionale Distanz zwischen zwei geometrischen Objekten als Linie zurück
  - **ST\_EndPoint** - Gibt die Anzahl der Stützpunkte eines ST\_LineString oder eines ST\_CircularString zurück.
  - **ST\_Envelope** - Gibt eine Geometrie in doppelter Genauigkeit (float8) zurück, welche das Umgebungsrechteck der beigegebenen Geometrie darstellt.
  - **ST\_Equals** - Prüft, ob zwei Geometrien dieselbe Menge von Punkten enthalten
-



- **ST\_ExteriorRing** - Gibt die Anzahl der inneren Ringe einer Polygoneometrie aus.
  - **ST\_GMLToSQL** - Gibt einen spezifizierten ST\_Geometry Wert aus einer GML-Darstellung zurück. Dies ist ein Aliasname für ST\_GeomFromGML
  - **ST\_GeomCollFromText** - Erzeugt eine Sammelgeometrie mit der gegebenen SRID aus einer WKT-Kollektion. Wenn keine SRID angegeben ist, wird diese standardmäßig auf 0 gesetzt.
  - **ST\_GeomFromText** - Gibt einen spezifizierten ST\_Geometry Wert aus einer Well-known-Text Darstellung (WKT) zurück.
  - **ST\_GeomFromWKB** - Erzeugt ein geometrisches Objekt aus der Well-known-Binary (WKB) Darstellung und einer optionalen SRID.
  - **ST\_GeometryFromText** - Gibt einen spezifizierten ST\_Geometry-Wert von einer Well-known-Text Darstellung (WKT) zurück. Die Bezeichnung ist ein Alias für ST\_GeomFromText
  - **ST\_GeometryN** - Gibt den Geometrietyp des ST\_Geometry Wertes zurück.
  - **ST\_GeometryType** - Gibt den Geometrietyp des ST\_Geometry Wertes zurück.
  - **ST\_GetFaceEdges** - Gibt die Kanten, die aface begrenzen, sortiert aus.
  - **ST\_GetFaceGeometry** - Gibt für eine Topologie und eine bestimmte Maschen-ID das Polygon zurück.
  - **ST\_InitTopoGeo** - Erstellt ein neues Topologie-Schema und trägt es in die Tabelle topology.topology ein.
  - **ST\_InteriorRingN** - Gibt die Anzahl der inneren Ringe einer Polygoneometrie aus.
  - **ST\_Intersection** - Berechnet eine Geometrie, die den gemeinsamen Teil der Geometrien A und B darstellt.
  - **ST\_Intersects** - Prüft, ob sich zwei Geometrien schneiden (sie haben mindestens einen Punkt gemeinsam)
  - **ST\_IsClosed** - Gibt den Wert TRUE zurück, wenn die Anfangs- und Endpunkte des LINESTRING's zusammenfallen. Bei polyedrischen Oberflächen, wenn sie geschlossen (volumetrisch) sind.
  - **ST\_IsEmpty** - Prüft, ob eine Geometrie leer ist.
  - **ST\_IsRing** - Prüft, ob ein LineString geschlossen und einfach ist.
  - **ST\_IsSimple** - Gibt den Wert (TRUE) zurück, wenn die Geometrie keine irregulären Stellen, wie Selbstüberschneidungen oder Selbstberührungen, aufweist.
  - **ST\_IsValid** - Prüft, ob eine Geometrie in 2D wohlgeformt ist.
  - **ST\_Length** - Gibt den geometrischen Schwerpunkt einer Geometrie zurück.
  - **ST\_LineFromText** - Erzeugt eine Geometrie aus einer WKT Darstellung mit der angegebenen SRID. Wenn keine SRID angegeben wird, wird diese standardmäßig auf 0 gesetzt.
  - **ST\_LineFromWKB** - Erzeugt einen LINESTRING mit gegebener SRID aus einer WKB-Darstellung
  - **ST\_LinestringFromWKB** - Erzeugt eine Geometrie mit gegebener SRID aus einer WKB-Darstellung.
  - **ST\_LocateAlong** - Gibt die Punkte auf einer Geometrie zurück, die einem Messwert entsprechen.
  - **ST\_LocateBetween** - Gibt die Teile einer Geometrie zurück, die einem Messbereich entsprechen.
  - **ST\_M** - Gibt die M-Koordinate eines Punktes zurück.
  - **ST\_MLineFromText** - Liest einen festgelegten ST\_MultiLineString Wert von einer WKT-Darstellung aus.
  - **ST\_MPointFromText** - Erzeugt eine Geometrie aus WKT mit der angegebenen SRID. Wenn keine SRID angegeben wird, wird diese standardmäßig auf 0 gesetzt.
  - **ST\_MPolyFromText** - Erzeugt eine MultiPolygon Geometrie aus WKT mit der angegebenen SRID. Wenn SRID nicht angegeben ist, wird sie standardmäßig auf 0 gesetzt.
-

- **ST\_ModEdgeHeal** - "Heilt" zwei Kanten, indem der verbindende Knoten gelöscht wird, die erste Kante modifiziert und die zweite Kante gelöscht wird. Gibt die ID des gelöschten Knoten zurück.
  - **ST\_ModEdgeSplit** - Trennt eine Kante auf, indem ein neuer Knoten entlang einer bestehenden Kante erstellt wird. Ändert die ursprüngliche Kante und fügt eine neue Kante hinzu.
  - **ST\_MoveIsoNode** - Verschiebt einen isolierten Knoten in einer Topologie von einer Stelle an eine andere. Falls die neue Geometrie apoint bereits als Knoten existiert, wird eine Fehlermeldung ausgegeben. Gibt eine Beschreibung der Verschiebung aus.
  - **ST\_NewEdgeHeal** - "Heilt" zwei Kanten, indem der verbindende Knoten und beide Kanten gelöscht werden. Die beiden Kanten werden durch eine Kante ersetzt, welche dieselbe Ausrichtung wie die erste Kante hat.
  - **ST\_NewEdgesSplit** - Trennt eine Kante auf, indem ein neuer Knoten entlang einer bestehenden Kante erstellt, die ursprüngliche Kante gelöscht und durch zwei neue Kanten ersetzt wird. Gibt die ID des neu erstellten Knotens aus, der die neuen Kanten verbindet.
  - **ST\_NumCurves** - Return the number of component curves in a CompoundCurve.
  - **ST\_NumGeometries** - Gibt die Anzahl der Punkte einer Geometrie zurück. Funktioniert für alle Geometrien.
  - **ST\_NumInteriorRings** - Gibt die Anzahl der inneren Ringe einer Polygoneometrie aus.
  - **ST\_NumPatches** - Gibt die Anzahl der Maschen einer polyedrischen Oberfläche aus. Gibt NULL zurück, wenn es sich nicht um polyedrische Geometrien handelt.
  - **ST\_NumPoints** - Gibt die Anzahl der Stützpunkte eines ST\_LineString oder eines ST\_CircularString zurück.
  - **ST\_OrderingEquals** - Prüft, ob zwei Geometrien die gleiche Geometrie darstellen und Punkte in der gleichen Richtungsreihenfolge haben
  - **ST\_Overlaps** - Prüft, ob zwei Geometrien die gleiche Abmessung haben und sich schneiden, aber jede mindestens einen Punkt hat, der nicht in der anderen liegt
  - **ST\_PatchN** - Gibt den Geometrietyp des ST\_Geometry Wertes zurück.
  - **ST\_Perimeter** - Gibt die Länge der Begrenzung einer polygonalen Geometrie oder Geografie zurück.
  - **ST\_Point** - Erzeugt einen Punkt mit X-, Y- und SRID-Werten.
  - **ST\_PointFromText** - Erzeugt eine Punktgeometrie mit gegebener SRID von WKT. Wenn SRID nicht angegeben ist, wird sie standardmäßig auf 0 gesetzt.
  - **ST\_PointFromWKB** - Erzeugt eine Geometrie mit gegebener SRID von WKB.
  - **ST\_PointN** - Gibt die Anzahl der Stützpunkte eines ST\_LineString oder eines ST\_CircularString zurück.
  - **ST\_PointOnSurface** - Berechnet einen Punkt, der garantiert in einem Polygon oder auf einer Geometrie liegt.
  - **ST\_Polygon** - Erzeugt ein Polygon aus einem LineString mit einem angegebenen SRID.
  - **ST\_PolygonFromText** - Erzeugt eine Geometrie aus WKT mit der angegebenen SRID. Wenn keine SRID angegeben wird, wird diese standardmäßig auf 0 gesetzt.
  - **ST\_Relate** - Prüft, ob zwei Geometrien eine topologische Beziehung haben, die einem Schnittpunktmatrixmuster entspricht, oder berechnet ihre Schnittpunktmatrix
  - **ST\_RemEdgeModFace** - Entfernt eine Kante, und wenn die Kante zwei Flächen trennt, wird eine Fläche gelöscht und die andere Fläche so verändert, dass sie den Raum beider Flächen abdeckt.
  - **ST\_RemEdgeNewFace** - Entfernt eine Kante. Falls die gelöschte Kante zwei Maschen voneinander getrennt hat, werden die ursprünglichen Maschen gelöscht und durch einer neuen Masche ersetzt.
  - **ST\_RemoveIsoEdge** - Löscht einen isolierten Knoten und gibt eine Beschreibung der getroffenen Maßnahmen aus. Falls der Knoten nicht isoliert ist, wird eine Fehlermeldung ausgegeben.
-

- **ST\_RemoveIsoNode** - Löscht einen isolierten Knoten und gibt eine Beschreibung der getroffenen Maßnahmen aus. Falls der Knoten nicht isoliert ist (ist der Anfangs- oder der Endpunkt einer Kante), wird eine Fehlermeldung ausgegeben.
- **ST\_SRID** - Gibt die Raumbezugskenung für eine Geometrie zurück.
- **ST\_StartPoint** - Gibt den ersten Punkt eines LineString zurück.
- **ST\_SymDifference** - Berechnet eine Geometrie, die die Teile der Geometrien A und B darstellt, die sich nicht überschneiden.
- **ST\_Touches** - Prüft, ob zwei Geometrien mindestens einen Punkt gemeinsam haben, aber ihre Innenräume sich nicht schneiden
- **ST\_Transform** - Rückgabe einer neuen Geometrie mit in ein anderes räumliches Bezugssystem transformierten Koordinaten.
- **ST\_Union** - Berechnet eine Geometrie, die die Punktmengenvereinigung der Eingabegeometrien darstellt.
- **ST\_Volume** - Berechnet das Volumen eines 3D-Volumens. Bei Anwendung auf (auch geschlossene) Flächengeometrien wird 0 zurückgegeben.
- **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
- **ST\_WKTToSQL** - Gibt einen spezifizierten ST\_Geometry-Wert von einer Well-known-Text Darstellung (WKT) zurück. Die Bezeichnung ist ein Alias für ST\_GeomFromText
- **ST\_Within** - Tests, wenn jeder Punkt von A in B liegt und ihre Innenräume einen gemeinsamen Punkt haben
- **ST\_X** - Gibt die X-Koordinate eines Punktes zurück.
- **ST\_Y** - Gibt die Y-Koordinate eines Punktes zurück.
- **ST\_Z** - Gibt die Z-Koordinate eines Punktes zurück.
- **ST\_SRID** - Gibt den räumlichen Referenzbezeichner für eine Topogeometrie zurück.

## 13.4 PostGIS-Funktionen zur Unterstützung der Geografie

Die unten aufgeführten Funktionen und Operatoren sind PostGIS-Funktionen/Operatoren, die als Eingabe ein Objekt des Datentyps **geography** annehmen oder als Ausgabe zurückgeben.

---

### Note



Funktionen mit einem (T) sind keine nativen geodätischen Funktionen und verwenden einen ST\_Transform-Aufruf zu und von der Geometrie, um die Operation durchzuführen. Daher verhalten sie sich beim Überschreiten der Datumsgrenze, der Pole und bei großen Geometrien oder Geometriepaaren, die mehr als eine UTM-Zone abdecken, möglicherweise nicht wie erwartet. Basistransformation - (bevorzugt UTM, Lambert Azimutal (Nord/Süd), und im schlimmsten Fall Rückgriff auf Mercator)

---

- **ST\_Area** - Gibt den geometrischen Schwerpunkt einer Geometrie zurück.
  - **ST\_AsBinary** - Rückgabe der OGC/ISO Well-Known Binary (WKB)-Darstellung der Geometrie/Geografie ohne SRID-Metadaten.
  - **ST\_AsEWKT** - Gibt die Well-known-Text(WKT) Darstellung der Geometrie mit den SRID-Metadaten zurück.
  - **ST\_AsGML** - Gibt die Geometrie als GML-Element - Version 2 oder 3 - zurück.
  - **ST\_AsGeoJSON** - Rückgabe einer Geometrie oder eines Merkmals im GeoJSON-Format.
  - **ST\_AsKML** - Gibt die Geometrie als GML-Element - Version 2 oder 3 - zurück.
  - **ST\_AsSVG** - Gibt eine Geometrie als SVG-Pfad aus.
  - **ST\_AsText** - Gibt die Well-known-Text(WKT) Darstellung der Geometrie/Geographie ohne die SRID Metadaten zurück.
-

- **ST\_Azimuth** - Gibt die 2-dimensionale kürzeste Strecke zwischen zwei Geometrien als Linie zurück
  - **ST\_Buffer** - Berechnet eine Geometrie, die alle Punkte innerhalb eines bestimmten Abstands zu einer Geometrie umfasst.
  - **ST\_Centroid** - Gibt den geometrischen Schwerpunkt einer Geometrie zurück.
  - **ST\_ClosestPoint** - Gibt den 2D-Punkt auf g1 zurück, der g2 am nächsten ist. Dies ist der erste Punkt der kürzesten Linie von einer Geometrie zur anderen.
  - **ST\_CoveredBy** - Prüft, ob jeder Punkt von A in B liegt
  - **ST\_Covers** - Prüft, ob jeder Punkt von B in A liegt
  - **ST\_DWithin** - Prüft, ob zwei Geometrien innerhalb eines bestimmten Abstands liegen
  - **ST\_Distance** - Gibt die größte 3-dimensionale Distanz zwischen zwei geometrischen Objekten als Linie zurück
  - **ST\_GeogFromText** - Gibt einen geographischen Datentyp aus einer Well-known-Text (WKT), oder einer erweiterten WKT (EWKT), Darstellung zurück.
  - **ST\_GeogFromWKB** - Erzeugt ein geographisches Objekt aus der Well-known-Binary (WKB) oder der erweiterten Well-known-Binary (EWKB) Darstellung.
  - **ST\_GeographyFromText** - Gibt einen geographischen Datentyp aus einer Well-known-Text (WKT), oder einer erweiterten WKT (EWKT), Darstellung zurück.
  - **=** - Gibt TRUE zurück, wenn die Koordinaten und die Reihenfolge der Koordinaten der Geometrie/Geographie A und der Geometrie/Geographie B ident sind.
  - **ST\_Intersection** - Berechnet eine Geometrie, die den gemeinsamen Teil der Geometrien A und B darstellt.
  - **ST\_Intersects** - Prüft, ob sich zwei Geometrien schneiden (sie haben mindestens einen Punkt gemeinsam)
  - **ST\_Length** - Gibt den geometrischen Schwerpunkt einer Geometrie zurück.
  - **ST\_LineInterpolatePoint** - Gibt einen oder mehrere, entlang einer Linie interpolierte Punkte zurück.
  - **ST\_LineInterpolatePoints** - Gibt einen oder mehrere, entlang einer Linie interpolierte Punkte zurück.
  - **ST\_LineLocatePoint** - Liefert die gebrochene Position des Punktes auf einer Linie, der einem Punkt am nächsten liegt.
  - **ST\_LineSubstring** - Gibt den Teil einer Linie zwischen zwei gebrochenen Stellen zurück.
  - **ST\_Perimeter** - Gibt die Länge der Begrenzung einer polygonalen Geometrie oder Geografie zurück.
  - **ST\_Project** - Gibt einen Punkt zurück, der von einem Startpunkt um eine bestimmte Entfernung und Peilung (Azimut) projiziert wird.
  - **ST\_Segmentize** - Gibt eine geänderte Geometrie/Geografie zurück, bei der kein Segment länger als eine bestimmte Entfernung ist.
  - **ST\_ShortestLine** - Gibt die 2-dimensionale kürzeste Strecke zwischen zwei Geometrien als Linie zurück
  - **ST\_Summary** - Gibt eine Zusammenfassung des Inhalts einer Geometrie wieder.
  - **<->** - Gibt die 2D Entfernung zwischen A und B zurück.
  - **&&** - Gibt TRUE zurück, wenn die 2D Bounding Box von A die 2D Bounding Box von B schneidet.
-

## 13.5 PostGIS-Funktionen zur Unterstützung von Rastern

Die unten aufgeführten Funktionen und Operatoren sind PostGIS-Funktionen/Operatoren, die als Eingabe ein Objekt des Datentyps **raster** annehmen oder als Ausgabe zurückgeben. Sie sind in alphabetischer Reihenfolge aufgelistet.

- **Box3D** - Stellt das umschreibende Rechteck eines Raster als Box3D dar.
  - **@** - Gibt TRUE zurück, wenn das umschreibende Rechteck von A in jenem von B enthalten ist. Das umschreibende Rechteck ist in Double Precision.
  - **~** - Gibt TRUE zurück, wenn das umschreibende Rechteck von A jenes von B enthält. Das umschreibende Rechteck ist in Double Precision.
  - **=** - Gibt TRUE zurück, wenn die umschreibenden Rechtecke von A und B ident sind. Das umschreibende Rechteck ist in Double Precision.
  - **&&** - Gibt TRUE zurück, wenn das umschreibende Rechteck von A das umschreibende Rechteck von B schneidet.
  - **&<** - Gibt TRUE zurück, wenn das umschreibende Rechteck von A links von dem von B liegt.
  - **&>** - Gibt TRUE zurück, wenn das umschreibende Rechteck von A rechts von dem von B liegt.
  - **~=** - Gibt TRUE zurück, wenn die bounding box von A ident mit jener von B ist.
  - **ST\_Retile** - Gibt konfigurierte Kacheln eines beliebig gekachelten Rastercoverage aus.
  - **ST\_AddBand** - Gibt einen Raster mit den neu hinzugefügten Band(Bändern) aus. Der Typ, der Ausgangswert und der Index für den Speicherort des Bandes kann angegeben werden. Wenn kein Index angegeben ist, wird das Band am Ende hinzugefügt.
  - **ST\_AsBinary/ST\_AsWKB** - Gibt die Well-known-Binary (WKB) Darstellung eines Rasters zurück.
  - **ST\_AsGDALRaster** - Gibt die Rasterkachel in dem ausgewiesenen Rasterformat von GDAL aus. Sie können jedes Rasterformat angeben, das von Ihrer Bibliothek unterstützt wird. Um eine Liste mit den unterstützten Formaten auszugeben, verwenden Sie bitte `ST_GDALDrivers()`.
  - **ST\_AsHexWKB** - Gibt die Well-known-Binary (WKB) Hex-Darstellung eines Rasters zurück.
  - **ST\_AsJPEG** - Gibt die ausgewählten Bänder der Rasterkachel als einzelnes Bild (Byte-Array) im Format "Joint Photographic Exports Group" (JPEG) aus. Wenn kein Band angegeben ist und 1 oder mehr als 3 Bänder ausgewählt wurden, dann wird nur das erste Band verwendet. Wenn 3 Bänder ausgewählt wurden, werden alle 3 Bänder verwendet und auf RGB abgebildet.
  - **ST\_AsPNG** - Gibt die ausgewählten Bänder der Rasterkachel als einzelnes, übertragbares Netzwerkgraphik (PNG) Bild (Byte-Feld) aus. Wenn der Raster 1,3 oder 4 Bänder hat und keine Bänder angegeben sind, dann werden alle Bänder verwendet. Wenn der Raster 2 oder mehr als 4 Bänder hat und keine Bänder angegeben sind, dann wird nur Band 1 verwendet. Die Bänder werden in den RGB- oder den RGBA-Raum abgebildet.
  - **ST\_AsRaster** - Konvertiert den geometrischen Datentyp von PostGIS in einen PostGIS Raster.
  - **ST\_AsTIFF** - Gibt die ausgewählten Bänder des Raster als einzelnes TIFF Bild (Byte-Feld) zurück. Wenn kein Band angegeben ist oder keines der angegebenen Bänder im Raster existiert, werden alle Bänder verwendet.
  - **ST\_Aspect** - Gibt die Exposition (standardmäßig in Grad) eines Rasterbandes mit Höhen aus. Nützlich für Terrain-Analysen.
  - **ST\_Band** - Gibt einen oder mehrere Bänder eines bestehenden Rasters als neuen Raster aus. Nützlich um neue Raster aus bestehenden Rastern abzuleiten.
  - **ST\_BandFileSize** - Gibt die Dateigröße eines im Dateisystem gespeicherten Bandes aus. Wenn "bandnum" nicht angegeben ist, wird 1 angenommen.
  - **ST\_BandFileTimestamp** - Gibt den Zeitstempel eines im Dateisystem gespeicherten Bandes aus. Wenn "bandnum" nicht angegeben ist, wird 1 angenommen.
  - **ST\_BandIsNoData** - Gibt TRUE aus, wenn das Band ausschließlich aus NODATA Werten besteht.
-

- **ST\_BandMetaData** - Gibt die grundlegenden Metadaten eines bestimmten Rasterbandes aus. Wenn der Parameter "bandnum" nicht angegeben ist, wird das 1ste Band angenommen.
  - **ST\_BandNoDataValue** - Gibt den NODATA Wert des gegebenen Bandes aus. Wenn der Parameter "bandnum" nicht angegeben ist, wird das 1ste Band angenommen.
  - **ST\_BandPath** - Gibt den Dateipfad aus, unter dem das Band im Dateisystem gespeichert ist. Wenn "bandnum" nicht angegeben ist, wird 1 angenommen.
  - **ST\_BandPixelType** - Gibt den Pixeltyp des angegebenen Bandes aus. Wenn der Parameter "bandnum" nicht angegeben ist, wird das 1ste Band angenommen.
  - **ST\_Clip** - Returns the raster clipped by the input geometry. If band number is not specified, all bands are processed. If crop is not specified or TRUE, the output raster is cropped. If touched is set to TRUE, then touched pixels are included, otherwise only if the center of the pixel is in the geometry it is included.
  - **ST\_ColorMap** - Erzeugt aus einem bestimmten Band des Ausgangsrasters einen neuen Raster mit bis zu vier 8BUI-Bändern (Grauwert, RGB, RGBA). Wenn kein Band angegeben ist, wird Band 1 angenommen.
  - **ST\_Contains** - Gibt TRUE zurück, wenn kein Punkt des Rasters "rastB" im Äußeren des Rasters "rastA" liegt und zumindest ein Punkt im Inneren von "rastB" auch im Inneren von "rastA" liegt.
  - **ST\_ContainsProperly** - Gibt TRUE zurück, wenn "rastB" das Innere von "rastA" schneidet, aber nicht die Begrenzung oder das Äußere von "rastA".
  - **ST\_Contour** - Erzeugt einen Satz von Vektorkonturen aus dem angegebenen Rasterband unter Verwendung des GDAL-Konturierungsalgorithmus.
  - **ST\_ConvexHull** - Gibt die Geometrie der konvexen Hülle des Raster, inklusive der Pixel deren Werte gleich BandNoDataValue sind. Bei regelmäßig geformten und nicht rotierten Raster ist das Ergebnis ident mit ST\_Envelope. Diese Funktion ist deshalb nur bei unregelmäßig geformten oder rotierten Raster nützlich.
  - **ST\_Count** - Gibt die Anzahl der Pixel für ein Band eines Rasters oder eines Raster-Coverage zurück. Wenn kein Band angegeben ist, wird standardmäßig Band 1 gewählt. Wenn der Parameter "exclude\_nodata\_value" auf TRUE gesetzt ist, werden nur Pixel mit Werten ungleich NODATA gezählt.
  - **ST\_CountAgg** - Aggregatfunktion. Gibt die Anzahl der Pixel in einem bestimmten Band der Raster aus. Wenn kein Band angegeben ist, wird Band 1 angenommen. Wenn "exclude\_nodata\_value" TRUE ist, werden nur die Pixel ohne NODATA Werte gezählt.
  - **ST\_CoveredBy** - Gibt TRUE zurück, wenn kein Punkt des Rasters "rastA" außerhalb des Rasters "rastB" liegt.
  - **ST\_Covers** - Gibt TRUE zurück, wenn kein Punkt des Rasters "rastB" außerhalb des Rasters "rastA" liegt.
  - **ST\_DFullyWithin** - Gibt TRUE zurück, wenn die Raster "rastA" und "rastB" zur Gänze innerhalb der angegebenen Distanz zueinander liegen.
  - **ST\_DWithin** - Gibt TRUE zurück, wenn die Raster "rastA" und "rastB" innerhalb der angegebenen Entfernung voneinander liegen.
  - **ST\_Disjoint** - Gibt TRUE zurück, wenn sich die Raster "rastA" und "rastB" räumlich nicht überschneiden.
  - **ST\_DumpAsPolygons** - Gibt geomval (geom,val) Zeilen eines Rasterbandes zurück. Wenn kein Band angegeben ist, wird die Bandnummer standardmäßig auf 1 gesetzt.
  - **ST\_DumpValues** - Gibt die Werte eines bestimmten Bandes als 2-dimensionales Feld aus.
  - **ST\_Envelope** - Stellt die Ausdehnung des Raster als Polygon dar.
  - **ST\_FromGDALRaster** - Erzeugt einen Raster aus einer von GDAL unterstützten Rasterdatei.
  - **ST\_GeoReference** - Gibt die Metadaten der Georeferenzierung, die sich üblicherweise in einem sogenannten "World File" befinden, im GDAL oder ESRI Format aus. Die Standardeinstellung ist GDAL.
-



- **ST\_Grayscale** - Erzeugt einen neuen Raster mit einem 8BUI-Band aus dem Ausgangsraster und den angegebenen Bändern für Rot, Grün und Blau
  - **ST\_HasNoBand** - Gibt TRUE aus, wenn kein Band mit der angegebenen Bandnummer existiert. Gibt den Pixeltyp des angegebenen Bandes aus. Wenn keine Bandnummer angegeben ist, wird das 1ste Band angenommen.
  - **ST\_Height** - Gibt die Höhe des Rasters in Pixel aus.
  - **ST\_HillShade** - Gibt für gegebenen Horizontalwinkel, Höhenwinkel, Helligkeit und Maßstabsverhältnis die hypothetische Beleuchtung eines Höhenrasterbandes zurück.
  - **ST\_Histogram** - Gibt Datensätze aus, welche die Verteilung der Daten eines Rasters oder eines Rastercoverage darstellen. Dabei wird die Wertemenge in Klassen aufgeteilt und für jede Klasse zusammengefasst. Wenn die Anzahl der Klassen nicht angegeben ist, wird sie automatisch berechnet.
  - **ST\_InterpolateRaster** - Interpoliert eine gerasterte Oberfläche auf der Grundlage eines Eingabesatzes von 3D-Punkten, wobei die X- und Y-Werte zur Positionierung der Punkte auf dem Gitter und der Z-Wert der Punkte als Oberflächenhöhe verwendet werden.
  - **ST\_Intersection** - Gibt Geometry-PixelValue Paare, oder einen Raster aus, der durch die Schnittmenge der beiden Raster bestimmt wird, oder durch die geometrische Verschneidung einer Vektorisierung des Rasters mit einem geometrischen Datentyp.
  - **ST\_Intersects** - Gibt TRUE zurück, wenn sich die Raster "rastA" und "rastB" nicht räumlich überschneiden.
  - **ST\_IsEmpty** - Gibt TRUE zurück, wenn der Raster leer ist (width = 0 and height = 0). Andernfalls wird FALSE zurückgegeben.
  - **ST\_MakeEmptyCoverage** - Bedeckt die georeferenzierte Fläche mit einem Gitter aus leeren Rasterkacheln.
  - **ST\_MakeEmptyRaster** - Gibt einen leeren Raster (ohne Bänder), mit den gegebenen Dimensionen (width & height), upperleft X und Y, Pixelgröße, Rotation (scalex, scaley, skewx & skewy) und Koordinatenreferenzsystem (SRID), zurück. Wenn ein Raster übergeben wird, dann wird ein neuer Raster mit der selben Größe, Ausrichtung und SRID zurückgegeben. Wenn SRID nicht angegeben ist, wird das Koordinatenreferenzsystem auf "unknown" (0) gesetzt.
  - **ST\_MapAlgebra (callback function version)** - Die Version mit der Rückruffunktion - Gibt für einen oder mehrere Eingaberaster einen Raster mit einem Band, den Bandindizes und einer vom Anwender vorgegebenen Rückruffunktion zurück.
  - **ST\_MapAlgebraExpr** - Version mit 1 Rasterband: Erzeugt ein neues Rasterband, dass über eine gültige, algebraische PostgreSQL Operation für ein Rasterband mit gegebenen Pixeltyp erstellt wird. Wenn kein Band bestimmt ist, wird Band 1 angenommen.
  - **ST\_MapAlgebraExpr** - Version mit 2 Rasterbändern: Erstellt einen neuen Einzelbandraster, indem eine gültige algebraische PostgreSQL Funktion auf die zwei Ausgangsrasterbänder und den entsprechenden Pixeltyp angewendet wird. Wenn keine Bandnummern angegeben sind, wird von jedem Raster Band 1 angenommen. Der Ergebnisraster wird nach dem Gitter des ersten Raster ausgerichtet (Skalierung, Versatz und Eckpunkte der Pixel) und hat die Ausdehnung, welche durch den Parameter "extenttype" definiert ist. Der Parameter "extenttype" kann die Werte INTERSECTION, UNION, FIRST, SECOND annehmen.
  - **ST\_MapAlgebraFct** - Version mit 1 Rasterband: Erzeugt ein neues Rasterband, dass über eine gültige PostgreSQL Funktion für ein gegebenes Rasterband und Pixeltyp erstellt wird. Wenn kein Band bestimmt ist, wird Band 1 angenommen.
  - **ST\_MapAlgebraFct** - Version mit 2 Rasterbändern: Erstellt einen neuen Einzelbandraster, indem eine gültige PostgreSQL Funktion auf die 2 gegebenen Rasterbänder und den entsprechenden Pixeltyp angewendet wird. Wenn kein Band bestimmt ist, wird Band 1 angenommen. Wenn der "Extent"-Typ nicht angegeben ist, wird standardmäßig INTERSECTION angenommen.
  - **ST\_MapAlgebraFctNgb** - Version mit 1em Band: Map Algebra Nearest Neighbor mit einer benutzerdefinierten PostgreSQL Funktion. Gibt einen Raster zurück, dessen Werte sich aus einer benutzerdefinierte PL/pgsql Funktion ergeben, welche die Nachbarschaftswerte des Ausgangsrasterbandes einbezieht.
  - **ST\_MapAlgebra (expression version)** - Version mit Ausdrücken - Gibt für einen oder zwei Ausgangsraster, Bandindizes und einer oder mehreren vom Anwender vorgegebenen SQL-Ausdrücken, einen Raster mit einem Band zurück.
  - **ST\_MemSize** - Gibt den Platzbedarf des Rasters (in Byte) aus.
  - **ST\_MetaData** - Gibt die wesentlichen Metadaten eines Rasterobjektes, wie Zellgröße, Rotation (Versatz) etc. aus
-

- **ST\_MinConvexHull** - Gibt die Geometrie der konvexen Hülle des Raster aus, wobei Pixel mit NODATA ausgenommen werden.
  - **ST\_NearestValue** - Gibt den nächstgelegenen nicht NODATA Wert eines bestimmten Pixels aus, das über "columnx" und "rowy" oder durch eine Punktgeometrie - im gleichen Koordinatenreferenzsystem wie der Raster - ausgewählt wird.
  - **ST\_Neighborhood** - Gibt ein 2-D Feld in "Double Precision" aus, das sich aus nicht NODATA Werten um ein bestimmtes Pixel herum zusammensetzt. Das Pixel Kann über "columnx" und "rowy" oder über eine Punktgeometrie - im gleichen Koordinatenreferenzsystem wie der Raster - ausgewählt werden.
  - **ST\_NotSameAlignmentReason** - Gibt eine Meldung aus, die angibt ob die Raster untereinander ausgerichtet sind oder nicht und warum wenn nicht.
  - **ST\_NumBands** - Gibt die Anzahl der Bänder des Rasters aus.
  - **ST\_Overlaps** - Gibt TRUE zurück, wenn sich die Raster "rastA" und "rastB" schneiden, aber ein Raster den anderen nicht zur Gänze enthält.
  - **ST\_PixelAsCentroid** - Gibt den geometrischen Schwerpunkt (Punktgeometrie) der Fläche aus, die durch das Pixel repräsentiert wird.
  - **ST\_PixelAsCentroids** - Gibt den geometrischen Schwerpunkt (Punktgeometrie) für jedes Pixel des Rasterbandes zurück, zusammen mit dem Zellwert und den X- und Y-Rasterkoordinaten eines jeden Pixels. Die Koordinaten der Punkte entsprechen dem geometrischen Schwerpunkt der Pixel.
  - **ST\_PixelAsPoint** - Gibt eine Punktgeometrie der oberen linken Ecke des Rasters zurück.
  - **ST\_PixelAsPoints** - Gibt eine Punktgeometrie für jedes Pixel des Rasterbandes zurück, zusammen mit dem Zellwert und den X- und Y-Rasterkoordinaten eines jeden Pixels. Die Koordinaten der Punkte entsprechen dem oberen linken Eck der Pixel.
  - **ST\_PixelAsPolygon** - Gibt die Polyongeometrie aus, die das Pixel einer bestimmten Zeile und Spalte begrenzt.
  - **ST\_PixelAsPolygons** - Gibt die umhüllende Polyongeometrie, den Zellwert, sowie die X- und Y-Rasterkoordinate für jedes Pixel aus.
  - **ST\_PixelHeight** - Gibt die Pixelhöhe in den Einheiten des Koordinatenreferenzsystem aus.
  - **ST\_PixelOfValue** - Gibt die columnx- und rowy-Koordinaten jener Pixel aus, deren Zellwert gleich dem gesuchten Wert ist.
  - **ST\_PixelWidth** - Gibt die Pixelbreite in den Einheiten des Koordinatenreferenzsystems aus.
  - **ST\_Polygon** - Gibt eine Geometrie mit Mehrfachpolygonen zurück, die aus der Vereinigung von Pixel mit demselben Zellwert gebildet werden. Pixel mit NODATA Werten werden nicht berücksichtigt. Wenn keine Band angegeben ist, wird die Bandnummer standardmäßig auf 1 gesetzt.
  - **ST\_Quantile** - Berechnet die Quantile eines Rasters oder einer Rastercoverage Tabelle im Kontext von Stichproben oder Bevölkerung. Dadurch kann untersucht werden, ob ein Wert bei 25%, 50% oder 75% Perzentil des Rasters liegt.
  - **ST\_RastFromHexWKB** - Gibt einen Rasterwert von einer Well-known-Binary (WKB) Hex-Darstellung eines Rasters zurück.
  - **ST\_RastFromWKB** - Gibt einen Rasterwert von einer Well-known-Binary (WKB) Darstellung eines Rasters zurück.
  - **ST\_RasterToWorldCoord** - Gibt die obere linke Ecke des Rasters in geodätischem X und Y (Länge und Breite) für eine gegebene Spalte und Zeile aus. Spalte und Zeile wird von 1 aufwärts gezählt.
  - **ST\_RasterToWorldCoordX** - Gibt die geodätische X Koordinate links oberhalb des Rasters, der Spalte und der Zeile aus. Die Nummerierung der Spalten und Zeilen beginnt mit 1.
  - **ST\_RasterToWorldCoordY** - Gibt die geodätische Y Koordinate links oberhalb des Rasters, der Spalte und der Zeile aus. Die Nummerierung der Spalten und Zeilen beginnt mit 1.
  - **ST\_Reclass** - Erstellt einen neuen Raster, der aus neu klassifizierten Bändern des Originalraster besteht. Das Band "nband" ist jenes das verändert werden soll. Wenn "nband" nicht angegeben ist, wird "Band 1" angenommen. Alle anderen Bänder bleiben unverändert. Anwendungsfall: zwecks einfacherer Visualisierung ein 16BUI-Band in ein 8BUI-Band konvertieren und so weiter.
-



- **ST\_Resample** - Skaliert einen Raster mit einem bestimmten Algorithmus, neuen Dimensionen, einer beliebigen Gitterecke und über Parameter zur Georeferenzierung des Rasters, die angegeben oder von einem anderen Raster übernommen werden können.
  - **ST\_Rescale** - Neuabtastung eines Rasters, indem nur die Skala (oder Pixelgröße) angepasst wird. Die neuen Pixelwerte werden mit den Algorithmen NearestNeighbor (englische oder amerikanische Schreibweise), Bilinear, Cubic, CubicSpline, Lanczos, Max oder Min resampling berechnet. Die Voreinstellung ist NearestNeighbor.
  - **ST\_Resize** - Ändert die Zellgröße - width/height - eines Rasters
  - **ST\_Reskew** - Skaliert einen Raster, indem lediglich der Versatz (oder Rotationsparameter) angepasst wird. Neue Pixelwerte werden über NearestNeighbor, bilinear, kubisch, CubicSpline oder mit dem Lanczos-Filter errechnet. Die Standardeinstellung ist NearestNeighbor.
  - **ST\_Rotation** - Gibt die Rotation des Rasters im Bogenmaß aus.
  - **ST\_Roughness** - Gibt einen Raster mit der berechneten "Rauhigkeit" des DHM zurück.
  - **ST\_SRID** - Gibt den Identifikator des Koordinatenreferenzsystems des Rasters aus, das in der Tabelle "spatial\_ref\_sys" definiert ist.
  - **ST\_SameAlignment** - Gibt TRUE zurück, wenn die Raster die selbe Rotation, Skalierung, Koordinatenreferenzsystem und Versatz (Pixel können auf dasselbe Gitter gelegt werden, ohne dass die Gitterlinien durch die Pixel schneiden) aufweisen. Wenn nicht, wird FALSE und eine Beschreibung des Problems ausgegeben.
  - **ST\_ScaleX** - Gibt die X-Komponente der Pixelbreite in den Einheiten des Koordinatenreferenzsystems aus.
  - **ST\_ScaleY** - Gibt die Y-Komponente der Pixelhöhe in den Einheiten des Koordinatenreferenzsystems aus.
  - **ST\_SetBandIndex** - Aktualisiert die externe Bandnummer eines out-db Bandes.
  - **ST\_SetBandIsNoData** - Setzt die Flag "isnodata" für das Band auf TRUE.
  - **ST\_SetBandNoDataValue** - Setzt den NODATA Wert eines Bandes. Wenn kein Band angegeben ist, wird Band 1 angenommen. Falls ein Band keinen NODATA Wert aufweisen soll, übergeben Sie bitte für den Parameter "nodatavalue" NULL.
  - **ST\_SetBandPath** - Aktualisiert den externen Dateipfad und die Bandnummer eines out-db Bandes.
  - **ST\_SetGeoReference** - Georeferenziert einen Raster über 6 Parameter in einem einzigen Aufruf. Die Zahlen müssen durch Leerzeichen getrennt sein. Die Funktion akzeptiert die Eingabe im Format von 'GDAL' und von 'ESRI'. Der Standardwert ist GDAL.
  - **ST\_SetM** - Gibt eine Geometrie mit denselben X/Y-Koordinaten wie die Eingabegeometrie zurück, wobei die Werte aus dem Raster mit dem gewünschten Resample-Algorithmus in die Dimension M kopiert werden.
  - **ST\_SetRotation** - Bestimmt die Rotation des Rasters in Radiant.
  - **ST\_SetSRID** - Setzt die SRID eines Rasters auf einen bestimmten Ganzzahlwert. Die SRID wird in der Tabelle "spatial\_ref\_sys" definiert.
  - **ST\_SetScale** - Setzt die X- und Y-Größe der Pixel in den Einheiten des Koordinatenreferenzsystems. Entweder eine Zahl pro Pixel oder Breite und Höhe.
  - **ST\_SetSkew** - Setzt den georeferenzierten X- und Y-Versatz (oder den Rotationsparameter). Wenn nur ein Wert übergeben wird, werden X und Y auf den selben Wert gesetzt.
  - **ST\_SetUpperLeft** - Setzt den Wert der oberen 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\_SetZ** - Gibt eine Geometrie mit denselben X/Y-Koordinaten wie die Eingabegeometrie zurück, wobei die Werte aus dem Raster mit dem gewünschten Resample-Algorithmus in die Z-Dimension kopiert werden.
  - **ST\_SkewX** - Gibt den georeferenzierten Versatz in X-Richtung (oder den Rotationsparameter) aus.
  - **ST\_SkewY** - Gibt den georeferenzierten Versatz in Y-Richtung (oder den Rotationsparameter) aus.
  - **ST\_Slope** - Gibt die Neigung (standardmäßig in Grad) eines Höhenrasterbandes zurück. Nützlich für Terrain-Analysen.
  - **ST\_SnapToGrid** - Skaliert einen Raster durch Fangen an einem Führungsgitter. Neue Pixelwerte werden über NearestNeighbor, bilinear, kubisch, CubicSpline oder mit dem Lanczos-Filter errechnet. Die Standardeinstellung ist NearestNeighbor.
  - **ST\_Summary** - Gibt eine textliche Zusammenfassung des Rasterinhalts zurück.
  - **ST\_SummaryStats** - Gibt eine zusammenfassende Statistik aus, bestehend aus der Anzahl, der Summe, dem arithmetischen Mittel, der Standardabweichung, dem Minimum und dem Maximum der Werte eines Rasterbandes oder eines Rastercoverage. Wenn kein Band angegeben ist, wird Band 1 angenommen.
  - **ST\_SummaryStatsAgg** - Aggregatfunktion. Gibt eine zusammenfassende Statistik aus, die aus der Anzahl, der Summe, dem arithmetischen Mittel, dem Minimum und dem Maximum der Werte eines bestimmten Bandes eines Rastersatzes besteht. Wenn kein Band angegeben ist, wird Band 1 angenommen.
  - **ST\_TPI** - Berechnet den "Topographic Position Index" eines Raster.
  - **ST\_TRI** - Gibt einen Raster mit errechneten Geländerauheitsindex aus.
  - **ST\_Tile** - Gibt Raster, die aus einer Teilungsoperation des Eingaberasters resultieren, mit den gewünschten Dimensionen aus.
  - **ST\_Touches** - Gibt TRUE zurück, wenn rastA und rastB zumindest einen Punkt gemeinsam haben sich aber nicht überschneiden.
  - **ST\_Transform** - Projiziert einen Raster von einem bekannten Koordinatenreferenzsystem in ein anderes bekanntes Koordinatenreferenzsystem um. Die Optionen für die Skalierung sind NearestNeighbor, Bilinear, Cubisch, CubicSpline und der Lanczos-Filter, die Standardeinstellung ist NearestNeighbor.
  - **ST\_Union** - Gibt die Vereinigung mehrerer Rasterkacheln in einem einzelnen Raster mit mehreren Bändern zurück.
  - **ST\_UpperLeftX** - Gibt die obere linke X-Koordinate des Rasters im Koordinatenprojektionssystem aus.
  - **ST\_UpperLeftY** - Gibt die obere linke Y-Koordinate des Rasters im Koordinatenprojektionssystem aus.
  - **ST\_Value** - Gibt den Zellwert eines Pixels aus, das über columnx und rowy oder durch einen bestimmten geometrischen Punkt angegeben wird. Die Bandnummern beginnen mit 1 und wenn keine Bandnummer angegeben ist, dann wird Band 1 angenommen. Wenn exclude\_nodata\_value auf FALSE gesetzt ist, werden auch die Pixel mit einem nodata Wert mit einbezogen. Wenn exclude\_nodata\_value nicht übergeben wird, dann wird er über die Metadaten des Rasters ausgelesen.
  - **ST\_ValueCount** - Gibt Datensätze aus, die den Zellwert und die Anzahl der Pixel eines Rasterbandes (oder 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.
-

## 13.6 PostGIS Geometrie / Geographie / Raster Dump Funktionen

Die nachstehend aufgeführten Funktionen sind PostGIS-Funktionen, die als Eingabe einen Satz oder ein einzelnes `geometry_dump` oder `geomval` Objekt vom Datentyp annehmen oder als Ausgabe zurückgeben.

- **ST\_DumpAsPolygons** - Gibt `geomval` (`geom, val`) Zeilen eines Rasterbandes zurück. Wenn kein Band angegeben ist, wird die Bandnummer standardmäßig auf 1 gesetzt.
- **ST\_Intersection** - Gibt Geometry-PixelValue Paare, oder einen Raster aus, der durch die Schnittmenge der beiden Raster bestimmt wird, oder durch die geometrische Verschneidung einer Vektorisierung des Rasters mit einem geometrischen Datentyp.
- **ST\_Dump** - Gibt einen Satz von `geometry_dump` Zeilen für die Komponenten einer Geometrie zurück.
- **ST\_DumpPoints** - Gibt eine Zusammenfassung des Inhalts einer Geometrie wieder.
- **ST\_DumpRings** - Gibt einen Satz von `geometry_dump` Zeilen für die äußeren und inneren Ringe eines Polygons zurück.
- **ST\_DumpSegments** - Gibt eine Zusammenfassung des Inhalts einer Geometrie wieder.

## 13.7 PostGIS-Box-Funktionen

Bei den nachstehenden Funktionen handelt es sich um PostGIS-Funktionen, die als Eingabe die `Box*`-Familie von PostGIS-Geodatentypen verwenden oder als Ausgabe zurückgeben. Die Familie der `Box`-Typen besteht aus `box2d` und `box3d`

- **Box2D** - Gibt ein `BOX2D` zurück, das die 2D-Ausdehnung einer Geometrie darstellt.
- **Box3D** - Gibt ein `BOX3D` zurück, das die 3D-Ausdehnung einer Geometrie darstellt.
- **Box3D** - Stellt das umschreibende Rechteck eines Raster als `Box3D` dar.
- **ST\_3DExtent** - Aggregatfunktion, die den 3D-Begrenzungsrahmen von Geometrien zurückgibt.
- **ST\_3DMakeBox** - Erzeugt einen `BOX3D`, der durch zwei 3D-Punktgeometrien definiert ist.
- **ST\_AsMVTGeom** - Transformiert eine Geometrie in den Koordinatenraum einer MVT-Kachel.
- **ST\_AsTWKB** - Gibt die Geometrie als TWKB, aka "Tiny Well-known Binary" zurück
- **ST\_Box2dFromGeoHash** - Gibt die `BOX2D` einer GeoHash Zeichenkette zurück.
- **ST\_ClipByBox2D** - Berechnet den Teil einer Geometrie, der innerhalb eines Rechtecks liegt.
- **ST\_EstimatedExtent** - Gibt die geschätzte Ausdehnung einer räumlichen Tabelle zurück.
- **ST\_Expand** - Gibt einen Begrenzungsrahmen zurück, der aus einem anderen Begrenzungsrahmen oder einer Geometrie erweitert wurde.
- **ST\_Extent** - Aggregatfunktion, die den Begrenzungsrahmen von Geometrien zurückgibt.
- **ST\_MakeBox2D** - Erzeugt ein `BOX2D`, das durch zwei 2D-Punktgeometrien definiert ist.
- **ST\_RemoveIrrelevantPointsForView** - Removes points that are irrelevant for rendering a specific rectangular view of a geometry.
- **ST\_XMax** - Gibt die X-Maxima eines 2D- oder 3D-Begrenzungsrahmens oder einer Geometrie zurück.
- **ST\_XMin** - Gibt die X-Minima eines 2D- oder 3D-Begrenzungsrahmens oder einer Geometrie zurück.
- **ST\_YMax** - Gibt die Y-Maxima eines 2D- oder 3D-Begrenzungsrahmens oder einer Geometrie zurück.
- **ST\_YMin** - Gibt die Y-Minima eines 2D- oder 3D-Begrenzungsrahmens oder einer Geometrie zurück.
- **ST\_ZMax** - Gibt die Z-Maxima eines 2D- oder 3D-Begrenzungsrahmens oder einer Geometrie zurück.

- **ST\_ZMin** - Gibt die Z-Minima eines 2D- oder 3D-Begrenzungsrahmens oder einer Geometrie zurück.
- **RemoveUnusedPrimitives** - Entfernt Topologieprimitive, die zur Definition bestehender TopoGeometry-Objekte nicht benötigt werden.
- **ValidateTopology** - Liefert eine Menge validate\_topology\_return\_type Objekte, die Probleme mit der Topologie beschreiben.
- **~(box2df,box2df)** - Gibt TRUE zurück, wenn eine 2D float precision bounding box (BOX2DF) eine andere 2D float precision bounding box (BOX2DF) enthält.
- **~(box2df,geometry)** - Gibt TRUE zurück, wenn eine 2D float precision bounding box (BOX2DF) die 2D Bounding Box einer Geometrie enthält.
- **~(geometry,box2df)** - Gibt TRUE zurück, wenn die 2D bounding box einer Geometrie eine 2D float precision bounding box (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 Bounding Box (BOX2DF) enthalten ist.
- **&&(box2df,box2df)** - Gibt TRUE zurück, wenn sich zwei 2D float precision Bounding Boxes (BOX2DF) überschneiden.
- **&&(box2df,geometry)** - Gibt TRUE zurück, wenn eine 2D float precision bounding box (BOX2DF) eine Geometrie (cached) 2D bounding box schneidet.
- **&&(geometry,box2df)** - Gibt TRUE zurück, wenn sich die 2D Bounding Box (cached) einer Geometrie mit einer 2D Bounding Box mit Gleitpunktgenauigkeit (BOX2DF) überschneidet.

## 13.8 PostGIS-Funktionen, die 3D unterstützen

Die unten aufgeführten Funktionen sind PostGIS-Funktionen, die den Z-Index nicht wegwerfen.

- **AddGeometryColumn** - Entfernt eine Geometriespalte aus einer räumlichen Tabelle.
- **Box3D** - Gibt ein BOX3D zurück, das die 3D-Ausdehnung einer Geometrie darstellt.
- **CG\_3DArea** - Berechnet die Fläche von 3D-Oberflächengeometrien. Gibt 0 für Solids zurück.
- **CG\_3DConvexHull** - Berechnet die konvexe Hülle einer Geometrie.
- **CG\_3DDifference** - 3D-Differenz durchführen
- **CG\_3DIntersection** - 3D-Schnitte durchführen
- **CG\_3DUnion** - Perform 3D union using postgis\_sfcgal.
- **CG\_ApproximateMedialAxis** - Berechnet die konvexe Hülle einer Geometrie.
- **CG\_ConstrainedDelaunayTriangles** - Gibt eine eingeschränkte Delaunay-Triangulation um die angegebene Eingabegeometrie zurück.
- **CG\_Extrude** - Extrudieren einer Oberfläche in ein zugehöriges Volumen
- **CG\_ForceLHR** - LHR-Ausrichtung erzwingen
- **CG\_IsPlanar** - Prüfen, ob eine Fläche planar ist oder nicht
- **CG\_IsSolid** - Prüfen, ob die Geometrie ein Solid ist. Es wird keine Gültigkeitsprüfung durchgeführt.

- **CG\_MakeSolid** - Gießen Sie die Geometrie in einen Körper. Es wird keine Prüfung durchgeführt. Um ein gültiges Solid zu erhalten, muss die Eingabegeometrie eine geschlossene polyedrische Fläche oder ein geschlossenes TIN sein.
  - **CG\_Orientation** - Bestimmung der Oberflächenausrichtung
  - **CG\_StraightSkeleton** - Berechnet die konvexe Hülle einer Geometrie.
  - **CG\_Tessellate** - Führt eine Oberflächentesselierung eines Polygons oder einer Polyederfläche durch und gibt diese als TIN oder Sammlung von TINS zurück
  - **CG\_Visibility** - Berechnen eines Sichtbarkeitspolygons aus einem Punkt oder einem Segment in einer Polygoneometrie
  - **CG\_Volume** - Berechnet das Volumen eines 3D-Volumens. Bei Anwendung auf (auch geschlossene) Flächengeometrien wird 0 zurückgegeben.
  - **DropGeometryColumn** - Entfernt eine Geometriespalte aus einer räumlichen Tabelle.
  - **GeometryType** - Gibt den Geometrietyp des ST\_Geometry Wertes zurück.
  - **ST\_3DArea** - Berechnet die Fläche von 3D-Oberflächengeometrien. Gibt 0 für Solids zurück.
  - **ST\_3DClosestPoint** - Gibt den 3-dimensionalen Punkt auf g1 zurück, der den kürzesten Abstand zu g2 hat. Dies ist der Anfangspunkt des kürzesten Abstands in 3D.
  - **ST\_3DConvexHull** - Berechnet die konvexe Hülle einer Geometrie.
  - **ST\_3DDFullyWithin** - Prüft, ob zwei 3D-Geometrien vollständig innerhalb eines bestimmten 3D-Abstands liegen
  - **ST\_3DDWithin** - Prüft, ob zwei 3D-Geometrien innerhalb eines bestimmten 3D-Abstands liegen
  - **ST\_3DDifference** - 3D-Differenz durchführen
  - **ST\_3DDistance** - Für den geometrischen Datentyp. Es wird der geringste 3-dimensionale kartesische Abstand (basierend auf dem Koordinatenreferenzsystem) zwischen zwei geometrischen Objekten in projizierten Einheiten zurückgegeben.
  - **ST\_3DExtent** - Aggregatfunktion, die den 3D-Begrenzungsrahmen von Geometrien zurückgibt.
  - **ST\_3DIntersection** - 3D-Schnitte durchführen
  - **ST\_3DIntersects** - Prüft, ob sich zwei Geometrien in 3D räumlich schneiden - nur für Punkte, Linienzüge, Polygone, polyedrische Flächen (Bereich)
  - **ST\_3DLength** - Gibt den geometrischen Schwerpunkt einer Geometrie zurück.
  - **ST\_3DLineInterpolatePoint** - Gibt einen oder mehrere, entlang einer Linie interpolierte Punkte zurück.
  - **ST\_3DLongestLine** - Gibt die größte 3-dimensionale Distanz zwischen zwei geometrischen Objekten als Linie zurück
  - **ST\_3DMaxDistance** - Für den geometrischen Datentyp. Gibt die maximale 3-dimensionale kartesische Distanz (basierend auf dem Koordinatenreferenzsystem) zwischen zwei geometrischen Objekten in projizierten Einheiten zurück.
  - **ST\_3DPerimeter** - Gibt den geometrischen Schwerpunkt einer Geometrie zurück.
  - **ST\_3DShortestLine** - Gibt den kürzesten 3-dimensionalen Abstand zwischen zwei geometrischen Objekten als Linie zurück
  - **ST\_3DUnion** - 3D-Vereinigung durchführen.
  - **ST\_AddMeasure** - Interpoliert Maße entlang einer linearen Geometrie.
  - **ST\_AddPoint** - Fügt einem Linienzug einen Punkt hinzu.
  - **ST\_Affine** - Wenden Sie eine affine 3D-Transformation auf eine Geometrie an.
  - **ST\_ApproximateMedialAxis** - Berechnet die konvexe Hülle einer Geometrie.
  - **ST\_AsBinary** - Rückgabe der OGC/ISO Well-Known Binary (WKB)-Darstellung der Geometrie/Geografie ohne SRID-Metadaten.
-

- **ST\_AsEWKB** - Rückgabe der Extended Well-Known Binary (EWKB) Darstellung der Geometrie mit SRID-Metadaten.
  - **ST\_AsEWKT** - Gibt die Well-known-Text(WKT) Darstellung der Geometrie mit den SRID-Metadaten zurück.
  - **ST\_AsGML** - Gibt die Geometrie als GML-Element - Version 2 oder 3 - zurück.
  - **ST\_AsGeoJSON** - Rückgabe einer Geometrie oder eines Merkmals im GeoJSON-Format.
  - **ST\_AsHEXEWKB** - Gibt eine Geometrie im HEXEWKB Format (als Text) aus; verwendet entweder die Little-Endian (NDR) oder die Big-Endian (XDR) Zeichenkodierung.
  - **ST\_AsKML** - Gibt die Geometrie als GML-Element - Version 2 oder 3 - zurück.
  - **ST\_AsX3D** - Gibt eine Geometrie im X3D XML Knotenelement-Format zurück: ISO-IEC-19776-1.2-X3DEncodings-XML
  - **ST\_Boundary** - Gibt die abgeschlossene Hülle aus der kombinierten Begrenzung der Geometrie zurück.
  - **ST\_BoundingDiagonal** - Gibt die Diagonale des Umgebungsdreiecks der angegebenen Geometrie zurück.
  - **ST\_CPAWithin** - Prüft, ob der nächstgelegene Punkt der Annäherung zweier Flughäfen innerhalb der angegebenen Entfernung liegt.
  - **ST\_ChaikinSmoothing** - Gibt eine geglättete Version einer Geometrie zurück, die den Chaikin-Algorithmus verwendet
  - **ST\_ClosestPointOfApproach** - Liefert ein Maß für den nächstgelegenen Punkt der Annäherung von zwei Flughäfen.
  - **ST\_Collect** - Erzeugt eine GeometryCollection oder Multi\*-Geometrie aus einer Reihe von Geometrien.
  - **ST\_ConstrainedDelaunayTriangles** - Gibt eine eingeschränkte Delaunay-Triangulation um die angegebene Eingabegeometrie zurück.
  - **ST\_ConvexHull** - Berechnet die konvexe Hülle einer Geometrie.
  - **ST\_CoordDim** - Gibt die Dimension der Koordinaten für den Wert von ST\_Geometry zurück.
  - **ST\_CurveN** - Returns the Nth component curve geometry of a CompoundCurve.
  - **ST\_CurveToLine** - Konvertiert eine Geometrie mit Kurven in eine lineare Geometrie.
  - **ST\_DelaunayTriangles** - Gibt die Delaunay-Triangulation der Scheitelpunkte einer Geometrie zurück.
  - **ST\_Difference** - Berechnet eine Geometrie, die den Teil der Geometrie A darstellt, der die Geometrie B nicht schneidet.
  - **ST\_DistanceCPA** - Liefert den Abstand zwischen dem nächstgelegenen Punkt der Annäherung zweier Flughäfen.
  - **ST\_Dump** - Gibt einen Satz von geometry\_dump Zeilen für die Komponenten einer Geometrie zurück.
  - **ST\_DumpPoints** - Gibt eine Zusammenfassung des Inhalts einer Geometrie wieder.
  - **ST\_DumpRings** - Gibt einen Satz von geometry\_dump Zeilen für die äußeren und inneren Ringe eines Polygons zurück.
  - **ST\_DumpSegments** - Gibt eine Zusammenfassung des Inhalts einer Geometrie wieder.
  - **ST\_EndPoint** - Gibt die Anzahl der Stützpunkte eines ST\_LineString oder eines ST\_CircularString zurück.
  - **ST\_ExteriorRing** - Gibt die Anzahl der inneren Ringe einer Polygoneometrie aus.
  - **ST\_Extrude** - Extrudieren einer Oberfläche in ein zugehöriges Volumen
  - **ST\_FlipCoordinates** - Gibt eine Version einer Geometrie mit gespiegelter X- und Y-Achse zurück.
  - **ST\_Force2D** - Die Geometrien in einen "2-dimensionalen Modus" zwingen.
  - **ST\_ForceCurve** - Wandelt einen geometrischen in einen Kurven Datentyp um, soweit anwendbar.
  - **ST\_ForceLHR** - LHR-Ausrichtung erzwingen
-



- **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\_Force3D** - Zwingt die Geometrien in einen XYZ Modus. Dies ist ein Alias für ST\_Force3DZ.
  - **ST\_Force3DZ** - Zwingt die Geometrien in einen XYZ Modus.
  - **ST\_Force4D** - Zwingt die Geometrien in einen XYZM Modus.
  - **ST\_ForceCollection** - Wandelt eine Geometrie in eine GEOMETRYCOLLECTION um.
  - **ST\_GeomFromEWKB** - Gibt einen geometrischen Datentyp (ST\_Geometry) aus einer Well-known-Binary (WKB) Darstellung zurück.
  - **ST\_GeomFromEWKT** - Gibt einen spezifizierten ST\_Geometry-Wert von einer erweiterten Well-known-Text Darstellung (EWKT) zurück.
  - **ST\_GeomFromGML** - Nimmt als Eingabe eine GML-Darstellung der Geometrie und gibt ein geometrisches PostGIS-Objekt aus.
  - **ST\_GeomFromGeoJSON** - Nimmt als Eingabe eine GeoJSON-Darstellung der Geometrie und gibt ein geometrisches PostGIS-Objekt aus.
  - **ST\_GeomFromKML** - Nimmt als Eingabe eine KML-Darstellung der Geometrie und gibt ein geometrisches PostGIS-Objekt aus.
  - **ST\_GeometricMedian** - Gibt den geometrischen Median eines Mehrfachpunktes zurück.
  - **ST\_GeometryN** - Gibt den Geometrietyp des ST\_Geometry Wertes zurück.
  - **ST\_GeometryType** - Gibt den Geometrietyp des ST\_Geometry Wertes zurück.
  - **ST\_HasArc** - Prüft, ob eine Geometrie einen Kreisbogen enthält
  - **ST\_HasM** - Prüft, ob eine Geometrie eine M-Dimension (Maß) hat.
  - **ST\_HasZ** - Prüft, ob eine Geometrie eine Z-Dimension hat.
  - **ST\_InteriorRingN** - Gibt die Anzahl der inneren Ringe einer Polygoneometrie aus.
  - **ST\_InterpolatePoint** - Für einen gegebenen Punkt wird die Kilometrierung auf dem nächstliegenden Punkt einer Geometrie zurück.
  - **ST\_Intersection** - Berechnet eine Geometrie, die den gemeinsamen Teil der Geometrien A und B darstellt.
  - **ST\_IsClosed** - Gibt den Wert TRUE zurück, wenn die Anfangs- und Endpunkte des LINESTRING's zusammenfallen. Bei polyedrischen Oberflächen, wenn sie geschlossen (volumetrisch) sind.
  - **ST\_IsCollection** - Gibt den Wert TRUE zurück, falls es sich bei der Geometrie um eine leere GeometryCollection, Polygon, Point etc. handelt.
  - **ST\_IsPlanar** - Prüfen, ob eine Fläche planar ist 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** - Prüfen, ob die Geometrie ein Solid ist. Es wird keine Gültigkeitsprüfung durchgeführt.
  - **ST\_IsValidTrajectory** - Prüft, ob die Geometrie eine gültige Flugbahn ist.
  - **ST\_LengthSpheroid** - Gibt den geometrischen Schwerpunkt einer Geometrie zurück.
  - **ST\_LineFromMultiPoint** - Erzeugt einen LineString aus einer MultiPoint Geometrie.
  - **ST\_LineInterpolatePoint** - Gibt einen oder mehrere, entlang einer Linie interpolierte Punkte zurück.
  - **ST\_LineInterpolatePoints** - Gibt einen oder mehrere, entlang einer Linie interpolierte Punkte zurück.
  - **ST\_LineSubstring** - Gibt den Teil einer Linie zwischen zwei gebrochenen Stellen zurück.
  - **ST\_LineToCurve** - Konvertiert eine lineare Geometrie in eine gekrümmte Geometrie.
  - **ST\_LocateBetweenElevations** - Gibt die Teile einer Geometrie zurück, die in einem Höhenbereich (Z) liegen.
  - **ST\_M** - Gibt die M-Koordinate eines Punktes zurück.
  - **ST\_MakeLine** - Erzeugt einen Linienzug aus einer Punkt-, Mehrfachpunkt- oder Liniengeometrie.
  - **ST\_MakePoint** - Erzeugt eine 2D-, 3DZ- oder 4D-Punktgeometrie.
  - **ST\_MakePolygon** - Erzeugt ein Polygon aus einer Schale und einer optionalen Liste von Löchern.
  - **ST\_MakeSolid** - Gießen Sie die Geometrie in einen Körper. Es wird keine Prüfung durchgeführt. Um ein gültiges Solid zu erhalten, muss die Eingabegeometrie eine geschlossene polyedrische Fläche oder ein geschlossenes TIN sein.
  - **ST\_MakeValid** - Versucht, eine ungültige Geometrie gültig zu machen, ohne dass Scheitelpunkte verloren gehen.
  - **ST\_MemSize** - Gibt den Geometrietyp des ST\_Geometry Wertes zurück.
  - **ST\_MemUnion** - Aggregatfunktion, die Geometrien auf eine speichereffiziente, aber langsamere Weise zusammenfasst
  - **ST\_NDims** - Gibt die Dimension der Koordinaten für den Wert von ST\_Geometry zurück.
  - **ST\_NPoints** - Gibt die Anzahl der Punkte (Knoten) einer Geometrie zurück.
  - **ST\_NRings** - Gibt die Anzahl der inneren Ringe einer Polygoneometrie aus.
  - **ST\_Node** - Knoten eine Sammlung von Linien.
  - **ST\_NumCurves** - Return the number of component curves in a CompoundCurve.
  - **ST\_NumGeometries** - Gibt die Anzahl der Punkte einer Geometrie zurück. Funktioniert für alle Geometrien.
  - **ST\_NumPatches** - Gibt die Anzahl der Maschen einer polyedrischen Oberfläche aus. Gibt NULL zurück, wenn es sich nicht um polyedrische Geometrien handelt.
  - **ST\_Orientation** - Bestimmung der Oberflächenausrichtung
  - **ST\_PatchN** - Gibt den Geometrietyp des ST\_Geometry Wertes zurück.
  - **ST\_PointFromWKB** - Erzeugt eine Geometrie mit gegebener SRID von WKB.
  - **ST\_PointN** - Gibt die Anzahl der Stützpunkte eines ST\_LineString oder eines ST\_CircularString zurück.
  - **ST\_PointOnSurface** - Berechnet einen Punkt, der garantiert in einem Polygon oder auf einer Geometrie liegt.
  - **ST\_Points** - Gibt einen MultiPoint zurück, welcher alle Koordinaten einer Geometrie enthält.
  - **ST\_Polygon** - Erzeugt ein Polygon aus einem LineString mit einem angegebenen SRID.
  - **ST\_RemovePoint** - Einen Punkt aus einem Linienzug entfernen.
  - **ST\_RemoveRepeatedPoints** - Gibt eine Version einer Geometrie zurück, bei der doppelte Punkte entfernt wurden.
  - **ST\_Reverse** - Gibt die Geometrie in umgekehrter Knotenreihenfolge zurück.
-



- **ST\_Rotate** - Dreht eine Geometrie um einen Ursprungspunkt.
  - **ST\_RotateX** - Dreht eine Geometrie um die X-Achse.
  - **ST\_RotateY** - Dreht eine Geometrie um die Y-Achse.
  - **ST\_RotateZ** - Dreht eine Geometrie um die Z-Achse.
  - **ST\_Scale** - Skaliert eine Geometrie um bestimmte Faktoren.
  - **ST\_Scroll** - Startpunkt eines geschlossenen LineStrings ändern.
  - **ST\_SetPoint** - Einen Punkt eines Linienzuges durch einen gegebenen Punkt ersetzen.
  - **ST\_ShiftLongitude** - Verschiebt die Längenkoordinaten einer Geometrie zwischen -180..180 und 0..360.
  - **ST\_SnapToGrid** - Fängt alle Punkte der Eingabegeometrie auf einem regelmäßigen Gitter.
  - **ST\_StartPoint** - Gibt den ersten Punkt eines LineString zurück.
  - **ST\_StraightSkeleton** - Berechnet die konvexe Hülle einer Geometrie.
  - **ST\_SwapOrdinates** - Gibt eine Version der Ausgangsgeometrie zurück, in der die angegebenen Ordinatenwerte ausgetauscht werden.
  - **ST\_SymDifference** - Berechnet eine Geometrie, die die Teile der Geometrien A und B darstellt, die sich nicht überschneiden.
  - **ST\_Tessellate** - Führt eine Oberflächentessellierung eines Polygons oder einer Polyederfläche durch und gibt diese als TIN oder Sammlung von TINS zurück
  - **ST\_TransScale** - Verschiebt und skaliert eine Geometrie mit vorgegebenen Offsets und Faktoren.
  - **ST\_Translate** - Verschiebt eine Geometrie um vorgegebene Offsets.
  - **ST\_UnaryUnion** - Berechnet die Vereinigung der Komponenten einer einzelnen Geometrie.
  - **ST\_Union** - Berechnet eine Geometrie, die die Punktmengevereinigung der Eingabegeometrien darstellt.
  - **ST\_Volume** - Berechnet das Volumen eines 3D-Volumens. Bei Anwendung auf (auch geschlossene) Flächengeometrien wird 0 zurückgegeben.
  - **ST\_WrapX** - Versammelt eine Geometrie um einen X-Wert
  - **ST\_X** - Gibt die X-Koordinate eines Punktes zurück.
  - **ST\_XMax** - Gibt die X-Maxima eines 2D- oder 3D-Begrenzungsrahmens oder einer Geometrie zurück.
  - **ST\_XMin** - Gibt die X-Minima eines 2D- oder 3D-Begrenzungsrahmens oder einer Geometrie zurück.
  - **ST\_Y** - Gibt die Y-Koordinate eines Punktes zurück.
  - **ST\_YMax** - Gibt die Y-Maxima eines 2D- oder 3D-Begrenzungsrahmens oder einer Geometrie zurück.
  - **ST\_YMin** - Gibt die Y-Minima eines 2D- oder 3D-Begrenzungsrahmens oder einer Geometrie zurück.
  - **ST\_Z** - Gibt die Z-Koordinate eines Punktes zurück.
  - **ST\_ZMax** - Gibt die Z-Maxima eines 2D- oder 3D-Begrenzungsrahmens oder einer Geometrie zurück.
  - **ST\_ZMin** - Gibt die Z-Minima eines 2D- oder 3D-Begrenzungsrahmens oder einer Geometrie zurück.
  - **ST\_Zmflag** - Gibt die Dimension der Koordinaten von ST\_Geometry zurück.
  - **Equals** - Gibt TRUE zurück, wenn zwei TopoGeometry Objekte aus denselben topologischen Elementarstrukturen bestehen.
  - **Intersects** - Gibt TRUE zurück, wenn sich kein beliebiges Paar von Elementarstrukturen zweier TopoGeometry Objekte überschneidet.
-

- **UpdateGeometrySRID** - Aktualisiert die SRID aller Features in einer Geometriespalte und die Metadaten der Tabelle.
- **&&&** - Gibt TRUE zurück, wenn A's n-D bounding box B's n-D bounding box schneidet.
- **&&&(geometry,gidx)** - Gibt TRUE zurück, wenn die (cached) n-D bounding box einer Geometrie eine n-D float precision bounding box (GIDX) schneidet.
- **&&&(gidx,geometry)** - Gibt TRUE zurück, wenn eine n-D float precision bounding box (GIDX) eine (cached) n-D bounding box einer Geometrie schneidet.
- **&&&(gidx,gidx)** - Gibt TRUE zurück, wenn sich zwei n-D float precision bounding boxes (GIDX) gegenseitig überschneiden.

## 13.9 PostGIS Funktionen zur Unterstützung gekrümmter Geometrien

Die folgenden Funktionen sind PostGIS-Funktionen, die CIRCULARSTRING, CURVEPOLYGON und andere gekrümmte Geometrietypen verwenden können

- **AddGeometryColumn** - Entfernt eine Geometriespalte aus einer räumlichen Tabelle.
- **Box2D** - Gibt ein BOX2D zurück, das die 2D-Ausdehnung einer Geometrie darstellt.
- **Box3D** - Gibt ein BOX3D zurück, das die 3D-Ausdehnung einer Geometrie darstellt.
- **DropGeometryColumn** - Entfernt eine Geometriespalte aus einer räumlichen Tabelle.
- **GeometryType** - Gibt den Geometrietyp des ST\_Geometry Wertes zurück.
- **PostGIS\_AddBBox** - Fügt der Geometrie ein umschreibendes Rechteck bei.
- **PostGIS\_DropBBox** - Löscht die umschreibenden Rechtecke der Geometrie.
- **PostGIS\_HasBBox** - Gibt TRUE zurück, wenn die BBox der Geometrie zwischengespeichert ist, andernfalls wird FALSE zurückgegeben.
- **ST\_3DExtent** - Aggregatfunktion, die den 3D-Begrenzungsrahmen von Geometrien zurückgibt.
- **ST\_Affine** - Wenden Sie eine affine 3D-Transformation auf eine Geometrie an.
- **ST\_AsBinary** - Rückgabe der OGC/ISO Well-Known Binary (WKB)-Darstellung der Geometrie/Geografie ohne SRID-Metadaten.
- **ST\_AsEWKB** - Rückgabe der Extended Well-Known Binary (EWKB) Darstellung der Geometrie mit SRID-Metadaten.
- **ST\_AsEWKT** - Gibt die Well-known-Text(WKT) Darstellung der Geometrie mit den SRID-Metadaten zurück.
- **ST\_AsHEXEWKB** - Gibt eine Geometrie im HEXEWKB Format (als Text) aus; verwendet entweder die Little-Endian (NDR) oder die Big-Endian (XDR) Zeichenkodierung.
- **ST\_AsSVG** - Gibt eine Geometrie als SVG-Pfad aus.
- **ST\_AsText** - Gibt die Well-known-Text(WKT) Darstellung der Geometrie/Geographie ohne die SRID Metadaten zurück.
- **ST\_ClusterDBSCAN** - Fensterfunktion, die eine Cluster-ID für jede Eingabegeometrie unter Verwendung des DBSCAN-Algorithmus zurückgibt.
- **ST\_ClusterWithin** - Aggregatfunktion, die Geometrien nach Trennungsabstand gruppiert.
- **ST\_ClusterWithinWin** - Fensterfunktion, die eine Cluster-ID für jede Eingabegeometrie zurückgibt, Clustering anhand des Trennungsabstands.
- **ST\_Collect** - Erzeugt eine GeometryCollection oder Multi\*-Geometrie aus einer Reihe von Geometrien.
- **ST\_CoordDim** - Gibt die Dimension der Koordinaten für den Wert von ST\_Geometry zurück.
- **ST\_CurveToLine** - Konvertiert eine Geometrie mit Kurven in eine lineare Geometrie.

- **ST\_Distance** - Gibt die größte 3-dimensionale Distanz zwischen zwei geometrischen Objekten als Linie zurück
  - **ST\_Dump** - Gibt einen Satz von geometry\_dump Zeilen für die Komponenten einer Geometrie zurück.
  - **ST\_DumpPoints** - Gibt eine Zusammenfassung des Inhalts einer Geometrie wieder.
  - **ST\_EndPoint** - Gibt die Anzahl der Stützpunkte eines ST\_LineString oder eines ST\_CircularString zurück.
  - **ST\_EstimatedExtent** - Gibt die geschätzte Ausdehnung einer räumlichen Tabelle zurück.
  - **ST\_FlipCoordinates** - Gibt eine Version einer Geometrie mit gespiegelter X- und Y-Achse zurück.
  - **ST\_Force2D** - Die Geometrien in einen "2-dimensionalen Modus" zwingen.
  - **ST\_ForceCurve** - Wandelt einen geometrischen in einen Kurven Datentyp um, soweit anwendbar.
  - **ST\_ForceSFS** - Erzwingt, dass Geometrien nur vom Typ SFS 1.1 sind.
  - **ST\_Force3D** - Zwingt die Geometrien in einen XYZ Modus. Dies ist ein Alias für ST\_Force3DZ.
  - **ST\_Force3DM** - Zwingt die Geometrien in einen XYM Modus.
  - **ST\_Force3DZ** - Zwingt die Geometrien in einen XYZ Modus.
  - **ST\_Force4D** - Zwingt die Geometrien in einen XYZM Modus.
  - **ST\_ForceCollection** - Wandelt eine Geometrie in eine GEOMETRYCOLLECTION um.
  - **ST\_GeoHash** - Gibt die Geometrie in der GeoHash Darstellung aus.
  - **ST\_GeogFromWKB** - Erzeugt ein geographisches Objekt aus der Well-known-Binary (WKB) oder der erweiterten Well-known-Binary (EWKB) Darstellung.
  - **ST\_GeomFromEWKB** - Gibt einen geometrischen Datentyp (ST\_Geometry) aus einer Well-known-Binary (WKB) Darstellung zurück.
  - **ST\_GeomFromEWKT** - Gibt einen spezifizierten ST\_Geometry-Wert von einer erweiterten Well-known-Text Darstellung (EWKT) zurück.
  - **ST\_GeomFromText** - Gibt einen spezifizierten ST\_Geometry Wert aus einer Well-known-Text Darstellung (WKT) zurück.
  - **ST\_GeomFromWKB** - Erzeugt ein geometrisches Objekt aus der Well-known-Binary (WKB) Darstellung und einer optionalen SRID.
  - **ST\_GeometryN** - Gibt den Geometriertyp des ST\_Geometry Wertes zurück.
  - **=** - Gibt TRUE zurück, wenn die Koordinaten und die Reihenfolge der Koordinaten der Geometrie/Geographie A und der Geometrie/Geographie B ident sind.
  - **&<l** - Gibt TRUE zurück, wenn die bounding box von A jene von B überlagert oder unterhalb liegt.
  - **ST\_HasArc** - Prüft, ob eine Geometrie einen Kreisbogen enthält
  - **ST\_Intersects** - Prüft, ob sich zwei Geometrien schneiden (sie haben mindestens einen Punkt gemeinsam)
  - **ST\_IsClosed** - Gibt den Wert TRUE zurück, wenn die Anfangs- und Endpunkte des LINESTRING's zusammenfallen. Bei polyedrischen Oberflächen, wenn sie geschlossen (volumetrisch) sind.
  - **ST\_IsCollection** - Gibt den Wert TRUE zurück, falls es sich bei der Geometrie um eine leere GeometryCollection, Polygon, Point etc. handelt.
  - **ST\_IsEmpty** - Prüft, ob eine Geometrie leer ist.
  - **ST\_LineToCurve** - Konvertiert eine lineare Geometrie in eine gekrümmte Geometrie.
  - **ST\_MemSize** - Gibt den Geometriertyp des ST\_Geometry Wertes zurück.
-

- **ST\_NPoints** - Gibt die Anzahl der Punkte (Knoten) einer Geometrie zurück.
  - **ST\_NRings** - Gibt die Anzahl der inneren Ringe einer Polygoneometrie aus.
  - **ST\_PointFromWKB** - Erzeugt eine Geometrie mit gegebener SRID von WKB.
  - **ST\_PointN** - Gibt die Anzahl der Stützpunkte eines ST\_LineString oder eines ST\_CircularString zurück.
  - **ST\_Points** - Gibt einen MultiPoint zurück, welcher alle Koordinaten einer Geometrie enthält.
  - **ST\_Rotate** - Dreht eine Geometrie um einen Ursprungspunkt.
  - **ST\_RotateZ** - Dreht eine Geometrie um die Z-Achse.
  - **ST\_SRID** - Gibt die Raumbezugskenung für eine Geometrie zurück.
  - **ST\_Scale** - Skaliert eine Geometrie um bestimmte Faktoren.
  - **ST\_SetSRID** - Legen Sie den SRID für eine Geometrie fest.
  - **ST\_StartPoint** - Gibt den ersten Punkt eines LineString 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** - Verschiebt und skaliert eine Geometrie mit vorgegebenen Offsets und Faktoren.
  - **ST\_Transform** - Rückgabe einer neuen Geometrie mit in ein anderes räumliches Bezugssystem transformierten Koordinaten.
  - **ST\_Translate** - Verschiebt eine Geometrie um vorgegebene Offsets.
  - **ST\_XMax** - Gibt die X-Maxima eines 2D- oder 3D-Begrenzungsrahmens oder einer Geometrie zurück.
  - **ST\_XMin** - Gibt die X-Minima eines 2D- oder 3D-Begrenzungsrahmens oder einer Geometrie zurück.
  - **ST\_YMax** - Gibt die Y-Maxima eines 2D- oder 3D-Begrenzungsrahmens oder einer Geometrie zurück.
  - **ST\_YMin** - Gibt die Y-Minima eines 2D- oder 3D-Begrenzungsrahmens oder einer Geometrie zurück.
  - **ST\_ZMax** - Gibt die Z-Maxima eines 2D- oder 3D-Begrenzungsrahmens oder einer Geometrie zurück.
  - **ST\_ZMin** - Gibt die Z-Minima eines 2D- oder 3D-Begrenzungsrahmens oder einer Geometrie zurück.
  - **ST\_Zmflag** - Gibt die Dimension der Koordinaten von ST\_Geometry zurück.
  - **UpdateGeometrySRID** - Aktualisiert die SRID aller Features in einer Geometriespalte und die Metadaten der Tabelle.
  - **~(box2df,box2df)** - Gibt TRUE zurück, wenn eine 2D float precision bounding box (BOX2DF) eine andere 2D float precision bounding box (BOX2DF) enthält.
  - **~(box2df,geometry)** - Gibt TRUE zurück, wenn eine 2D float precision bounding box (BOX2DF) die 2D Bounding Box einer Geometrie enthält.
  - **~(geometry,box2df)** - Gibt TRUE zurück, wenn die 2D bounding box einer Geometrie eine 2D float precision bounding box (GIDX) enthält.
  - **&&** - Gibt TRUE zurück, wenn die 2D Bounding Box von A die 2D Bounding Box von B schneidet.
  - **&&&** - Gibt TRUE zurück, wenn A's n-D bounding box B's n-D bounding box schneidet.
  - **@(box2df,box2df)** - Gibt TRUE zurück, wenn eine 2D float precision bounding box (BOX2DF) innerhalb einer anderen 2D float precision bounding box enthalten ist.
  - **@(box2df,geometry)** - Gibt TRUE zurück, wenn eine 2D float precision bounding box (BOX2DF) in der 2D Bounding Box einer Geometrie enthalten ist..
-

- `@(geometry,box2df)` - Gibt TRUE zurück, wenn die 2D Bounding Box einer Geometrie in einer 2D float precision Bounding Box (BOX2DF) enthalten ist.
- `&&(box2df,box2df)` - Gibt TRUE zurück, wenn sich zwei 2D float precision Bounding Boxes (BOX2DF) überschneiden.
- `&&(box2df,geometry)` - Gibt TRUE zurück, wenn eine 2D float precision bounding box (BOX2DF) eine Geometrie (cached) 2D bounding box schneidet.
- `&&(geometry,box2df)` - Gibt TRUE zurück, wenn sich die 2D Bounding Box (cached) einer Geometrie mit einer 2D Bounding Box mit Gleitpunktgenauigkeit (BOX2DF) überschneidet.
- `&&&(geometry,gidx)` - Gibt TRUE zurück, wenn die (cached) n-D bounding box einer Geometrie eine n-D float precision bounding box (GIDX) schneidet.
- `&&&(gidx,geometry)` - Gibt TRUE zurück, wenn eine n-D float precision bounding box (GIDX) eine (cached) n-D bounding box einer Geometrie schneidet.
- `&&&(gidx,gidx)` - Gibt TRUE zurück, wenn sich zwei n-D float precision bounding boxes (GIDX) gegenseitig überschneiden.

## 13.10 PostGIS-Funktionen zur Unterstützung polyedrischer Flächen

Die folgenden Funktionen sind PostGIS-Funktionen, die die Geometrien POLYHEDRALSURFACE und POLYHEDRALSURFACEM verwenden können

- `Box2D` - Gibt ein BOX2D zurück, das die 2D-Ausdehnung einer Geometrie darstellt.
- `Box3D` - Gibt ein BOX3D zurück, das die 3D-Ausdehnung einer Geometrie darstellt.
- `CG_3DArea` - Berechnet die Fläche von 3D-Oberflächengeometrien. Gibt 0 für Solids zurück.
- `CG_3DConvexHull` - Berechnet die konvexe Hülle einer Geometrie.
- `CG_3DDifference` - 3D-Differenz durchführen
- `CG_3DIntersection` - 3D-Schnitte durchführen
- `CG_3DUnion` - Perform 3D union using postgis\_sfcgal.
- `CG_ApproximateMedialAxis` - Berechnet die konvexe Hülle einer Geometrie.
- `CG_Extrude` - Extrudieren einer Oberfläche in ein zugehöriges Volumen
- `CG_ForceLHR` - LHR-Ausrichtung erzwingen
- `CG_IsPlanar` - Prüfen, ob eine Fläche planar ist oder nicht
- `CG_IsSolid` - Prüfen, ob die Geometrie ein Solid ist. Es wird keine Gültigkeitsprüfung durchgeführt.
- `CG_MakeSolid` - Gießen Sie die Geometrie in einen Körper. Es wird keine Prüfung durchgeführt. Um ein gültiges Solid zu erhalten, muss die Eingabegerometrie eine geschlossene polyedrische Fläche oder ein geschlossenes TIN sein.
- `CG_StraightSkeleton` - Berechnet die konvexe Hülle einer Geometrie.
- `CG_Tessellate` - Führt eine Oberflächentesselierung eines Polygons oder einer Polyederfläche durch und gibt diese als TIN oder Sammlung von TINS zurück
- `CG_Visibility` - Berechnen eines Sichtbarkeitspolygons aus einem Punkt oder einem Segment in einer Polygeometrie
- `CG_Volume` - Berechnet das Volumen eines 3D-Volumens. Bei Anwendung auf (auch geschlossene) Flächengeometrien wird 0 zurückgegeben.
- `GeometryType` - Gibt den Geometrietyp des ST\_Geometry Wertes zurück.
- `ST_3DArea` - Berechnet die Fläche von 3D-Oberflächengeometrien. Gibt 0 für Solids zurück.

- **ST\_3DClosestPoint** - Gibt den 3-dimensionalen Punkt auf g1 zurück, der den kürzesten Abstand zu g2 hat. Dies ist der Anfangspunkt des kürzesten Abstands in 3D.
  - **ST\_3DConvexHull** - Berechnet die konvexe Hülle einer Geometrie.
  - **ST\_3DDFullyWithin** - Prüft, ob zwei 3D-Geometrien vollständig innerhalb eines bestimmten 3D-Abstands liegen
  - **ST\_3DDWithin** - Prüft, ob zwei 3D-Geometrien innerhalb eines bestimmten 3D-Abstands liegen
  - **ST\_3DDifference** - 3D-Differenz durchführen
  - **ST\_3DDistance** - Für den geometrischen Datentyp. Es wird der geringste 3-dimensionale kartesische Abstand (basierend auf dem Koordinatenreferenzsystem) zwischen zwei geometrischen Objekten in projizierten Einheiten zurückgegeben.
  - **ST\_3DExtent** - Aggregatfunktion, die den 3D-Begrenzungsrahmen von Geometrien zurückgibt.
  - **ST\_3DIntersection** - 3D-Schnitte durchführen
  - **ST\_3DIntersects** - Prüft, ob sich zwei Geometrien in 3D räumlich schneiden - nur für Punkte, Linienzüge, Polygone, polyedrische Flächen (Bereich)
  - **ST\_3DLongestLine** - Gibt die größte 3-dimensionale Distanz zwischen zwei geometrischen Objekten als Linie zurück
  - **ST\_3DMaxDistance** - Für den geometrischen Datentyp. Gibt die maximale 3-dimensionale kartesische Distanz (basierend auf dem Koordinatenreferenzsystem) zwischen zwei geometrischen Objekten in projizierten Einheiten zurück.
  - **ST\_3DShortestLine** - Gibt den kürzesten 3-dimensionalen Abstand zwischen zwei geometrischen Objekten als Linie zurück
  - **ST\_3DUnion** - 3D-Vereinigung durchführen.
  - **ST\_Affine** - Wenden Sie eine affine 3D-Transformation auf eine Geometrie an.
  - **ST\_ApproximateMedialAxis** - Berechnet die konvexe Hülle einer Geometrie.
  - **ST\_Area** - Gibt den geometrischen Schwerpunkt einer Geometrie zurück.
  - **ST\_AsBinary** - Rückgabe der OGC/ISO Well-Known Binary (WKB)-Darstellung der Geometrie/Geografie ohne SRID-Metadaten.
  - **ST\_AsEWKB** - Rückgabe der Extended Well-Known Binary (EWKB) Darstellung der Geometrie mit SRID-Metadaten.
  - **ST\_AsEWKT** - Gibt die Well-known-Text(WKT) Darstellung der Geometrie mit den SRID-Metadaten zurück.
  - **ST\_AsGML** - Gibt die Geometrie als GML-Element - Version 2 oder 3 - zurück.
  - **ST\_AsX3D** - Gibt eine Geometrie im X3D XML Knotenelement-Format zurück: ISO-IEC-19776-1.2-X3DEncodings-XML
  - **ST\_CoordDim** - Gibt die Dimension der Koordinaten für den Wert von ST\_Geometry zurück.
  - **ST\_Dimension** - Gibt die Dimension der Koordinaten für den Wert von ST\_Geometry zurück.
  - **ST\_Dump** - Gibt einen Satz von geometry\_dump Zeilen für die Komponenten einer Geometrie zurück.
  - **ST\_DumpPoints** - Gibt eine Zusammenfassung des Inhalts einer Geometrie wieder.
  - **ST\_Expand** - Gibt einen Begrenzungsrahmen zurück, der aus einem anderen Begrenzungsrahmen oder einer Geometrie erweitert wurde.
  - **ST\_Extent** - Aggregatfunktion, die den Begrenzungsrahmen von Geometrien zurückgibt.
  - **ST\_Extrude** - Extrudieren einer Oberfläche in ein zugehöriges Volumen
  - **ST\_FlipCoordinates** - Gibt eine Version einer Geometrie mit gespiegelter X- und Y-Achse zurück.
  - **ST\_Force2D** - Die Geometrien in einen "2-dimensionalen Modus" zwingen.
  - **ST\_ForceLHR** - LHR-Ausrichtung erzwingen
  - **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 den Geometrietyp des ST\_Geometry Wertes zurück.
  - **ST\_GeometryType** - Gibt den Geometrietyp des ST\_Geometry Wertes zurück.
  - **=** - Gibt TRUE zurück, wenn die Koordinaten und die Reihenfolge der Koordinaten der Geometrie/Geographie A und der Geometrie/Geographie B ident sind.
  - **&<l** - 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** - Prüfen, ob eine Fläche planar ist oder nicht
  - **ST\_IsSolid** - Prüfen, ob die Geometrie ein Solid ist. Es wird keine Gültigkeitsprüfung durchgeführt.
  - **ST\_MakeSolid** - Gießen Sie die Geometrie in einen Körper. Es wird keine Prüfung durchgeführt. Um ein gültiges Solid zu erhalten, muss die Eingabegeometrie eine geschlossene polyedrische Fläche oder ein geschlossenes TIN sein.
  - **ST\_MemSize** - Gibt den Geometrietyp des ST\_Geometry Wertes zurück.
  - **ST\_NPoints** - Gibt die Anzahl der Punkte (Knoten) einer Geometrie zurück.
  - **ST\_NumGeometries** - Gibt die Anzahl der Punkte einer Geometrie zurück. Funktioniert für alle Geometrien.
  - **ST\_NumPatches** - Gibt die Anzahl der Maschen einer polyedrischen Oberfläche aus. Gibt NULL zurück, wenn es sich nicht um polyedrische Geometrien handelt.
  - **ST\_PatchN** - Gibt den Geometrietyp des ST\_Geometry Wertes zurück.
  - **ST\_RemoveRepeatedPoints** - Gibt eine Version einer Geometrie zurück, bei der doppelte Punkte entfernt wurden.
  - **ST\_Reverse** - Gibt die Geometrie in umgekehrter Knotenreihenfolge zurück.
  - **ST\_Rotate** - Dreht eine Geometrie um einen Ursprungspunkt.
  - **ST\_RotateX** - Dreht eine Geometrie um die X-Achse.
  - **ST\_RotateY** - Dreht eine Geometrie um die Y-Achse.
  - **ST\_RotateZ** - Dreht eine Geometrie um die Z-Achse.
  - **ST\_Scale** - Skaliert eine Geometrie um bestimmte Faktoren.
  - **ST\_ShiftLongitude** - Verschiebt die Längenkoordinaten einer Geometrie zwischen -180..180 und 0..360.
  - **ST\_StraightSkeleton** - Berechnet die konvexe Hülle einer Geometrie.
-




- **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** - Führt eine Oberflächentesselierung eines Polygons oder einer Polyederfläche durch und gibt diese als TIN oder Sammlung von TINS zurück
- **ST\_Transform** - Rückgabe einer neuen Geometrie mit in ein anderes räumliches Bezugssystem transformierten Koordinaten.
- **ST\_Volume** - Berechnet das Volumen eines 3D-Volumens. Bei Anwendung auf (auch geschlossene) Flächengeometrien 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.

## 13.11 PostGIS Funktionsunterstützungsmatrix


Nachfolgend finden Sie eine alphabetische Auflistung der raumspezifischen Funktionen in PostGIS und die Arten von Raumtypen, mit denen sie arbeiten, bzw. die OGC/SQL-Konformität, die sie zu erfüllen versuchen.

- Ein  bedeutet, dass die Funktion mit dem Typ oder Subtyp von Haus aus arbeitet.
- A  bedeutet, dass es funktioniert, aber mit einem Transformations-Cast eingebaut mit Cast-to-Geometrie, transformieren, um eine "beste srid" räumliche ref und dann zurück zu werfen. Die Ergebnisse sind möglicherweise nicht wie erwartet für große Bereiche oder Bereiche an Polen und können Fließkomma-Müll ansammeln.



- Eine  bedeutet, dass die Funktion mit dem Typ funktioniert, weil ein Auto-Cast auf einen anderen, wie z.B. auf box3d, anstatt einer direkten Typunterstützung.



- Eine  bedeutet, dass die Funktion nur verfügbar ist, wenn PostGIS mit SFCGAL-Unterstützung kompiliert wurde.
- geom - Grundlegende 2D-Geometrieunterstützung (x,y).
- geog - Grundlegende 2D-Geografie-Unterstützung (x,y).
- 2.5D - grundlegende 2D-Geometrien im 3 D/4D-Raum (mit Z- oder M-Koordinaten).
- PS - Polyedrische Flächen
- T - Dreiecke und triangulierte unregelmäßige Netzflächen (TIN)

Funktion	geom	geog	2.5D	Kurven	SQL MM	PS	T
ST_Collect	✓		✓	✓			
ST_LineFromMulti	✓ t		✓				
ST_MakeEnvelope	✓						
ST_MakeLine	✓		✓				
ST_MakePoint	✓		✓				
ST_MakePointM	✓						
ST_MakePolygon	✓		✓				
ST_Point	✓				✓		
ST_PointZ	✓						
ST_PointM	✓						
ST_PointZM	✓						
ST_Polygon	✓		✓		✓		
ST_TileEnvelope	✓						
ST_HexagonGrid	✓						
ST_Hexagon	✓						
ST_SquareGrid	✓						
ST_Square	✓						
ST_Letters	✓						
GeometryType	✓		✓	✓		✓	✓
ST_Boundary	✓		✓		✓		
ST_BoundingDiago	✓		✓				
ST_CoordDim	✓		✓	✓	✓	✓	✓
ST_Dimension	✓				✓	✓	✓
ST_Dump	✓		✓	✓		✓	✓
ST_DumpPoints	✓		✓	✓		✓	✓

Funktion	geom	geog	2.5D	Kurven	SQL MM	PS	T
ST_DumpSegments	✓		✓				✓
ST_DumpRings	✓		✓				
ST_EndPoint	✓		✓	✓	✓		
ST_Envelope	✓				✓		
ST_ExteriorRing	✓		✓		✓		
ST_GeometryN	✓		✓	✓	✓	✓	✓
ST_GeometryType	✓		✓		✓	✓	
ST_HasArc	✓		✓	✓			
ST_InteriorRingN	✓		✓		✓		
ST_NumCurves	✓		✓		✓		
ST_CurveN	✓		✓		✓		
ST_IsClosed	✓		✓	✓	✓	✓	
ST_IsCollection	✓		✓	✓			
ST_IsEmpty	✓			✓	✓		
ST_IsPolygonCCW	✓		✓				
ST_IsPolygonCW	✓		✓				
ST_IsRing	✓				✓		
ST_IsSimple	✓		✓		✓		
ST_M	✓		✓		✓		
ST_MemSize	✓		✓	✓		✓	✓
ST_NDims	✓		✓				
ST_NPoints	✓		✓	✓		✓	
ST_NRings	✓		✓	✓			
ST_NumGeometries	✓		✓		✓	✓	✓
ST_NumInteriorRings	✓				✓		
ST_NumInteriorRings	✓						
ST_NumPatches	✓		✓		✓	✓	
ST_NumPoints	✓				✓		
ST_PatchN	✓		✓		✓	✓	
ST_PointN	✓		✓	✓	✓		
ST_Points	✓		✓	✓			
ST_StartPoint	✓		✓	✓	✓		
ST_Summary	✓	✓		✓		✓	✓
ST_X	✓		✓		✓		

Funktion	geom	geog	2.5D	Kurven	SQL MM	PS	T
ST_Y	✓		✓		✓		
ST_Z	✓		✓		✓		
ST_Zmflag	✓		✓	✓			
ST_HasZ	✓		✓				
ST_HasM	✓		✓				
ST_AddPoint	✓		✓				
ST_CollectionExtra	✓						
ST_CollectionHomogenize	✓						
ST_CurveToLine	✓		✓	✓	✓		
ST_Scroll	✓		✓				
ST_FlipCoordinates	✓		✓	✓		✓	✓
ST_Force2D	✓		✓	✓		✓	
ST_Force3D	✓		✓	✓		✓	
ST_Force3DZ	✓		✓	✓		✓	
ST_Force3DM	✓			✓			
ST_Force4D	✓		✓	✓			
ST_ForceCollection	✓		✓	✓		✓	
ST_ForceCurve	✓		✓	✓			
ST_ForcePolygonC	✓		✓				
ST_ForcePolygonC	✓		✓				
ST_ForceSFS	✓		✓	✓		✓	✓
ST_ForceRHR	✓		✓			✓	
ST_LineExtend	✓						
ST_LineToCurve	✓		✓	✓			
ST_Multi	✓						
ST_Normalize	✓						
ST_Project	✓	✓					
ST_QuantizeCoordinates	✓						
ST_RemovePoint	✓		✓				
ST_RemoveRepeatedPoints	✓		✓			✓	
ST_RemoveIrrelevantPointsForView	✓						
ST_RemoveSmallPoints	✓						
ST_Reverse	✓		✓			✓	
ST_Segmentize	✓	✓					

Funktion	geom	geog	2.5D	Kurven	SQL MM	PS	T
ST_SetPoint	✓		✓				
ST_ShiftLongitude	✓		✓			✓	✓
ST_WrapX	✓		✓				
ST_SnapToGrid	✓		✓				
ST_Snap	✓						
ST_SwapOrdinates	✓		✓	✓		✓	✓
ST_IsValid	✓				✓		
ST_IsValidDetail	✓						
ST_IsValidReason	✓						
ST_MakeValid	✓		✓				
ST_InverseTransform	✓	pipeline					
ST_SetSRID	✓			✓			
ST_SRID	✓			✓	✓		
ST_Transform	✓			✓	✓	✓	
ST_TransformPipeline	✓						
postgis_srs_codes							
postgis_srs							
postgis_srs_all							
postgis_srs_search	✓						
ST_BdPolyFromText	✓						
ST_BdMPolyFromText	✓						
ST_GeogFromText		✓					
ST_GeographyFromText		✓					
ST_GeomCollFromText	✓				✓		
ST_GeomFromEWKB	✓		✓	✓		✓	✓
ST_GeomFromMultipart	✓						
ST_GeometryFromText	✓				✓		
ST_GeomFromText	✓			✓	✓		
ST_LineFromText	✓				✓		
ST_MLineFromText	✓				✓		
ST_MPointFromText	✓				✓		
ST_MPolyFromText	✓				✓		
ST_PointFromText	✓				✓		
ST_PolygonFromText	✓				✓		
ST_WKTToSQL	✓				✓		

Funktion	geom	geog	2.5D	Kurven	SQL MM	PS	T
ST_GeogFromWKB		✓		✓			
ST_GeomFromEWKB	✓		✓	✓		✓	✓
ST_GeomFromWK	✓			✓	✓		
ST_LineFromWKB	✓				✓		
ST_LinestringFromWKB	✓				✓		
ST_PointFromWK	✓		✓	✓	✓		
ST_WKBTToSQL	✓				✓		
ST_Box2dFromGeoHash	✓						
ST_GeomFromGeoHash	✓						
ST_GeomFromGML	✓		✓			✓	✓
ST_GeomFromGeoJSON	✓		✓				
ST_GeomFromKML	✓		✓				
ST_GeomFromTWKB	✓						
ST_GMLToSQL	✓				✓		
ST_LineFromEncodedPolyline	✓						
ST_PointFromGeoHash							
ST_FromFlatGeobufToTable							
ST_FromFlatGeobuf							
ST_AsEWKT	✓	✓	✓	✓		✓	✓
ST_AsText	✓	✓		✓	✓		
ST_AsBinary	✓	✓	✓	✓	✓	✓	✓
ST_AsEWKB	✓		✓	✓		✓	✓
ST_AsHEXEWKB	✓		✓	✓			
ST_AsEncodedPolyline	✓						
ST_AsFlatGeobuf	✗						
ST_AsGeobuf	✗						
ST_AsGeoJSON	✓	✓	✓				
ST_AsGML	✓	✓	✓		✓	✓	✓
ST_AsKML	✓	✓	✓				
ST_AsLatLonText	✓						
ST_AsMARC21	✓						
ST_AsMVTGeom	✓						
ST_AsMVT	✗						
ST_AsSVG	✓	✓		✓			
ST_AsTWKB	✓						

Funktion	geom	geog	2.5D	Kurven	SQL MM	PS	T
ST_AsX3D	✓		✓			✓	✓
ST_GeoHash	✓			✓			
&&	✓	✓		✓		✓	
&&(geometry,box2df)	✓			✓		✓	
&&(box2df,geometry)	✓			✓		✓	
&&(box2df,box2df)	✗			✓		✓	
&&&	✓		✓	✓		✓	✓
&&&(geometry,gid)	✓		✓	✓		✓	✓
&&&(gid,geometry)	✓		✓	✓		✓	✓
&&&(gid,gid)			✓	✓		✓	✓
&<	✓						
&<	✓			✓		✓	
&>	✓						
<<	✓						
<<	✓						
=	✓	✓		✓		✓	
>>	✓						
@	✓						
@(geometry,box2df)	✓			✓		✓	
@(box2df,geometry)	✓			✓		✓	
@(box2df,box2df)	✗			✓		✓	
&>	✓						
>>	✓						
~	✓						
~(geometry,box2df)	✓			✓		✓	
~(box2df,geometry)	✓			✓		✓	
~(box2df,box2df)	✗			✓		✓	
~=	✓					✓	
<->	✓	✓					
=	✓						
<#>	✓						
<<->>	✓						
ST_3DIntersects	✓		✓		✓	✓	✓
ST_Contains	✓				✓		

Funktion	geom	geog	2.5D	Kurven	SQL MM	PS	T
ST_ContainsProperly	✓						
ST_CoveredBy	✓	✓					
ST_Covers	✓	✓					
ST_Crosses	✓				✓		
ST_Disjoint	✓				✓		
ST_Equals	✓				✓		
ST_Intersects	✓	✓		✓	✓		✓
ST_LineCrossingDirection	✓						
ST_OrderingEquals	✓				✓		
ST_Overlaps	✓				✓		
ST_Relate	✓				✓		
ST_RelateMatch							
ST_Touches	✓				✓		
ST_Within	✓				✓		
ST_3DDWithin	✓		✓		✓	✓	
ST_3DDFullyWithin	✓		✓			✓	
ST_DFullyWithin	✓						
ST_DWithin	✓	✓					
ST_PointInsideCircle	✓						
ST_Area	✓	✓			✓	✓	
ST_Azimuth	✓	✓					
ST_Angle	✓						
ST_ClosestPoint	✓	✓					
ST_3DClosestPoint	✓		✓			✓	
ST_Distance	✓	✓		✓	✓		
ST_3DDistance	✓		✓		✓	✓	
ST_DistanceSphere	✓						
ST_DistanceSpheroidal	✓						
ST_FrechetDistance	✓						
ST_HausdorffDistance	✓						
ST_Length	✓	✓			✓		
ST_Length2D	✓						
ST_3DLength	✓		✓		✓		
ST_LengthSpheroidal	✓		✓				

Funktion	geom	geog	2.5D	Kurven	SQL MM	PS	T
ST_LongestLine	✓						
ST_3DLongestLine	✓		✓			✓	
ST_MaxDistance	✓						
ST_3DMaxDistance	✓		✓			✓	
ST_MinimumClearance	✓						
ST_MinimumClearanceLine	✓						
ST_Perimeter	✓	✓			✓		
ST_Perimeter2D	✓						
ST_3DPerimeter	✓		✓		✓		
ST_ShortestLine	✓	✓					
ST_3DShortestLine	✓		✓			✓	
ST_ClipByBox2D	✓						
ST_Difference	✓		✓		✓		
ST_Intersection	✓	😄	✓		✓		
ST_MemUnion	✓		✓				
ST_Node	✓		✓				
ST_Split	✓						
ST_Subdivide	✓						
ST_SymDifference	✓		✓		✓		
ST_UnaryUnion	✓		✓				
ST_Union	✓		✓		✓		
ST_Buffer	✓	😄			✓		
ST_BuildArea	✓						
ST_Centroid	✓	✓			✓		
ST_ChaikinSmooth	✓		✓				
ST_ConcaveHull	✓						
ST_ConvexHull	✓		✓		✓		
ST_DelaunayTriangulation	✓		✓				✓
ST_FilterByM	✓						
ST_GeneratePoints	✓						
ST_GeometricMedian	✓		✓				
ST_LineMerge	✓						
ST_MaximumInscribedCircle	✓						
ST_LargestEmptyCircle	✓						



Funktion	geom	geog	2.5D	Kurven	SQL MM	PS	T
ST_MinimumBoun	✓ Circle						
ST_MinimumBoun	✓ Radius						
ST_OrientedEnvelo	✓						
ST_OffsetCurve	✓						
ST_PointOnSurface	✓		✓		✓		
ST_Polygonize	✓						
ST_ReducePrecision	✓						
ST_SharedPaths	✓						
ST_Simplify	✓						
ST_SimplifyPreserv	✓ pology						
ST_SimplifyPolygo	✓ ll						
ST_SimplifyVW	✓						
ST_SetEffectiveAre	✓						
ST_TriangulatePoly	✓						
ST_VoronoiLines	✓						
ST_VoronoiPolygor	✓						
ST_CoverageInvalid	✓ ges						
ST_CoverageSimpli	✓						
ST_CoverageUnion	✓						
ST_Affine	✓		✓	✓		✓	✓
ST_Rotate	✓		✓	✓		✓	✓
ST_RotateX	✓		✓			✓	✓
ST_RotateY	✓		✓			✓	✓
ST_RotateZ	✓		✓	✓		✓	✓
ST_Scale	✓		✓	✓		✓	✓
ST_Translate	✓		✓	✓			
ST_TransScale	✓		✓	✓			
ST_ClusterDBSCA	✓			✓			
ST_ClusterIntersect	✓						
ST_ClusterIntersect	✓ Win						
ST_ClusterKMeans	✓						
ST_ClusterWithin	✓			✓			
ST_ClusterWithinW	✓			✓			
Box2D	✓			✓		✓	✓

Funktion	geom	geog	2.5D	Kurven	SQL MM	PS	T
Box3D	✓		✓	✓		✓	✓
ST_EstimatedExtent	✓			✓			
ST_Expand	✓					✓	✓
ST_Extent	✓					✓	✓
ST_3DExtent	✓		✓	✓		✓	✓
ST_MakeBox2D	✓						
ST_3DMakeBox	✓						
ST_XMax	✓		✓	✓			
ST_XMin	✓		✓	✓			
ST_YMax	✓		✓	✓			
ST_YMin	✓		✓	✓			
ST_ZMax	✓		✓	✓			
ST_ZMin	✓		✓	✓			
ST_LineInterpolatePoint	✓	✓	✓				
ST_3DLineInterpolatePoint	✓		✓				
ST_LineInterpolatePoints	✓	✓	✓				
ST_LineLocatePoint	✓	✓					
ST_LineSubstring	✓	✓	✓				
ST_LocateAlong	✓				✓		
ST_LocateBetween	✓				✓		
ST_LocateBetweenPoints	✓		✓				
ST_InterpolatePoint	✓		✓				
ST_AddMeasure	✓		✓				
ST_IsValidTrajectory	✓		✓				
ST_ClosestPointOfTrajectory	✓		✓				
ST_DistanceCPA	✓		✓				
ST_CPASWithin	✓		✓				
postgis.backend							
postgis.gdal_datapath							
postgis.gdal_enabled_drivers							
postgis.enable_outdb_rasters							
postgis.gdal_vsi_options							
PostGIS_AddBBox	✓			✓			
PostGIS_DropBBox	✓			✓			
PostGIS_HasBBox	✓			✓			
postgis_sfcgal_version							
postgis_sfcgal_full_version							

Funktion	geom	geog	2.5D	Kurven	SQL MM	PS	T
CG_ForceLHR							
CG_IsPlanar							
CG_IsSolid							
CG_MakeSolid							
CG_Orientation							
CG_Area							
CG_3DArea							
CG_Volume							
ST_ForceLHR							
ST_IsPlanar							
ST_IsSolid							
ST_MakeSolid							
ST_Orientation							
ST_3DArea							
ST_Volume							
CG_Intersection							
CG_Intersects							
CG_3DIntersects							
CG_Difference							
ST_3DDifference							
CG_3DDifference							
CG_Distance							
CG_3DDistance							
ST_3DConvexHull							

Funktion	geom	geog	2.5D	Kurven	SQL MM	PS	T
CG_3DConvexHull							
ST_3DIntersection							
CG_3DIntersection							
CG_Union							
ST_3DUnion							
CG_3DUnion							
ST_AlphaShape	✓						
CG_AlphaShape							
CG_ApproxConvexHull							
ST_ApproximateMedialAxis							
CG_ApproximateMedialAxis							
ST_ConstrainedDelaunayTriangles							
CG_ConstrainedDelaunayTriangles							
ST_Extrude							
CG_Extrude							
CG_ExtrudeStraightSkeleton							
CG_GreeneApproximateHexPartition							
ST_MinkowskiSum							
CG_MinkowskiSum							
ST_OptimalAlphaShape							
CG_OptimalAlphaShape							
CG_OptimalConvexHull							
CG_StraightSkeleton							
ST_StraightSkeleton							

Funktion	geom	geog	2.5D	Kurven	SQL MM	PS	T
ST_Tessellate							
CG_Tessellate							
CG_Triangulate							
CG_Visibility							
CG_YMonotonePartition							
getfaceedges_returntype							
TopoGeometry							
validatetopology_returntype							
TopoElement							
TopoElementArray							
AddTopoGeometryColumn							
RenameTopoGeometryColumn							
DropTopology							
RenameTopology							
DropTopoGeometryColumn							
Populate_Topology_Layer							
TopologySummary							
ValidateTopology ✓							
ValidateTopologyRelation							
FindTopology							
FindLayer							
CreateTopology							
CopyTopology							
ST_InitTopoGeo					✓		
ST_CreateTopoGeo ✓					✓		
TopoGeo_AddPoint ✓							
TopoGeo_AddLineString ✓							
TopoGeo_AddPolygon ✓							
TopoGeo_LoadGeometry ✓							
ST_AddIsoNode ✓					✓		
ST_AddIsoEdge ✓					✓		
ST_AddEdgeNewFace ✓					✓		
ST_AddEdgeModFace ✓					✓		
ST_RemEdgeNewFace					✓		
ST_RemEdgeModFace					✓		
ST_ChangeEdgeGeometry ✓					✓		
ST_ModEdgeSplit ✓					✓		
ST_ModEdgeHeal					✓		

Funktion	geom	geog	2.5D	Kurven	SQL MM	PS	T
ST_NewEdgeHeal					✓		
ST_MoveIsoNode	✓				✓		
ST_NewEdgesSplit	✓				✓		
ST_RemoveIsoNode					✓		
ST_RemoveIsoEdge					✓		
GetEdgeByPoint	✓						
GetFaceByPoint	✓						
GetFaceContainingI	✓ t						
GetNodeByPoint	✓						
GetTopologyID							
GetTopologySRID							
GetTopologyName							
ST_GetFaceEdges					✓		
ST_GetFaceGeomet	✓				✓		
GetRingEdges							
GetNodeEdges							
Polygonize							
AddNode	✓						
AddEdge	✓						
AddFace	✓						
ST_Simplify	✓						
RemoveUnusedPrin	✓ es						
CreateTopoGeom	✓						
toTopoGeom	✓						
TopoElementArray_Agg							
TopoElement	✓						
clearTopoGeom	✓						
TopoGeom_addEler	✓ :						
TopoGeom_remEle	✓ t						
TopoGeom_addTop	✓ om						
toTopoGeom							
GetTopoGeomElementArray							
GetTopoGeomElements							
ST_SRID	✓				✓		
AsGML	✓						
AsTopoJSON	✓						
Equals	✓		✓				
Intersects	✓		✓				

Funktion	geom	geog	2.5D	Kurven	SQL MM	PS	T
geomval							
addbandarg							
rastbandarg							
raster							
reclassarg							
summarystats							
unionarg							
AddRasterConstraints							
DropRasterConstraints							
AddOverviewConstraints							
DropOverviewConstraints							
PostGIS_GDAL_Version							
PostGIS_Raster_Lib_Build_Date							
PostGIS_Raster_Lib_Version							
ST_GDALDrivers							
ST_Contour							
ST_InterpolateRaster	✓						
UpdateRasterSRID							
ST_CreateOverview							
ST_AddBand							
ST_AsRaster	✓						
ST_Band							
ST_MakeEmptyCoverage							
ST_MakeEmptyRaster							
ST_Tile							
ST_Retile	✓						
ST_FromGDALRaster							
ST_GeoReference							
ST_Height							
ST_IsEmpty							
ST_MemSize							
ST_MetaData							
ST_NumBands							
ST_PixelHeight							
ST_PixelWidth							
ST_ScaleX							
ST_ScaleY							
ST_RasterToWorldCoord							
ST_RasterToWorldCoordX							
ST_RasterToWorldCoordY							
ST_Rotation							
ST_SkewX							
ST_SkewY							
ST_SRID							
ST_Summary							
ST_UpperLeftX							
ST_UpperLeftY							
ST_Width							
ST_WorldToRasterCoord	✓						
ST_WorldToRasterCoordX	✓						
ST_WorldToRasterCoordY	✓						
ST_BandMetaData							

Funktion	geom	geog	2.5D	Kurven	SQL MM	PS	T
ST_BandNoDataValue							
ST_BandIsNoData							
ST_BandPath							
ST_BandFileSize							
ST_BandFileTimestamp							
ST_BandPixelType							
ST_MinPossibleValue							
ST_HasNoBand							
ST_PixelAsPolygon	✓						
ST_PixelAsPolygons							
ST_PixelAsPoint	✓						
ST_PixelAsPoints							
ST_PixelAsCentroid	✓						
ST_PixelAsCentroids							
ST_Value	✓						
ST_NearestValue	✓						
ST_SetZ	✓						
ST_SetM	✓						
ST_Neighborhood	✓						
ST_SetValue	✓						
ST_SetValues							
ST_DumpValues							
ST_PixelOfValue							
ST_SetGeoReference							
ST_SetRotation							
ST_SetScale							
ST_SetSkew							
ST_SetSRID							
ST_SetUpperLeft							
ST_Resample							
ST_Rescale							
ST_Reskew							
ST_SnapToGrid							
ST_Resize							
ST_Transform							
ST_SetBandNoDataValue							
ST_SetBandIsNoData							
ST_SetBandPath							
ST_SetBandIndex							
ST_Count							
ST_CountAgg							
ST_Histogram							
ST_Quantile							
ST_SummaryStats							
ST_SummaryStatsAgg							
ST_ValueCount							
ST_RastFromWKB							
ST_RastFromHexWKB							
ST_AsBinary/ST_AsWKB							
ST_AsHexWKB							



Funktion	geom	geog	2.5D	Kurven	SQL MM	PS	T
ST_AsGDALRaster							
ST_AsJPEG							
ST_AsPNG							
ST_AsTIFF							
ST_Clip	✓						
ST_ColorMap							
ST_Grayscale							
ST_Intersection	✓						
ST_MapAlgebra (callback function version)							
ST_MapAlgebra (expression version)							
ST_MapAlgebraExpr							
ST_MapAlgebraExpr							
ST_MapAlgebraFct							
ST_MapAlgebraFct							
ST_MapAlgebraFctNgb							
ST_Reclass							
ST_Union							
ST_Distinct4ma							
ST_InvDistWeight4ma							
ST_Max4ma							
ST_Mean4ma							
ST_Min4ma							
ST_MinDist4ma							
ST_Range4ma							
ST_StdDev4ma							
ST_Sum4ma							
ST_Aspect							
ST_HillShade							
ST_Roughness							
ST_Slope							
ST_TPI							
ST_TRI							
Box3D	<input checked="" type="checkbox"/>						
ST_ConvexHull	✓						
ST_DumpAsPolygons							
ST_Envelope	✓						
ST_MinConvexHull	✓						
ST_Polygon	✓						
&&	✓						
&<							
&>							
=							
@	✓						
~=							

Funktion	geom	geog	2.5D	Kurven	SQL MM	PS	T
~	✓						
ST_Contains							
ST_ContainsProperly							
ST_Covers							
ST_CoveredBy							
ST_Disjoint							
ST_Intersects	✓						
ST_Overlaps							
ST_Touches							
ST_SameAlignment							
ST_NotSameAlignmentReason							
ST_Within							
ST_DWithin							
ST_DFullyWithin							
stdaddr							
rules table							
lex table							
gaz table							
debug_standardize_address							
parse_address							
standardize_address							
Drop_Indexes_Generate_Script							
Drop_Nation_Tables_Generate_Script							
Drop_State_Tables_Generate_Script							
Geocode	✓						
Geocode_Intersectio	✓						
Get_Geocode_Setting							
Get_Tract	✓						
Install_Missing_Indexes							
Loader_Generate_Census_Script							
Loader_Generate_Script							
Loader_Generate_Nation_Script							
Missing_Indexes_Generate_Script							
Normalize_Address							
Page_Normalize_Address							
Pprint_Addy							
Reverse_Geocode	✓						
Topology_Load_Tiger							
Set_Geocode_Setting							

## 13.12 Neue, erweiterte oder geänderte PostGIS-Funktionen

### 13.12.1 PostGIS-Funktionen neu oder erweitert in 3.5

Die unten aufgeführten Funktionen sind PostGIS-Funktionen, die hinzugefügt oder erweitert wurden.

Neue Funktionen in PostGIS 3.5

- **CG\_3DArea** - Verfügbarkeit: 3.5.0 Berechnet die Fläche von 3D-Oberflächengeometrien. Gibt 0 für Solids zurück.
- **CG\_3DConvexHull** - Verfügbarkeit: 3.5.0 Berechnet die konvexe Hülle einer Geometrie.

- **CG\_3DDifference** - Verfügbarkeit: 3.5.0 3D-Differenz durchführen
  - **CG\_3DDistance** - Verfügbarkeit: 3.5.0 Computes the minimum 3D distance between two geometries
  - **CG\_3DIntersection** - Verfügbarkeit: 3.5.0 3D-Schnitte durchführen
  - **CG\_3DIntersects** - Verfügbarkeit: 3.5.0 Tests if two 3D geometries intersect
  - **CG\_3DUnion** - Verfügbarkeit: 3.5.0 Perform 3D union using postgis\_sfcgal.
  - **CG\_AlphaShape** - Availability: 3.5.0 - requires SFCGAL  $\geq$  1.4.1. Berechnet eine Alpha-Form, die eine Geometrie umschließt
  - **CG\_ApproxConvexPartition** - Verfügbarkeit: 3.5.0 - erfordert SFCGAL  $\geq$  1.5.0. Berechnet die approximale konvexe Partition der Polyongeometrie
  - **CG\_ApproximateMedialAxis** - Verfügbarkeit: 3.5.0 Berechnet die konvexe Hülle einer Geometrie.
  - **CG\_Area** - Verfügbarkeit: 3.5.0 Calculates the area of a geometry
  - **CG\_Difference** - Verfügbarkeit: 3.5.0 Computes the geometric difference between two geometries
  - **CG\_Distance** - Verfügbarkeit: 3.5.0 Computes the minimum distance between two geometries
  - **CG\_Extrude** - Verfügbarkeit: 3.5.0 Extrudieren einer Oberfläche in ein zugehöriges Volumen
  - **CG\_ExtrudeStraightSkeleton** - Verfügbarkeit: 3.5.0 - erfordert SFCGAL  $\geq$  1.5.0. Gerade Skelett-Extrusion
  - **CG\_ForceLHR** - Verfügbarkeit: 3.5.0 LHR-Ausrichtung erzwingen
  - **CG\_GreeneApproxConvexPartition** - Verfügbarkeit: 3.5.0 - erfordert SFCGAL  $\geq$  1.5.0. Berechnet die approximale konvexe Partition der Polyongeometrie
  - **CG\_Intersection** - Verfügbarkeit: 3.5.0 Computes the intersection of two geometries
  - **CG\_Intersects** - Verfügbarkeit: 3.5.0 Prüft, ob sich zwei Geometrien schneiden (sie haben mindestens einen Punkt gemeinsam)
  - **CG\_IsPlanar** - Verfügbarkeit: 3.5.0 Prüfen, ob eine Fläche planar ist oder nicht
  - **CG\_IsSolid** - Verfügbarkeit: 3.5.0 Prüfen, ob die Geometrie ein Solid ist. Es wird keine Gültigkeitsprüfung durchgeführt.
  - **CG\_MakeSolid** - Verfügbarkeit: 3.5.0 Gießen Sie die Geometrie in einen Körper. Es wird keine Prüfung durchgeführt. Um ein gültiges Solid zu erhalten, muss die Eingabegeometrie eine geschlossene polyedrische Fläche oder ein geschlossenes TIN sein.
  - **CG\_MinkowskiSum** - Verfügbarkeit: 3.5.0 Führt die Minkowski-Summe aus
  - **CG\_OptimalAlphaShape** - Availability: 3.5.0 - requires SFCGAL  $\geq$  1.4.1. Berechnet eine Alpha-Form, die eine Geometrie umschließt, unter Verwendung eines "optimalen" Alpha-Wertes.
  - **CG\_OptimalConvexPartition** - Verfügbarkeit: 3.5.0 - erfordert SFCGAL  $\geq$  1.5.0. Berechnet eine optimale konvexe Partition der Polyongeometrie
  - **CG\_Orientation** - Verfügbarkeit: 3.5.0 Bestimmung der Oberflächenausrichtung
  - **CG\_StraightSkeleton** - Verfügbarkeit: 3.5.0 Berechnet die konvexe Hülle einer Geometrie.
  - **CG\_Tessellate** - Verfügbarkeit: 3.5.0 Führt eine Oberflächentessellierung eines Polygons oder einer Polyederfläche durch und gibt diese als TIN oder Sammlung von TINS zurück
  - **CG\_Triangulate** - Verfügbarkeit: 3.5.0 Triangulates a polygonal geometry
  - **CG\_Union** - Verfügbarkeit: 3.5.0 Computes the union of two geometries
  - **CG\_Visibility** - Verfügbarkeit: 3.5.0 - erfordert SFCGAL  $\geq$  1.5.0. Berechnen eines Sichtbarkeitspolygons aus einem Punkt oder einem Segment in einer Polyongeometrie
-

- **CG\_Volume** - Verfügbarkeit: 3.5.0 Berechnet das Volumen eines 3D-Volumens. Bei Anwendung auf (auch geschlossene) Flächengeometrien wird 0 zurückgegeben.
- **CG\_YMonotonePartition** - Verfügbarkeit: 3.5.0 - erfordert SFCGAL >= 1.5.0. Berechnet die y-monotone Partition der Polygoneometrie
- **ST\_HasM** - Verfügbarkeit: 3.5.0 Prüft, ob eine Geometrie eine M-Dimension (Maß) hat.
- **ST\_HasZ** - Verfügbarkeit: 3.5.0 Prüft, ob eine Geometrie eine Z-Dimension hat.
- **ST\_RemoveIrrelevantPointsForView** - Verfügbarkeit: 3.5.0 Removes points that are irrelevant for rendering a specific rectangular view of a geometry.
- **ST\_RemoveSmallParts** - Verfügbarkeit: 3.5.0 Removes small parts (polygon rings or linestrings) of a geometry.
- **TopoGeo\_LoadGeometry** - Verfügbarkeit: 3.5.0 Load a geometry into an existing topology, snapping and splitting as needed.

#### Erweiterte Funktionen in PostGIS 3.5

- **ST\_Clip** - Verbessert: 3.5.0 - berührtes Argument hinzugefügt. Returns the raster clipped by the input geometry. If band number is not specified, all bands are processed. If crop is not specified or TRUE, the output raster is cropped. If touched is set to TRUE, then touched pixels are included, otherwise only if the center of the pixel is in the geometry it is included.

#### Geänderte Funktionen in PostGIS 3.5

- **ST\_AsGeoJSON** - Geändert: 3.5.0 erlaubt die Angabe der Spalte, die die Feature-ID enthält Rückgabe einer Geometrie oder eines Merkmals im GeoJSON-Format.
- **ST\_DFullyWithin** - Changed: 3.5.0 : the logic behind the function now uses a test of containment within a buffer, rather than the ST\_MaxDistance algorithm. Results will differ from prior versions, but should be closer to user expectations. Tests if a geometry is entirely inside a distance of another

### 13.12.2 PostGIS-Funktionen neu oder erweitert in 3.4

Die unten aufgeführten Funktionen sind PostGIS-Funktionen, die hinzugefügt oder erweitert wurden.

#### Neue Funktionen in PostGIS 3.4

- **PostGIS\_GEOS\_Compiled\_Version** - Verfügbarkeit: 3.4.0 Gibt die Versionsnummer der GEOS-Bibliothek zurück, mit der PostGIS erstellt wurde.
- **PostGIS\_PROJ\_Compiled\_Version** - Verfügbarkeit: 3.5.0 Returns the version number of the PROJ library against which PostGIS was built.
- **RenameTopoGeometryColumn** - Verfügbarkeit: 3.4.0 Benennt eine topogeometrische Spalte um
- **RenameTopology** - Verfügbarkeit: 3.4.0 Benennt eine Topologie um
- **ST\_ClusterIntersectingWin** - Verfügbarkeit: 3.4.0 Fensterfunktion, die für jede Eingabegeometrie eine Cluster-ID zurückgibt und die Eingabegeometrien in zusammenhängende Gruppen clustert.
- **ST\_ClusterWithinWin** - Verfügbarkeit: 3.4.0 Fensterfunktion, die eine Cluster-ID für jede Eingabegeometrie zurückgibt, Clustering anhand des Trennungsabstands.
- **ST\_CoverageInvalidEdges** - Verfügbarkeit: 3.4.0 Fensterfunktion, die Stellen findet, an denen die Polygone keine gültige Abdeckung bilden.
- **ST\_CoverageSimplify** - Verfügbarkeit: 3.4.0 Fensterfunktion, die die Kanten einer polygonalen Abdeckung vereinfacht.
- **ST\_CoverageUnion** - Verfügbarkeit: 3.4.0 - erfordert GEOS >= 3.8.0 Berechnet die Vereinigung einer Menge von Polygonen, die eine Abdeckung bilden, indem gemeinsame Kanten entfernt werden.

- **ST\_InverseTransformPipeline** - Verfügbarkeit: 3.4.0 Rückgabe einer neuen Geometrie mit in ein anderes räumliches Bezugssystem transformierten Koordinaten unter Verwendung der Umkehrung einer definierten Koordinatentransformationspipeline.
- **ST\_LargestEmptyCircle** - Verfügbarkeit: 3.4.0. Berechnet den größten Kreis, der eine Geometrie nicht überschneidet.
- **ST\_LineExtend** - Verfügbarkeit: 3.4.0 Gibt eine Linie zurück, die um die angegebenen Abstände vorwärts und rückwärts verlängert wurde.
- **ST\_TransformPipeline** - Verfügbarkeit: 3.4.0 Rückgabe einer neuen Geometrie mit in ein anderes räumliches Bezugssystem transformierten Koordinaten unter Verwendung einer definierten Koordinatentransformationspipeline.
- **TopoElement** - Verfügbarkeit: 3.4.0 Konvertiert eine Topogeometrie in ein Topoelement.
- **debug\_standardize\_address** - Verfügbarkeit: 3.4.0 Gibt einen json-formatierten Text zurück, der die Parse-Token und Standardisierungen auflistet
- **postgis\_srs** - Verfügbarkeit: 3.4.0 Rückgabe eines Metadatensatzes für die angefragte Behörde und srid.
- **postgis\_srs\_all** - Verfügbarkeit: 3.4.0 Gibt Metadatensätze für jedes räumliche Bezugssystem in der zugrunde liegenden Proj-Datenbank zurück.
- **postgis\_srs\_codes** - Verfügbarkeit: 3.4.0 Gibt die Liste der SRS-Codes zurück, die mit der angegebenen Behörde verbunden sind.
- **postgis\_srs\_search** - Verfügbarkeit: 3.4.0 Gibt Metadatensätze für projizierte Koordinatensysteme zurück, die Nutzungsbereiche haben, die den Parameter bounds vollständig enthalten.

#### Erweiterte Funktionen in PostGIS 3.4

- **PostGIS\_Full\_Version** - Verbessert: 3.4.0 enthält jetzt zusätzliche PROJ-Konfigurationen NETWORK\_ENABLED, URL\_ENDPOINT und DATABASE\_PATH des proj.db-Speicherorts Meldet die vollständige PostGIS-Version und Informationen zur Build-Konfiguration.
- **PostGIS\_PROJ\_Version** - Verbessert: 3.4.0 enthält jetzt NETWORK\_ENABLED, URL\_ENDPOINT und DATABASE\_PATH des proj.db-Speicherorts Gibt die Versionsnummer der PROJ4-Bibliothek zurück.
- **ST\_AsSVG** - Verbessert: 3.4.0 zur Unterstützung aller Kurventypen Gibt eine Geometrie als SVG-Pfad aus.
- **ST\_ClosestPoint** - Verbessert: 3.4.0 - Unterstützung für Geographie. Gibt den 2D-Punkt auf g1 zurück, der g2 am nächsten ist. Dies ist der erste Punkt der kürzesten Linie von einer Geometrie zur anderen.
- **ST\_LineSubstring** - Verbessert: 3.4.0 - Unterstützung für Geographie wurde eingeführt. Gibt den Teil einer Linie zwischen zwei gebrochenen Stellen zurück.
- **ST\_Project** - Verbessert: 3.4.0 Erlaubt Geometrieargumente und Zweipunktform ohne Azimut. Gibt einen Punkt zurück, der von einem Startpunkt um eine bestimmte Entfernung und Peilung (Azimut) projiziert wird.
- **ST\_Resample** - Verbessert: 3.4.0 Max und Min Resampling Optionen hinzugefügt Skaliert einen Raster mit einem bestimmten Algorithmus, neuen Dimensionen, einer beliebigen Gitterecke und über Parameter zur Georeferenzierung des Rasters, die angegeben oder von einem anderen Raster übernommen werden können.
- **ST\_Rescale** - Verbessert: 3.4.0 Max und Min Resampling Optionen hinzugefügt Neuabtastung eines Rasters, indem nur die Skala (oder Pixelgröße) angepasst wird. Die neuen Pixelwerte werden mit den Algorithmen NearestNeighbor (englische oder amerikanische Schreibweise), Bilinear, Cubic, CubicSpline, Lanczos, Max oder Min resampling berechnet. Die Voreinstellung ist NearestNeighbor.
- **ST\_ShortestLine** - Verbessert: 3.4.0 - Unterstützung für Geographie. Gibt die 2-dimensionale kürzeste Strecke zwischen zwei Geometrien als Linie zurück

#### Geänderte Funktionen in PostGIS 3.4

- **PostGIS\_Extensions\_Upgrade** - Geändert: 3.4.0 um das Argument target\_version hinzuzufügen. Packt und aktualisiert PostGIS-Erweiterungen (z.B. postgis\_raster, postgis\_topology, postgis\_sfcgal) auf die angegebene oder neueste Version.

### 13.12.3 PostGIS-Funktionen neu oder erweitert in 3.3

Die unten aufgeführten Funktionen sind PostGIS-Funktionen, die hinzugefügt oder erweitert wurden.

Neue Funktionen in PostGIS 3.3

- **RemoveUnusedPrimitives** - Verfügbarkeit: 3.3.0 Entfernt Topologieprimitive, die zur Definition bestehender TopoGeometry-Objekte nicht benötigt werden.
- **ST\_3DConvexHull** - Verfügbarkeit: 3.3.0 Berechnet die konvexe Hülle einer Geometrie.
- **ST\_3DUnion** - Verfügbarkeit: 3.3.0 Aggregatvariante wurde hinzugefügt 3D-Vereinigung durchführen.
- **ST\_AsMARC21** - Verfügbarkeit: 3.3.0 Gibt die Geometrie als MARC21/XML-Datensatz mit einem geografischen Datenfeld (034) zurück.
- **ST\_GeomFromMARC21** - Verfügbarkeit: 3.3.0, erfordert libxml2 2.6+ Nimmt MARC21/XML-Geodaten als Eingabe und gibt ein PostGIS-Geometrieobjekt zurück.
- **ST\_Letters** - Verfügbarkeit: 3.3.0 Gibt die eingegebenen Buchstaben als Geometrie mit einer Standardstartposition am Ursprung und einer Standardtexthöhe von 100 zurück.
- **ST\_OptimalAlphaShape** - Verfügbarkeit: 3.3.0 - erfordert SFCGAL >= 1.4.1. Berechnet eine Alpha-Form, die eine Geometrie umschließt, unter Verwendung eines "optimalen" Alpha-Wertes.
- **ST\_SimplifyPolygonHull** - Verfügbarkeit: 3.3.0. Berechnet eine vereinfachte topologieerhaltende äußere oder innere Hülle einer polygonalen Geometrie.
- **ST\_TriangulatePolygon** - Verfügbarkeit: 3.3.0. Berechnet die eingeschränkte Delaunay-Triangulation von Polygonen
- **postgis\_sfcgal\_full\_version** - Verfügbarkeit: 3.3.0 Liefert die vollständige Version von SFCGAL, einschließlich der CGAL- und Boost-Versionen

Erweiterte Funktionen in PostGIS 3.3

- **ST\_ConcaveHull** - Verbessert: 3.3.0, native GEOS-Implementierung aktiviert für GEOS 3.11+ Berechnet eine möglicherweise konkave Geometrie, die alle Eckpunkte der Eingabegeometrie enthält
- **ST\_LineMerge** - Verbessert: 3.3.0 akzeptiert einen gerichteten Parameter. Gibt die Linien zurück, die durch das Zusammenfügen eines MultiLineString gebildet werden.

Geänderte Funktionen in PostGIS 3.3

- **PostGIS\_Extensions\_Upgrade** - Geändert: 3.3.0 Unterstützung für Upgrades von jeder PostGIS-Version. Funktioniert nicht auf allen Systemen. Packt und aktualisiert PostGIS-Erweiterungen (z.B. postgis\_raster, postgis\_topology, postgis\_sfcgal) auf die angegebene oder neueste Version.

### 13.12.4 PostGIS-Funktionen neu oder erweitert in 3.2

Die unten aufgeführten Funktionen sind PostGIS-Funktionen, die hinzugefügt oder erweitert wurden.

Neue Funktionen in PostGIS 3.2

- **FindLayer** - Verfügbarkeit: 3.2.0 Gibt einen topology.layer-Datensatz mit anderen Mitteln zurück.
- **FindTopology** - Verfügbarkeit: 3.2.0 Gibt einen Topologie-Datensatz mit anderen Mitteln zurück.
- **GetFaceContainingPoint** - Verfügbarkeit: 3.2.0 Findet die Fläche, die einen Punkt enthält.
- **ST\_AsFlatGeobuf** - Verfügbarkeit: 3.2.0 Rückgabe einer FlatGeobuf-Darstellung einer Reihe von Zeilen.

- **ST\_Contour** - Verfügbarkeit: 3.2.0 Erzeugt einen Satz von Vektorkonturen aus dem angegebenen Rasterband unter Verwendung des GDAL-Konturierungsalgorithmus.
- **ST\_DumpSegments** - Verfügbarkeit: 3.2.0 Gibt eine Zusammenfassung des Inhalts einer Geometrie wieder.
- **ST\_FromFlatGeobuf** - Verfügbarkeit: 3.2.0 Liest FlatGeobuf-Daten.
- **ST\_FromFlatGeobufToTable** - Verfügbarkeit: 3.2.0 Erstellt eine Tabelle auf der Grundlage der Struktur der FlatGeobuf-Daten.
- **ST\_InterpolateRaster** - Verfügbarkeit: 3.2.0 Interpoliert eine gerasterte Oberfläche auf der Grundlage eines Eingabesatzes von 3D-Punkten, wobei die X- und Y-Werte zur Positionierung der Punkte auf dem Gitter und der Z-Wert der Punkte als Oberflächenhöhe verwendet werden.
- **ST\_SRID** - Verfügbarkeit: 3.2.0 Gibt den räumlichen Referenzbezeichner für eine Topogeometrie zurück.
- **ST\_Scroll** - Verfügbarkeit: 3.2.0 Startpunkt eines geschlossenen LineStrings ändern.
- **ST\_SetM** - Verfügbarkeit: 3.2.0 Gibt eine Geometrie mit denselben X/Y-Koordinaten wie die Eingabegeometrie zurück, wobei die Werte aus dem Raster mit dem gewünschten Resample-Algorithmus in die Dimension M kopiert werden.
- **ST\_SetZ** - Verfügbarkeit: 3.2.0 Gibt eine Geometrie mit denselben X/Y-Koordinaten wie die Eingabegeometrie zurück, wobei die Werte aus dem Raster mit dem gewünschten Resample-Algorithmus in die Z-Dimension kopiert werden.
- **TopoGeom\_addTopoGeom** - Verfügbarkeit: 3.2 Fügt Element einer TopoGeometry zur Definition einer anderen TopoGeometry hinzu.
- **ValidateTopologyRelation** - Verfügbarkeit: 3.2.0 Gibt Informationen über ungültige Topologiebeziehungssätze zurück
- **postgis.gdal\_vsi\_options** - Verfügbarkeit: 3.2.0 Eine boolesche Konfigurationsmöglichkeit um den Zugriff auf out-db Rasterbänder zu ermöglichen

#### Erweiterte Funktionen in PostGIS 3.2

- **GetFaceByPoint** - Verbessert: 3.2.0 effizientere Implementierung und klarerer Vertrag, funktioniert nicht mehr mit ungültigen Topologien. Findet eine Fläche, die einen bestimmten Punkt schneidet.
- **ST\_ClusterKMeans** - Verbessert: 3.2.0 Unterstützung für max\_radius Fensterfunktion, die eine Cluster-ID für jede Eingabegeometrie unter Verwendung des K-Means-Algorithmus zurückgibt.
- **ST\_MakeValid** - Verbessert: 3.2.0, zusätzliche Algorithmus-Optionen, 'Linienwerk' und 'Struktur', die GEOS >= 3.10.0 erfordern. Versucht, eine ungültige Geometrie gültig zu machen, ohne dass Scheitelpunkte verloren gehen.
- **ST\_PixelAsCentroid** - Verbessert: 3.2.0 Schneller jetzt in C implementiert. Gibt den geometrischen Schwerpunkt (Punktgeometrie) der Fläche aus, die durch das Pixel repräsentiert wird.
- **ST\_PixelAsCentroids** - Verbessert: 3.2.0 Schneller jetzt in C implementiert. Gibt den geometrischen Schwerpunkt (Punktgeometrie) für jedes Pixel des Rasterbandes zurück, zusammen mit dem Zellwert und den X- und Y-Rasterkoordinaten eines jeden Pixels. Die Koordinaten der Punkte entsprechen dem geometrischen Schwerpunkt der Pixel.
- **ST\_Point** - Verbessert: 3.2.0 srid wurde als zusätzliches optionales Argument hinzugefügt. Ältere Installationen erfordern die Kombination mit ST\_SetSRID, um das Raster auf der Geometrie zu markieren. Erzeugt einen Punkt mit X-, Y- und SRID-Werten.
- **ST\_PointM** - Verbessert: 3.2.0 srid wurde als zusätzliches optionales Argument hinzugefügt. Ältere Installationen erfordern die Kombination mit ST\_SetSRID, um das Raster auf der Geometrie zu markieren. Erzeugt einen Punkt mit den Werten X, Y, M und SRID.
- **ST\_PointZ** - Verbessert: 3.2.0 srid wurde als zusätzliches optionales Argument hinzugefügt. Ältere Installationen erfordern die Kombination mit ST\_SetSRID, um das Raster auf der Geometrie zu markieren. Erzeugt einen Punkt mit X-, Y-, Z- und SRID-Werten.
- **ST\_PointZM** - Verbessert: 3.2.0 srid wurde als zusätzliches optionales Argument hinzugefügt. Ältere Installationen erfordern die Kombination mit ST\_SetSRID, um das Raster auf der Geometrie zu markieren. Erzeugt einen Punkt mit den Werten X, Y, Z, M und SRID.



- **ST\_RemovePoint** - Verbessert: 3.2.0 Einen Punkt aus einem Linienzug entfernen.
- **ST\_RemoveRepeatedPoints** - Verbessert: 3.2.0 Gibt eine Version einer Geometrie zurück, bei der doppelte Punkte entfernt wurden.
- **ST\_StartPoint** - Verbessert: 3.2.0 gibt einen Punkt für alle Geometrien zurück. Vorheriges Verhalten gibt NULLs zurück, wenn die Eingabe kein LineString war. Gibt den ersten Punkt eines LineString zurück.
- **ST\_Value** - Verbessert: 3.2.0 Das optionale Argument resample wurde hinzugefügt. Gibt den Zellwert eines Pixels aus, das über columnx und rowy oder durch einen bestimmten geometrischen Punkt angegeben wird. Die Bandnummern beginnen mit 1 und wenn keine Bandnummer angegeben ist, dann wird Band 1 angenommen. Wenn exclude\_nodata\_value auf FALSE gesetzt ist, werden auch die Pixel mit einem nodata Wert mit einbezogen. Wenn exclude\_nodata\_value nicht übergeben wird, dann wird er über die Metadaten des Rasters ausgelesen.
- **TopoGeo\_AddLineString** - Enhanced: 3.2.0 added support for returning signed identifier. Adds a linestring to an existing topology using a tolerance and possibly splitting existing edges/faces.

#### Geänderte Funktionen in PostGIS 3.2

- **ST\_Boundary** - Geändert: 3.2.0 Unterstützung für TIN, verwendet keine Geos, linearisiert keine Kurven Gibt die abgeschlossene Hülle aus der kombinierten Begrenzung der Geometrie zurück.
- **ValidateTopology** - Geändert: 3.2.0 fügte den optionalen bbox-Parameter hinzu und führte Prüfungen der Flächenbeschriftung und der Kantenverknüpfung durch. Liefert eine Menge validate\_topology\_returntype Objekte, die Probleme mit der Topologie beschreiben.

### 13.12.5 PostGIS-Funktionen neu oder erweitert in 3.1

Die unten aufgeführten Funktionen sind PostGIS-Funktionen, die hinzugefügt oder erweitert wurden.

#### Neue Funktionen in PostGIS 3.1

- **ST\_Hexagon** - Verfügbarkeit: 2.1.0 Liefert ein einzelnes Sechseck unter Verwendung der angegebenen Kantengröße und Zellkoordinate innerhalb des Sechseck-Gitterraums.
- **ST\_HexagonGrid** - Verfügbarkeit: 2.1.0 Gibt eine Menge von Sechsecken und Zellindizes zurück, die die Grenzen des Arguments Geometrie vollständig abdecken.
- **ST\_MaximumInscribedCircle** - Verfügbarkeit: 3.1.0. Berechnet die konvexe Hülle einer Geometrie.
- **ST\_ReducePrecision** - Verfügbarkeit: 3.1.0. Gibt eine gültige Geometrie mit auf eine Rastertoleranz gerundeten Punkten zurück.
- **ST\_Square** - Verfügbarkeit: 2.1.0 Gibt ein einzelnes Quadrat mit der angegebenen Kantengröße und Zellkoordinate innerhalb des quadratischen Gitterraums zurück.
- **ST\_SquareGrid** - Verfügbarkeit: 2.1.0 Gibt eine Menge von Gitterquadraten und Zellindizes zurück, die die Grenzen des Arguments Geometrie vollständig abdecken.

#### Erweiterte Funktionen in PostGIS 3.1

- **ST\_AsEWKT** - Verbessert: 3.1.0 Unterstützung für optionale Präzisionsparameter. Gibt die Well-known-Text(WKT) Darstellung der Geometrie mit den SRID-Metadaten zurück.
- **ST\_ClusterKMeans** - Verbessert: 3.1.0 Unterstützung für 3D-Geometrien und Gewichte Fensterfunktion, die eine Cluster-ID für jede Eingabegeometrie unter Verwendung des K-Means-Algorithmus zurückgibt.
- **ST\_Difference** - Verbessert: 3.1.0 akzeptiert einen gridSize-Parameter. Berechnet eine Geometrie, die den Teil der Geometrie A darstellt, der die Geometrie B nicht schneidet.



- **ST\_Intersection** - Verbessert: 3.1.0 akzeptiert einen gridSize Parameter Berechnet eine Geometrie, die den gemeinsamen Teil der Geometrien A und B darstellt.
- **ST\_MakeValid** - Verbessert: 3.1.0, Entfernen von Koordinaten mit NaN-Werten hinzugefügt. Versucht, eine ungültige Geometrie gültig zu machen, ohne dass Scheitelpunkte verloren gehen.
- **ST\_Subdivide** - Verbessert: 3.1.0 akzeptiert einen gridSize-Parameter. Berechnet eine geradlinige Unterteilung einer Geometrie.
- **ST\_SymDifference** - Verbessert: 3.1.0 akzeptiert einen gridSize-Parameter. Berechnet eine Geometrie, die die Teile der Geometrien A und B darstellt, die sich nicht überschneiden.
- **ST\_TileEnvelope** - Erweiterung: 2.0.0 Standardwert für den optionalen Parameter SRID eingefügt. Erzeugt ein rechteckiges Polygon in Web Mercator (SRID:3857) unter Verwendung des XYZ-Kachelsystems.
- **ST\_UnaryUnion** - Verbessert: 3.1.0 akzeptiert einen gridSize-Parameter. Berechnet die Vereinigung der Komponenten einer einzelnen Geometrie.
- **ST\_Union** - Verbessert: 3.1.0 akzeptiert einen gridSize-Parameter. Berechnet eine Geometrie, die die Punktmengenvereinigung der Eingabegeometrien darstellt.

#### Geänderte Funktionen in PostGIS 3.1

- **ST\_Count** - Geändert: 3.1.0 - Die ST\_Count(rastertable, rastercolumn, ...) Varianten wurden entfernt. Verwenden Sie stattdessen . Gibt die Anzahl der Pixel für ein Band eines Rasters oder eines Raster-Coverage zurück. Wenn kein Band angegeben ist, wird standardmäßig Band 1 gewählt. Wenn der Parameter "exclude\_nodata\_value" auf TRUE gesetzt ist, werden nur Pixel mit Werten ungleich NODATA gezählt.
- **ST\_Force3D** - Geändert: 3.1.0. Unterstützung für die Angabe eines Z-Wertes ungleich Null wurde zugefügt. Zwingt die Geometrien in einen XYZ Modus. Dies ist ein Alias für ST\_Force3DZ.
- **ST\_Force3DM** - Geändert: 3.1.0. Unterstützung für die Angabe eines M-Wertes ungleich Null wurde hinzugefügt. Zwingt die Geometrien in einen XYM Modus.
- **ST\_Force3DZ** - Geändert: 3.1.0. Unterstützung für die Angabe eines Z-Wertes ungleich Null wurde zugefügt. Zwingt die Geometrien in einen XYZ Modus.
- **ST\_Force4D** - Geändert: 3.1.0. Unterstützung für die Angabe von Z- und M-Werten ungleich Null wurde hinzugefügt. Zwingt die Geometrien in einen XYZM Modus.
- **ST\_Histogram** - Geändert: 3.1.0 Die Variante ST\_Histogram(table\_name, column\_name) wurde entfernt. Gibt Datensätze aus, welche die Verteilung der Daten eines Rasters oder eines Rastercoverage darstellen. Dabei wird die Wertemenge in Klassen aufgeteilt und für jede Klasse zusammengefasst. Wenn die Anzahl der Klassen nicht angegeben ist, wird sie automatisch berechnet.
- **ST\_Quantile** - Geändert: 3.1.0 Die Variante ST\_Quantile(table\_name, column\_name) wurde entfernt. Berechnet die Quantile eines Rasters oder einer Rastercoverage Tabelle im Kontext von Stichproben oder Bevölkerung. Dadurch kann untersucht werden, ob ein Wert bei 25%, 50% oder 75% Perzentil des Rasters liegt.
- **ST\_SummaryStats** - Geändert: 3.1.0 ST\_SummaryStats(rastertable, rastercolumn, ...) Varianten wurden entfernt. Verwenden Sie stattdessen . 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.

### 13.12.6 PostGIS-Funktionen neu oder erweitert in 3.0

Die unten aufgeführten Funktionen sind PostGIS-Funktionen, die hinzugefügt oder erweitert wurden.

Neue Funktionen in PostGIS 3.0

- **CG\_ConstrainedDelaunayTriangles** - Verfügbarkeit: 2.1.0 Gibt eine eingeschränkte Delaunay-Triangulation um die angegebene Eingabegeometrie zurück.
- **ST\_3DLineInterpolatePoint** - Verfügbarkeit: 2.1.0 Gibt einen oder mehrere, entlang einer Linie interpolierte Punkte zurück.
- **ST\_ConstrainedDelaunayTriangles** - Verfügbarkeit: 2.1.0 Gibt eine eingeschränkte Delaunay-Triangulation um die angegebene Eingabegeometrie zurück.
- **ST\_TileEnvelope** - Verfügbarkeit: 2.1.0 Erzeugt ein rechteckiges Polygon in Web Mercator (SRID:3857) unter Verwendung des XYZ-Kachelsystems.

#### Erweiterte Funktionen in PostGIS 3.0

- **ST\_AsMVT** - Erweiterung: 3.0 - Unterstützung für eine Feature-ID. Aggregatfunktion, die eine MVT-Darstellung einer Reihe von Zeilen zurückgibt.
- **ST\_Contains** - Verbessert: 3.0.0 ermöglicht die Unterstützung von GEOMETRYCOLLECTION Tests, wenn jeder Punkt von B in A liegt und ihre Innenräume einen gemeinsamen Punkt haben
- **ST\_ContainsProperly** - Verbessert: 3.0.0 ermöglicht die Unterstützung von GEOMETRYCOLLECTION Prüft, ob jeder Punkt von B im Inneren von A liegt
- **ST\_CoveredBy** - Verbessert: 3.0.0 ermöglicht die Unterstützung von GEOMETRYCOLLECTION Prüft, ob jeder Punkt von A in B liegt
- **ST\_Covers** - Verbessert: 3.0.0 ermöglicht die Unterstützung von GEOMETRYCOLLECTION Prüft, ob jeder Punkt von B in A liegt
- **ST\_Crosses** - Verbessert: 3.0.0 ermöglicht die Unterstützung von GEOMETRYCOLLECTION Prüft, ob zwei Geometrien einige, aber nicht alle, innere Punkte gemeinsam haben
- **ST\_CurveToLine** - Erweiterung: 3.0.0 führte eine minimale Anzahl an Segmenten pro linearisierten Bogen ein, um einem topologischen Kollaps vorzubeugen. Konvertiert eine Geometrie mit Kurven in eine lineare Geometrie.
- **ST\_Disjoint** - Verbessert: 3.0.0 ermöglicht die Unterstützung von GEOMETRYCOLLECTION Prüft, ob zwei Geometrien keine gemeinsamen Punkte haben
- **ST\_Equals** - Verbessert: 3.0.0 ermöglicht die Unterstützung von GEOMETRYCOLLECTION Prüft, ob zwei Geometrien dieselbe Menge von Punkten enthalten
- **ST\_GeneratePoints** - Erweiterung: mit 3.0.0 wurde das Argument "seed" hinzugefügt Erzeugt einen Multipunkt aus zufälligen Punkten, die in einem Polygon oder MultiPolygon enthalten sind.
- **ST\_GeomFromGeoJSON** - Verbessert: 3.0.0 Geometrie wird standardmäßig auf SRID=4326 gesetzt, wenn nicht anders angegeben. Nimmt als Eingabe eine GeoJSON-Darstellung der Geometrie und gibt ein geometrisches PostGIS-Objekt aus.
- **ST\_LocateBetween** - Verbessert: 3.0.0 - Unterstützung für POLYGON, TIN, TRIANGLE hinzugefügt. Gibt die Teile einer Geometrie zurück, die einem Messbereich entsprechen.
- **ST\_LocateBetweenElevations** - Verbessert: 3.0.0 - Unterstützung für POLYGON, TIN, TRIANGLE hinzugefügt. Gibt die Teile einer Geometrie zurück, die in einem Höhenbereich (Z) liegen.
- **ST\_Overlaps** - Verbessert: 3.0.0 ermöglicht die Unterstützung von GEOMETRYCOLLECTION Prüft, ob zwei Geometrien die gleiche Abmessung haben und sich schneiden, aber jede mindestens einen Punkt hat, der nicht in der anderen liegt
- **ST\_Relate** - Verbessert: 3.0.0 ermöglicht die Unterstützung von GEOMETRYCOLLECTION Prüft, ob zwei Geometrien eine topologische Beziehung haben, die einem Schnittpunktmatrixmuster entspricht, oder berechnet ihre Schnittpunktmatrix
- **ST\_Segmentize** - Verbessert: 3.0.0 Segmentize-Geometrie erzeugt jetzt Teilsegmente gleicher Länge Gibt eine geänderte Geometrie/Geografie zurück, bei der kein Segment länger als eine bestimmte Entfernung ist.
- **ST\_Touches** - Verbessert: 3.0.0 ermöglicht die Unterstützung von GEOMETRYCOLLECTION Prüft, ob zwei Geometrien mindestens einen Punkt gemeinsam haben, aber ihre Innenräume sich nicht schneiden

- **ST\_Within** - Verbessert: 3.0.0 ermöglicht die Unterstützung von GEOMETRYCOLLECTION Tests, wenn jeder Punkt von A in B liegt und ihre Innenräume einen gemeinsamen Punkt haben

#### Geänderte Funktionen in PostGIS 3.0

- **PostGIS\_Extensions\_Upgrade** - Geändert: 3.0.0, um lose Erweiterungen neu zu packen und postgis\_raster zu unterstützen. Packt und aktualisiert PostGIS-Erweiterungen (z.B. postgis\_raster, postgis\_topology, postgis\_sfcgal) auf die angegebene oder neueste Version.
- **ST\_3DDistance** - Geändert: 3.0.0 - SFCGAL-Version entfernt Für den geometrischen Datentyp. Es wird der geringste 3-dimensionale kartesische Abstand (basierend auf dem Koordinatenreferenzsystem) zwischen zwei geometrischen Objekten in projizierten Einheiten zurückgegeben.
- **ST\_3DIntersects** - Änderung: 3.0.0 das SFCGAL Back-end wurde entfernt, das GEOS Back-end unterstützt TIN. Prüft, ob sich zwei Geometrien in 3D räumlich schneiden - nur für Punkte, Linienzüge, Polygone, polyedrische Flächen (Bereich)
- **ST\_Area** - Geändert: 3.0.0 - hängt nicht mehr von SFCGAL ab. Gibt den geometrischen Schwerpunkt einer Geometrie zurück.
- **ST\_AsGeoJSON** - Änderung: 3.0.0 Unterstützung von Datensätzen bei der Eingabe Rückgabe einer Geometrie oder eines Merkmals im GeoJSON-Format.
- **ST\_AsGeoJSON** - Änderung: 3.0.0 Ausgabe der SRID wenn nicht EPSG:4326 Rückgabe einer Geometrie oder eines Merkmals im GeoJSON-Format.
- **ST\_AsKML** - Geändert: 3.0.0 - Die Signatur der "versionierten" Variante wurde entfernt. Gibt die Geometrie als GML-Element - Version 2 oder 3 - zurück.
- **ST\_Distance** - Geändert: 3.0.0 - hängt nicht mehr von SFCGAL ab. Gibt die größte 3-dimensionale Distanz zwischen zwei geometrischen Objekten als Linie zurück
- **ST\_Intersection** - Geändert: 3.0.0 ist nicht von SFCGAL abhängig. Berechnet eine Geometrie, die den gemeinsamen Teil der Geometrien A und B darstellt.
- **ST\_Intersects** - Geändert: 3.0.0 SFCGAL Version entfernt und native Unterstützung für 2D TINS hinzugefügt. Prüft, ob sich zwei Geometrien schneiden (sie haben mindestens einen Punkt gemeinsam)
- **ST\_Union** - Geändert: 3.0.0 ist nicht von SFCGAL abhängig. Berechnet eine Geometrie, die die Punktmengenvereinigung der Eingabegeometrien darstellt.

### 13.12.7 PostGIS-Funktionen neu oder erweitert in 2.5

Die unten aufgeführten Funktionen sind PostGIS-Funktionen, die hinzugefügt oder erweitert wurden.

#### Neue Funktionen in PostGIS 2.5

- **PostGIS\_Extensions\_Upgrade** - Verfügbarkeit: 2.5.0 Packt und aktualisiert PostGIS-Erweiterungen (z.B. postgis\_raster, postgis\_topology, postgis\_sfcgal) auf die angegebene oder neueste Version.
  - **ST\_Angle** - Verfügbarkeit: 2.5.0 Gibt den Winkel zwischen 3 Punkten oder zwischen 2 Vektoren (4 Punkte oder 2 Linien) zurück.
  - **ST\_AsHexWKB** - Verfügbarkeit: 2.5.0 Gibt die Well-known-Binary (WKB) Hex-Darstellung eines Rasters zurück.
  - **ST\_BandFileSize** - Verfügbarkeit: 2.5.0 Gibt die Dateigröße eines im Dateisystem gespeicherten Bandes aus. Wenn "bandnum" nicht angegeben ist, wird 1 angenommen.
  - **ST\_BandFileTimestamp** - Verfügbarkeit: 2.5.0 Gibt den Zeitstempel eines im Dateisystem gespeicherten Bandes aus. Wenn "bandnum" nicht angegeben ist, wird 1 angenommen.
  - **ST\_ChaikinSmoothing** - Verfügbarkeit: 2.5.0 Gibt eine geglättete Version einer Geometrie zurück, die den Chaikin-Algorithmus verwendet
-

- **ST\_FilterByM** - Verfügbarkeit: 2.5.0 Entfernt Scheitelpunkte basierend auf ihrem M-Wert
- **ST\_Grayscale** - Verfügbarkeit: 2.5.0 Erzeugt einen neuen Raster mit einem 8BUI-Band aus dem Ausgangsraster und den angegebenen Bändern für Rot, Grün und Blau
- **ST\_LineInterpolatePoints** - Verfügbarkeit: 2.5.0 Gibt einen oder mehrere, entlang einer Linie interpolierte Punkte zurück.
- **ST\_OrientedEnvelope** - Verfügbarkeit: 2.5.0. Gibt ein Rechteck mit minimalem Flächeninhalt zurück, das eine Geometrie enthält.
- **ST\_QuantizeCoordinates** - Verfügbarkeit: 2.5.0 Setzt die niedrigwertigsten Bits der Koordinaten auf Null
- **ST\_RastFromHexWKB** - Verfügbarkeit: 2.5.0 Gibt einen Rasterwert von einer Well-known-Binary (WKB) Hex-Darstellung eines Rasters zurück.
- **ST\_RastFromWKB** - Verfügbarkeit: 2.5.0 Gibt einen Rasterwert von einer Well-known-Binary (WKB) Darstellung eines Rasters zurück.
- **ST\_SetBandIndex** - Verfügbarkeit: 2.5.0 Aktualisiert die externe Bandnummer eines out-db Bandes.
- **ST\_SetBandPath** - Verfügbarkeit: 2.5.0 Aktualisiert den externen Dateipfad und die Bandnummer eines out-db Bandes.

#### Erweiterte Funktionen in PostGIS 2.5

- **ST\_AsBinary/ST\_AsWKB** - Erweiterung: 2.5.0 ST\_AsWKB hinzugefügt Gibt die Well-known-Binary (WKB) Darstellung eines Rasters zurück.
- **ST\_AsMVT** - Erweiterung: 2.5.0 - Unterstützung von nebenläufigen Abfragen. Aggregatfunktion, die eine MVT-Darstellung einer Reihe von Zeilen zurückgibt.
- **ST\_AsText** - Erweiterung: 2.5 - der optionale Parameter "precision" wurde eingeführt. Gibt die Well-known-Text(WKT) Darstellung der Geometrie/Geographie ohne die SRID Metadaten zurück.
- **ST\_BandMetaData** - Erweiterung: 2.5.0 inkludiert jetzt outdbbandnum, filesize und filetimestamp für outdb Raster. Gibt die grundlegenden Metadaten eines bestimmten Rasterbandes aus. Wenn der Parameter "bandnum" nicht angegeben ist, wird das 1ste Band angenommen.
- **ST\_Buffer** - Erweiterung: 2.5.0 - ST\_Buffer ermöglicht jetzt auch eine seitliche Pufferzonenberechnung über side=bothleftright. Berechnet eine Geometrie, die alle Punkte innerhalb eines bestimmten Abstands zu einer Geometrie umfasst.
- **ST\_GeomFromGeoJSON** - Erweiterung: 2.5.0 unterstützt nun auch die Eingabe von json und jsonb. Nimmt als Eingabe eine GeoJSON-Darstellung der Geometrie und gibt ein geometrisches PostGIS-Objekt aus.
- **ST\_GeometricMedian** - Erweiterung: 2.5.0 Unterstützung für M zur Gewichtung nach Punkten. Gibt den geometrischen Median eines Mehrfachpunktes zurück.
- **ST\_Intersects** - Verbessert: 2.5.0 Unterstützt GEOMETRYCOLLECTION. Prüft, ob sich zwei Geometrien schneiden (sie haben mindestens einen Punkt gemeinsam)
- **ST\_OffsetCurve** - Erweiterung: ab 2.5 wird auch GEOMETRYCOLLECTION und MULTILINESTRING unterstützt. Gibt eine versetzte Linie in einem bestimmten Abstand und einer bestimmten Seite von einer Eingabelinie zurück.
- **ST\_Scale** - Verbessert: In Version 2.5.0 wurde die Unterstützung für die Skalierung relativ zu einem lokalen Ursprung (Parameter origin) eingeführt. Skaliert eine Geometrie um bestimmte Faktoren.
- **ST\_Split** - Verbessert: In Version 2.5.0 wurde die Unterstützung für die Aufteilung eines Polygons durch eine Mehrlinie eingeführt. Gibt eine Sammlung von Geometrien zurück, die durch Aufteilung einer Geometrie durch eine andere Geometrie entstanden sind.
- **ST\_Subdivide** - Verbessert: 2.5.0 verwendet vorhandene Punkte bei der Polygonaufteilung wieder, die Anzahl der Scheitelpunkte wurde von 8 auf 5 gesenkt. Berechnet eine geradlinige Unterteilung einer Geometrie.

#### Geänderte Funktionen in PostGIS 2.5

- **ST\_GDALDrivers** - Änderung: 2.5.0 - die Spalten can\_read und can\_write hinzugefügt. Gibt eine Liste der Rasterformate aus, die von PostGIS über die Bibliothek GDAL unterstützt werden. Nur die Formate mit can\_write=True können von ST\_AsGDALRaster verwendet werden.

### 13.12.8 PostGIS-Funktionen neu oder erweitert in 2.4

Die unten aufgeführten Funktionen sind PostGIS-Funktionen, die hinzugefügt oder erweitert wurden.

Neue Funktionen in PostGIS 2.4

- **ST\_AsGeobuf** - Verfügbarkeit: 2.4.0 Gibt eine Menge an Zeilen in der Geobuf Darstellung aus.
- **ST\_AsMVT** - Verfügbarkeit: 2.4.0 Aggregatfunktion, die eine MVT-Darstellung einer Reihe von Zeilen zurückgibt.
- **ST\_AsMVTGeom** - Verfügbarkeit: 2.4.0 Transformiert eine Geometrie in den Koordinatenraum einer MVT-Kachel.
- **ST\_Centroid** - Verfügbarkeit: Mit 2.4.0 wurde die Unterstützung für den geographischen Datentyp eingeführt. Gibt den geometrischen Schwerpunkt einer Geometrie zurück.
- **ST\_ForcePolygonCCW** - Verfügbarkeit: 2.4.0 Richtet alle äußeren Ringe gegen den Uhrzeigersinn und alle inneren Ringe mit dem Uhrzeigersinn aus.
- **ST\_ForcePolygonCW** - Verfügbarkeit: 2.4.0 Richtet alle äußeren Ringe im Uhrzeigersinn und alle inneren Ringe gegen den Uhrzeigersinn aus.
- **ST\_FrechetDistance** - Verfügbarkeit: 2.4.0 - benötigt GEOS >= 3.7.0 Gibt den kürzesten 3-dimensionalen Abstand zwischen zwei geometrischen Objekten als Linie zurück
- **ST\_IsPolygonCCW** - Verfügbarkeit: 2.4.0 Gibt TRUE zurück, wenn alle äußeren Ringe gegen den Uhrzeigersinn orientiert sind und alle inneren Ringe im Uhrzeigersinn ausgerichtet sind.
- **ST\_IsPolygonCW** - Verfügbarkeit: 2.4.0 Gibt den Wert TRUE zurück, wenn alle äußeren Ringe im Uhrzeigersinn und alle inneren Ringe gegen den Uhrzeigersinn ausgerichtet sind.
- **ST\_MakeEmptyCoverage** - Verfügbarkeit: 2.4.0 Bedeckt die georeferenzierte Fläche mit einem Gitter aus leeren Rasterkacheln.

Erweiterte Funktionen in PostGIS 2.4

- **Loader\_Generate\_Nation\_Script** - Verbessert: 2.4.1 zip code 5 tabulation area (zcta5) Ladeschritt wurde korrigiert und wenn aktiviert, werden zcta5 Daten als eine einzige Tabelle namens zcta5\_all als Teil des Nationskripts geladen. Erzeugt für die angegebene Plattform ein Shell-Skript, welches die County und State Lookup Tabellen ladet.
- **Normalize\_Address** - Erweiterung: 2.4.0 die zusätzlichen Felder "zip4" und "address\_alphanumeric" wurden zum Objekt "norm\_addy" hinzugefügt. Für einen gegebenen Adressentext wird der zusammengesetzte Datentyp norm\_addy zurückgegeben, der ein Suffix und ein Präfix für die Straße, einen normierten Datentyp, die Straße, den Straßennamen etc. enthält und diese einzelnen Attributen zuweist. Diese Funktion benötigt lediglich die "lookup data", die mit dem Tiger Geokodierer paketiert sind (Tiger Census Daten werden nicht benötigt).
- **Page\_Normalize\_Address** - Erweiterung: 2.4.0 die zusätzlichen Felder "zip4" und "address\_alphanumeric" wurden zum Objekt "norm\_addy" hinzugefügt. Für einen gegebenen Adressentext wird der zusammengesetzte Datentyp norm\_addy zurückgegeben, der ein Suffix und ein Präfix für die Straße, einen normierten Datentyp, die Straße, den Straßennamen etc. enthält und diese einzelnen Attributen zuweist. Diese Funktion benötigt lediglich die "lookup data", die mit dem Tiger Geokodierer paketiert sind (Tiger Census Daten werden nicht benötigt). Benötigt die Erweiterung "address\_standardizer".
- **Reverse\_Geocode** - Verbessert: 2.4.1 Wenn der optionale zcta5-Datensatz geladen ist, kann die Funktion reverse\_geocode nach state und zip auflösen, auch wenn die spezifischen Statusdaten nicht geladen sind. Siehe für Einzelheiten zum Laden von zcta5-Daten. Nimmt einen geometrischen Punkt in einem bekannten Koordinatenreferenzsystem entgegen und gibt einen Datensatz zurück, das ein Feld mit theoretisch möglichen Adressen und ein Feld mit Straßenkreuzungen beinhaltet. Wenn include\_stnum\_range = true, dann beinhalten die Straßenkreuzungen den "Street Range" (Kennung des Straßenabschnitts).
- **ST\_AsTWKB** - Erweiterung: 2.4.0 Hauptspeicher- und Geschwindigkeitsverbesserungen. Gibt die Geometrie als TWKB, aka "Tiny Well-known Binary" zurück
- **ST\_Covers** - Verbessert: 2.4.0 Unterstützung für Polygon in Polygon und Linie in Polygon für Geografietypen hinzugefügt Prüft, ob jeder Punkt von B in A liegt

- **ST\_CurveToLine** - Erweiterung: ab 2.4.0 kann die Toleranz über die 'maximale Abweichung' und den 'maximalen Winkel' angegeben werden. Die symmetrische Ausgabe wurde hinzugefügt. Konvertiert eine Geometrie mit Kurven in eine lineare Geometrie.
- **ST\_Project** - Verbessert: 2.4.0 Erlaubt negative Entfernungen und nicht-normierte Azimute. Gibt einen Punkt zurück, der von einem Startpunkt um eine bestimmte Entfernung und Peilung (Azimut) projiziert wird.
- **ST\_Reverse** - Erweiterung: mit 2.4.0 wurde die Unterstützung für Kurven eingeführt. Gibt die Geometrie in umgekehrter Knotenreihenfolge zurück.

#### Geänderte Funktionen in PostGIS 2.4

- **=** - Änderung: 2.4.0, in Vorgängerversionen war dies die Gleichheit der umschreibenden Rechtecke, nicht die geometrische Gleichheit. Falls Sie auf Gleichheit der umschreibenden Rechtecke prüfen wollen, verwenden Sie stattdesse bitte `ST_Equals`. Gibt TRUE zurück, wenn die Koordinaten und die Reihenfolge der Koordinaten der Geometrie/Geographie A und der Geometrie/Geographie B ident sind.
- **ST\_Node** - Geändert: 2.4.0 verwendet diese Funktion intern `GEOSNode` anstelle von `GEOSUnaryUnion`. Dies kann dazu führen, dass die resultierenden Linestrings eine andere Reihenfolge und Richtung haben als in PostGIS < 2.4. Knoten eine Sammlung von Linien.

### 13.12.9 PostGIS-Funktionen neu oder erweitert in 2.3

Die unten aufgeführten Funktionen sind PostGIS-Funktionen, die hinzugefügt oder erweitert wurden.

#### Neue Funktionen in PostGIS 2.3

- **&&&(geometry,gidx)** - Verfügbarkeit: Mit 2.3.0 wurde die Unterstützung von Block Range INDEXes (BRIN) eingeführt. Erfordert PostgreSQL 9.5+. Gibt TRUE zurück, wenn die (cached) n-D bounding box einer Geometrie eine n-D float precision bounding box (GIDX) schneidet.
- **&&&(gidx,geometry)** - Verfügbarkeit: Mit 2.3.0 wurde die Unterstützung von Block Range INDEXes (BRIN) eingeführt. Erfordert PostgreSQL 9.5+. Gibt TRUE zurück, wenn eine n-D float precision bounding box (GIDX) eine (cached) n-D bounding box einer Geometrie schneidet.
- **&&&(gidx,gidx)** - Verfügbarkeit: Mit 2.3.0 wurde die Unterstützung von Block Range INDEXes (BRIN) eingeführt. Erfordert PostgreSQL 9.5+. Gibt TRUE zurück, wenn sich zwei n-D float precision bounding boxes (GIDX) gegenseitig überschneiden.
- **&&(box2df,box2df)** - Verfügbarkeit: Mit 2.3.0 wurde die Unterstützung von Block Range INDEXes (BRIN) eingeführt. Erfordert PostgreSQL 9.5+. Gibt TRUE zurück, wenn sich zwei 2D float precision Bounding Boxes (BOX2DF) überschneiden.
- **&&(box2df,geometry)** - Verfügbarkeit: Mit 2.3.0 wurde die Unterstützung von Block Range INDEXes (BRIN) eingeführt. Erfordert PostgreSQL 9.5+. Gibt TRUE zurück, wenn eine 2D float precision bounding box (BOX2DF) eine Geometrie (cached) 2D bounding box schneidet.
- **&&(geometry,box2df)** - Verfügbarkeit: Mit 2.3.0 wurde die Unterstützung von Block Range INDEXes (BRIN) eingeführt. Erfordert PostgreSQL 9.5+. Gibt TRUE zurück, wenn sich die 2D Bounding Box (cached) einer Geometrie mit einer 2D Bounding Box mit Gleitpunktgenauigkeit (BOX2DF) überschneidet.
- **@(box2df,box2df)** - Verfügbarkeit: Mit 2.3.0 wurde die Unterstützung von Block Range INDEXes (BRIN) eingeführt. Erfordert PostgreSQL 9.5+. Gibt TRUE zurück, wenn eine 2D float precision bounding box (BOX2DF) innerhalb einer anderen 2D float precision bounding box enthalten ist.
- **@(box2df,geometry)** - Verfügbarkeit: Mit 2.3.0 wurde die Unterstützung von Block Range INDEXes (BRIN) eingeführt. Erfordert PostgreSQL 9.5+. Gibt TRUE zurück, wenn eine 2D float precision bounding box (BOX2DF) in der 2D Bounding Box einer Geometrie enthalten ist..
- **@(geometry,box2df)** - Verfügbarkeit: Mit 2.3.0 wurde die Unterstützung von Block Range INDEXes (BRIN) eingeführt. Erfordert PostgreSQL 9.5+. Gibt TRUE zurück, wenn die 2D Bounding Box einer Geometrie in einer 2D float precision Bounding Box (BOX2DF) enthalten ist.



- **Populate\_Topology\_Layer** - Verfügbarkeit: 2.3.0 Fügt fehlende Einträge zu der Tabelle topology.layer hinzu, indem Metadaten aus den topologischen Tabellen ausgelesen werden.
- **ST\_ClusterDBSCAN** - Verfügbarkeit: 2.3.0 Fensterfunktion, die eine Cluster-ID für jede Eingabegeometrie unter Verwendung des DBSCAN-Algorithmus zurückgibt.
- **ST\_ClusterKMeans** - Verfügbarkeit: 2.3.0 Fensterfunktion, die eine Cluster-ID für jede Eingabegeometrie unter Verwendung des K-Means-Algorithmus zurückgibt.
- **ST\_GeneratePoints** - Verfügbarkeit: 2.3.0 Erzeugt einen Multipunkt aus zufälligen Punkten, die in einem Polygon oder Multi-Polygon enthalten sind.
- **ST\_GeometricMedian** - Verfügbarkeit: 2.3.0 Gibt den geometrischen Median eines Mehrfachpunktes zurück.
- **ST\_MakeLine** - Verfügbarkeit: Mit 2.3.0 wurde die Unterstützung zur Eingabe von MultiPoint Elementen eingeführt Erzeugt einen Linienzug aus einer Punkt-, Mehrfachpunkt- oder Liniengeometrie.
- **ST\_MinimumBoundingRadius** - Verfügbarkeit: 2.3.0 Gibt den Mittelpunkt und den Radius des kleinsten Kreises zurück, der eine Geometrie enthält.
- **ST\_MinimumClearance** - Verfügbarkeit: 2.3.0 Gibt das Mindestabstandsmaß für eine Geometrie zurück; ein Maß für die Robustheit einer Geometrie.
- **ST\_MinimumClearanceLine** - Verfügbarkeit: 2.3.0 - benötigt GEOS >= 3.6.0 Gibt ein Liniensegment mit zwei Punkten zurück, welche sich über das Mindestabstandsmaß erstreckt.
- **ST\_Normalize** - Verfügbarkeit: 2.3.0 Gibt die Geometrie in Normalform zurück.
- **ST\_Points** - Verfügbarkeit: 2.3.0 Gibt einen MultiPoint zurück, welcher alle Koordinaten einer Geometrie enthält.
- **ST\_VoronoiLines** - Verfügbarkeit: 2.3.0 Gibt die Grenzen des Voronoi-Diagramms der Eckpunkte einer Geometrie zurück.
- **ST\_VoronoiPolygons** - Verfügbarkeit: 2.3.0 Gibt die Zellen des Voronoi-Diagramms der Scheitelpunkte einer Geometrie zurück.
- **ST\_WrapX** - Verfügbarkeit: 2.3.0 erfordert GEOS Versammelt eine Geometrie um einen X-Wert
- **TopoGeom\_addElement** - Verfügbarkeit: 2.3 Fügt ein Element zu der Definition einer TopoGeometry hinzu.
- **TopoGeom\_remElement** - Verfügbarkeit: 2.3 Entfernt ein Element aus der Definition einer TopoGeometry.
- **~(box2df,box2df)** - Verfügbarkeit: Mit 2.3.0 wurde die Unterstützung von Block Range INdexas (BRIN) eingeführt. Erfordert PostgreSQL 9.5+. Gibt TRUE zurück, wenn eine 2D float precision bounding box (BOX2DF) eine andere 2D float precision bounding box (BOX2DF) enthält.
- **~(box2df,geometry)** - Verfügbarkeit: Mit 2.3.0 wurde die Unterstützung von Block Range INdexas (BRIN) eingeführt. Erfordert PostgreSQL 9.5+. Gibt TRUE zurück, wenn eine 2D float precision bounding box (BOX2DF) die 2D Bounding Box einer Geometrie enthält.
- **~(geometry,box2df)** - Verfügbarkeit: Mit 2.3.0 wurde die Unterstützung von Block Range INdexas (BRIN) eingeführt. Erfordert PostgreSQL 9.5+. Gibt TRUE zurück, wenn die 2D bounding box einer Geometrie eine 2D float precision bounding box (GIDX) enthält.

### Erweiterte Funktionen in PostGIS 2.3

- **ST\_Contains** - Verbessert: 2.3.0 Verbesserung des PIP-Kurzschlusses erweitert um die Unterstützung von MultiPoints mit wenigen Punkten. Frühere Versionen unterstützten nur Punkte in Polygonen. Tests, wenn jeder Punkt von B in A liegt und ihre Innenräume einen gemeinsamen Punkt haben
- **ST\_Covers** - Verbessert: 2.3.0 Verbesserung des PIP-Kurzschlusses für Geometrien, erweitert um die Unterstützung von MultiPoints mit wenigen Punkten. Frühere Versionen unterstützten nur Punkte in Polygonen. Prüft, ob jeder Punkt von B in A liegt

- **ST\_Expand** - Verbessert: In Version 2.3.0 wurde die Möglichkeit hinzugefügt, eine Box um unterschiedliche Beträge in verschiedenen Dimensionen zu erweitern. Gibt einen Begrenzungsrahmen zurück, der aus einem anderen Begrenzungsrahmen oder einer Geometrie erweitert wurde.
- **ST\_Intersects** - Verbessert: 2.3.0 Verbesserung des PIP-Kurzschlusses erweitert um die Unterstützung von MultiPoints mit wenigen Punkten. Frühere Versionen unterstützten nur Punkte in Polygonen. Prüft, ob sich zwei Geometrien schneiden (sie haben mindestens einen Punkt gemeinsam)
- **ST\_Segmentize** - Verbessert: 2.3.0 Die Segmentierung der Geografie erzeugt nun Teilsegmente gleicher Länge Gibt eine geänderte Geometrie/Geografie zurück, bei der kein Segment länger als eine bestimmte Entfernung ist.
- **ST\_Transform** - Verbessert: In Version 2.3.0 wurde die Unterstützung für direkten PROJ.4 Text eingeführt. Rückgabe einer neuen Geometrie mit in ein anderes räumliches Bezugssystem transformierten Koordinaten.
- **ST\_Within** - Verbessert: 2.3.0 Verbesserung des PIP-Kurzschlusses für Geometrien, erweitert um die Unterstützung von MultiPoints mit wenigen Punkten. Frühere Versionen unterstützten nur Punkte in Polygonen. Tests, wenn jeder Punkt von A in B liegt und ihre Innenräume einen gemeinsamen Punkt haben

### Geänderte Funktionen in PostGIS 2.3

- **ST\_PointN** - Änderung: 2.3.0 : negatives Indizieren verfügbar (-1 entspricht dem Endpunkt) Gibt die Anzahl der Stützpunkte eines ST\_LineString oder eines ST\_CircularString zurück.

## 13.12.10 PostGIS-Funktionen neu oder erweitert in 2.2

Die unten aufgeführten Funktionen sind PostGIS-Funktionen, die hinzugefügt oder erweitert wurden.

### Neue Funktionen in PostGIS 2.2

- **<<->** - Verfügbarkeit: 2.2.0 -- KNN steht erst ab PostgreSQL 9.1+ zur Verfügung. Gibt den n-D-Abstand zwischen den Geometrien oder Begrenzungsrahmen von A und B zurück
- **ST\_3DDifference** - Verfügbarkeit: 2.2.0 3D-Differenz durchführen
- **ST\_3DUnion** - Verfügbarkeit: 2.2.0 3D-Vereinigung durchführen.
- **ST\_ApproximateMedialAxis** - Verfügbarkeit: 2.2.0 Berechnet die konvexe Hülle einer Geometrie.
- **ST\_AsEncodedPolyline** - Verfügbarkeit: 2.2.0 Erzeugt eine codierte Polylinie aus einer LineString Geometrie.
- **ST\_AsTWKB** - Verfügbarkeit: 2.2.0 Gibt die Geometrie als TWKB, aka "Tiny Well-known Binary" zurück
- **ST\_BoundingDiagonal** - Verfügbarkeit: 2.2.0 Gibt die Diagonale des Umgebungsrechtecks der angegebenen Geometrie zurück.
- **ST\_CPAWithin** - Verfügbarkeit: 2.2.0 Prüft, ob der nächstgelegene Punkt der Annäherung zweier Flugbahnen innerhalb der angegebenen Entfernung liegt.
- **ST\_ClipByBox2D** - Verfügbarkeit: 2.2.0 Berechnet den Teil einer Geometrie, der innerhalb eines Rechtecks liegt.
- **ST\_ClosestPointOfApproach** - Verfügbarkeit: 2.2.0 Liefert ein Maß für den nächstgelegenen Punkt der Annäherung von zwei Flugbahnen.
- **ST\_ClusterIntersecting** - Verfügbarkeit: 2.2.0 Aggregatfunktion, die Eingabegeometrien zu zusammenhängenden Mengen clustert.
- **ST\_ClusterWithin** - Verfügbarkeit: 2.2.0 Aggregatfunktion, die Geometrien nach Trennungsabstand gruppiert.
- **ST\_CountAgg** - Verfügbarkeit: 2.2.0 Aggregatfunktion. Gibt die Anzahl der Pixel in einem bestimmten Band der Raster aus. Wenn kein Band angegeben ist, wird Band 1 angenommen. Wenn "exclude\_nodata\_value" TRUE ist, werden nur die Pixel ohne NODATA Werte gezählt.
- **ST\_CreateOverview** - Verfügbarkeit: 2.2.0 Erzeugt eine Version des gegebenen Raster-Coverage mit geringerer Auflösung.



- **ST\_DistanceCPA** - Verfügbarkeit: 2.2.0 Liefert den Abstand zwischen dem nächstgelegenen Punkt der Annäherung zweier Flugbahnen.
- **ST\_ForceCurve** - Verfügbarkeit: 2.2.0 Wandelt einen geometrischen in einen Kurven Datentyp um, soweit anwendbar.
- **ST\_IsPlanar** - Verfügbarkeit: 2.2.0: Dies war in 2.1.0 dokumentiert, wurde aber versehentlich in der Version 2.1 ausgelassen. Prüfen, ob eine Fläche planar ist oder nicht
- **ST\_IsSolid** - Verfügbarkeit: 2.2.0 Prüfen, ob die Geometrie ein Solid ist. Es wird keine Gültigkeitsprüfung durchgeführt.
- **ST\_IsValidTrajectory** - Verfügbarkeit: 2.2.0 Prüft, ob die Geometrie eine gültige Flugbahn ist.
- **ST\_LineFromEncodedPolyline** - Verfügbarkeit: 2.2.0 Erzeugt einen LineString aus einem codierten Linienzug.
- **ST\_MakeSolid** - Verfügbarkeit: 2.2.0 Gießen Sie die Geometrie in einen Körper. Es wird keine Prüfung durchgeführt. Um ein gültiges Solid zu erhalten, muss die Eingabegeometrie eine geschlossene polyedrische Fläche oder ein geschlossenes TIN sein.
- **ST\_MapAlgebra (callback function version)** - Verfügbarkeit: 2.2.0: Möglichkeit eine Maske hinzuzufügen Die Version mit der Rückruffunktion - Gibt für einen oder mehrere Eingaberaster einen Raster mit einem Band, den Bandindizes und einer vom Anwender vorgegebenen Rückruffunktion zurück.
- **ST\_MemSize** - Verfügbarkeit: 2.2.0 Gibt den Platzbedarf des Rasters (in Byte) aus.
- **ST\_RemoveRepeatedPoints** - Verfügbarkeit: 2.2.0 Gibt eine Version einer Geometrie zurück, bei der doppelte Punkte entfernt wurden.
- **ST\_Retile** - Verfügbarkeit: 2.2.0 Gibt konfigurierte Kacheln eines beliebig gekachelten Rastercoverage aus.
- **ST\_SetEffectiveArea** - Verfügbarkeit: 2.2.0 Legt die effektive Fläche für jeden Scheitelpunkt unter Verwendung des Visvalingam-Whyatt-Algorithmus fest.
- **ST\_SimplifyVW** - Verfügbarkeit: 2.2.0 Liefert eine vereinfachte Darstellung einer Geometrie unter Verwendung des Visvalingam-Whyatt-Algorithmus
- **ST\_Subdivide** - Verfügbarkeit: 2.2.0 Berechnet eine geradlinige Unterteilung einer Geometrie.
- **ST\_SummaryStatsAgg** - Verfügbarkeit: 2.2.0 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\_SwapOrdinates** - Verfügbarkeit: 2.2.0 Gibt eine Version der Ausgangsgeometrie zurück, in der die angegebenen Ordinatenwerte ausgetauscht werden.
- **ST\_Volume** - Verfügbarkeit: 2.2.0 Berechnet das Volumen eines 3D-Volumens. Bei Anwendung auf (auch geschlossene) Flächengeometrien wird 0 zurückgegeben.
- **parse\_address** - Verfügbarkeit: 2.2.0 Nimmt eine 1-zeilige Adresse entgegen und zerlegt sie in die Einzelteile
- **postgis.enable\_outdb\_rasters** - Verfügbarkeit: 2.2.0 Eine boolesche Konfigurationsmöglichkeit um den Zugriff auf out-db Rasterbänder zu ermöglichen
- **postgis.gdal\_datapath** - Verfügbarkeit: 2.2.0 Eine Konfigurationsmöglichkeit um den Wert von GDAL's GDAL\_DATA Option zu setzen. Wenn sie nicht gesetzt ist, wird die Umgebungsvariable GDAL\_DATA verwendet.
- **postgis.gdal\_enabled\_drivers** - Verfügbarkeit: 2.2.0 Eine Konfigurationsmöglichkeit um einen GDAL Treiber in der PostGIS Umgebung zu aktivieren. Beeinflusst die Konfigurationsvariable GDAL\_SKIP von GDAL.
- **standardize\_address** - Verfügbarkeit: 2.2.0 Gibt eine gegebene Adresse in der Form "stdaddr" zurück. Verwendet die Tabellen "lex", "gaz" und "rule".
- **l=|** - Verfügbarkeit: 2.2.0. Index-unterstützt steht erst ab PostgreSQL 9.5+ zur Verfügung. Gibt die Entfernung zwischen den Trajektorien A und B, am Ort der dichtesten Annäherung, an.

- **<->** - Verbesserung: 2.2.0 -- Echtes KNN ("K nearest neighbor") Verhalten für Geometrie und Geographie ab PostgreSQL 9.5+. Beachten Sie bitte, das KNN für Geographie auf der Späre und nicht auf dem Sphäroid beruht. Für PostgreSQL 9.4 und darunter, wird die Berechnung nur auf Basis des Centroids der Box unterstützt. Gibt die 2D Entfernung zwischen A und B zurück.
- **AsTopoJSON** - Erweiterung: 2.2.1 Unterstützung für punktförmige Eingabewerte hinzugefügt Gibt die TopoJSON-Darstellung einer TopoGeometry zurück.
- **ST\_Area** - Erweiterung: 2.2.0 - die Messung auf dem Referenzellipsoid wird mit der Bibliothek "GeographicLib" durchgeführt. Dadurch wurde die Genauigkeit und die Robustheit erhöht. Um die Vorteile dieser neuen Funktionalität zu nutzen, benötigen Sie Proj >= 4.9.0. Gibt den geometrischen Schwerpunkt einer Geometrie zurück.
- **ST\_AsX3D** - Erweiterung: 2.2.0: Unterstützung für geographische Koordinaten und Vertauschen der Achsen (x/y, Länge/Breite). Für nähere Details siehe Optionen. Gibt eine Geometrie im X3D XML Knotenelement-Format zurück: ISO-IEC-19776-1.2-X3DEncodings-XML
- **ST\_Azimuth** - Erweiterung: 2.2.0 die Messungen auf dem Referenzellipsoid werden mit der Bibliothek "GeographicLib" durchgeführt. Dadurch wurde die Genauigkeit und die Robustheit erhöht. Um die Vorteile dieser neuen Funktionalität zu nutzen, benötigen Sie Proj >= 4.9.0. Gibt die 2-dimensionale kürzeste Strecke zwischen zwei Geometrien als Linie zurück
- **ST\_Distance** - Erweiterung: 2.2.0 - die Messung auf dem Referenzellipsoid wird mit der Bibliothek "GeographicLib" durchgeführt. Dadurch wurde die Genauigkeit und die Robustheit erhöht. Um die Vorteile dieser neuen Funktionalität zu nutzen, benötigen Sie Proj >= 4.9.0. Gibt die größte 3-dimensionale Distanz zwischen zwei geometrischen Objekten als Linie zurück
- **ST\_Scale** - Verbessert: In Version 2.2.0 wurde die Unterstützung für die Skalierung aller Dimensionen (Parameterfactor) eingeführt. Skaliert eine Geometrie um bestimmte Faktoren.
- **ST\_Split** - Verbessert: In Version 2.2.0 wurde die Unterstützung für die Aufteilung einer Linie durch eine Mehrlinien-, eine Mehrpunkt- oder eine (Mehr-)Polygonbegrenzung eingeführt. Gibt eine Sammlung von Geometrien zurück, die durch Aufteilung einer Geometrie durch eine andere Geometrie entstanden sind.
- **ST\_Summary** - Erweiterung: 2.2.0 Unterstützung für TIN und Kurven Gibt eine Zusammenfassung des Inhalts einer Geometrie wieder.

#### Geänderte Funktionen in PostGIS 2.2

- **<->** - Änderung: 2.2.0 -- Da für Anwender von PostgreSQL 9.5 der alte hybride Syntax langsamer sein kann, möchten sie diesen Hack eventuell loswerden, falls der Code nur auf PostGIS 2.2+ 9.5+ läuft. Siehe die unteren Beispiele. Gibt die 2D Entfernung zwischen A und B zurück.
- **Get\_Geocode\_Setting** - Änderung: 2.2.0: die Standardeinstellungen befinden sich nun in der Tabelle "geocode\_settings\_default". Die vom Anwender angepassten Einstellungen - und nur diese - befinden sich in der Tabelle "geocode\_settings". Gibt die in der Tabelle "tiger.geocode\_settings" gespeicherten Einstellungen zurück.
- **ST\_3DClosestPoint** - Änderung: 2.2.0 - Wenn 2 geometrische Objekte in 2D übergeben werden, wird ein 2D-Punkt zurückgegeben (anstelle wie früher 0 für ein fehlendes Z). Im Falle von 2D und 3D, wird für fehlende Z nicht länger 0 angenommen. Gibt den 3-dimensionalen Punkt auf g1 zurück, der den kürzesten Abstand zu g2 hat. Dies ist der Anfangspunkt des kürzesten Abstands in 3D.
- **ST\_3DDistance** - Änderung: 2.2.0 - Im Falle von 2D und 3D wird für ein fehlendes Z nicht mehr 0 angenommen. Für den geometrischen Datentyp. Es wird der geringste 3-dimensionale kartesische Abstand (basierend auf dem Koordinatenreferenzsystem) zwischen zwei geometrischen Objekten in projizierten Einheiten zurückgegeben.
- **ST\_3DLongestLine** - Änderung: 2.2.0 - Wenn 2 geometrische Objekte in 2D übergeben werden, wird ein 2D-Punkt zurückgegeben (anstelle wie früher 0 für ein fehlendes Z). Im Falle von 2D und 3D, wird für fehlende Z nicht länger 0 angenommen. Gibt die größte 3-dimensionale Distanz zwischen zwei geometrischen Objekten als Linie zurück
- **ST\_3DMaxDistance** - Änderung: 2.2.0 - Im Falle von 2D und 3D wird für ein fehlendes Z nicht mehr 0 angenommen. Für den geometrischen Datentyp. Gibt die maximale 3-dimensionale kartesische Distanz (basierend auf dem Koordinatenreferenzsystem) zwischen zwei geometrischen Objekten in projizierten Einheiten zurück.

- **ST\_3DShortestLine** - Änderung: 2.2.0 - Wenn 2 geometrische Objekte in 2D übergeben werden, wird ein 2D-Punkt zurückgegeben (anstelle wie früher 0 für ein fehlendes Z). Im Falle von 2D und 3D, wird für fehlende Z nicht länger 0 angenommen. Gibt den kürzesten 3-dimensionalen Abstand zwischen zwei geometrischen Objekten als Linie zurück
- **ST\_DistanceSphere** - Änderung: 2.2.0 In Vorgängerversionen als ST\_Distance\_Sphere bezeichnet. Gibt die kürzeste Distanz zwischen zwei geometrischen Objekten zurück, die über Länge, Breite und ein bestimmtes Referenzellipsoid gegeben sind. Vorgängerversionen von PostGIS 1.5 unterstützten nur Punkte.
- **ST\_DistanceSpheroid** - Änderung: 2.2.0 In Vorgängerversionen als ST\_Distance\_Spheroid bezeichnet. Gibt die kürzeste Distanz zwischen zwei geometrischen Objekten zurück, die über Länge, Breite und ein bestimmtes Referenzellipsoid gegeben sind. Vorgängerversionen von PostGIS 1.5 unterstützten nur Punkte.
- **ST\_Equals** - Geändert: 2.2.0 Gibt auch bei ungültigen Geometrien true zurück, wenn sie binär gleich sind Prüft, ob zwei Geometrien dieselbe Menge von Punkten enthalten
- **ST\_LengthSpheroid** - Änderung: 2.2.0 In Vorgängerversionen als ST\_Length\_Spheroid bezeichnet.und mit dem Alias "ST\_3DLength\_S" versehen Gibt den geometrischen Schwerpunkt einer Geometrie zurück.
- **ST\_MemSize** - Geändert: 2.2.0 Name geändert in ST\_MemSize, um der Namenskonvention zu folgen. Gibt den Geometrietyp des ST\_Geometry Wertes zurück.
- **ST\_PointInsideCircle** - Geändert: 2.2.0 In früheren Versionen hieß dies ST\_Point\_Inside\_Circle Prüft, ob ein geometrischer Punkt innerhalb eines Kreises liegt, der durch einen Mittelpunkt und einen Radius definiert ist
- **ValidateTopology** - Änderung: 2.2.0 Bei 'edge crosses node' wurden die Werte für id1 und id2 vertauscht, um mit der Fehlerbeschreibung konsistent zu sein. Liefert eine Menge validate\_topology\_returntype Objekte, die Probleme mit der Topologie beschreiben.

### 13.12.11 PostGIS-Funktionen neu oder erweitert in 2.1

Die unten aufgeführten Funktionen sind PostGIS-Funktionen, die hinzugefügt oder erweitert wurden.

Neue Funktionen in PostGIS 2.1

- **=** - Verfügbarkeit: 2.1.0 Gibt TRUE zurück, wenn die umschreibenden Rechtecke von A und B ident sind. Das umschreibende Rechteck ist in Double Precision.
- **AsTopoJSON** - Verfügbarkeit: 2.1.0 Gibt die TopoJSON-Darstellung einer TopoGeometry zurück.
- **Drop\_Nation\_Tables\_Generate\_Script** - Verfügbarkeit: 2.1.0 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.
- **Get\_Geocode\_Setting** - Verfügbarkeit: 2.1.0 Gibt die in der Tabelle "tiger.geocode\_settings" gespeicherten Einstellungen zurück.
- **Loader\_Generate\_Nation\_Script** - Verfügbarkeit: 2.1.0 Erzeugt für die angegebene Plattform ein Shell-Skript, welches die County und State Lookup Tabellen ladet.
- **Page\_Normalize\_Address** - Verfügbarkeit: 2.1.0 Für einen gegebenen Adressentext wird der zusammengesetzte Datentyp norm\_addy zurückgegeben, der ein Suffix und ein Präfix für die Straße, einen normierten Datentyp, die Straße, den Straßennamen etc. enthält und diese einzelnen Attributen zuweist. Diese Funktion benötigt lediglich die "lookup data", die mit dem Tiger Geokodierer paketiert sind (Tiger Census Daten werden nicht benötigt). Benötigt die Erweiterung "address\_standardizer".
- **ST\_3DArea** - Verfügbarkeit: 2.1.0 Berechnet die Fläche von 3D-Oberflächengeometrien. Gibt 0 für Solids zurück.
- **ST\_3DIntersection** - Verfügbarkeit: 2.1.0 3D-Schnitte durchführen
- **ST\_Box2dFromGeoHash** - Verfügbarkeit: 2.1.0 Gibt die BOX2D einer GeoHash Zeichenkette zurück.
- **ST\_ColorMap** - Verfügbarkeit: 2.1.0 Erzeugt aus einem bestimmten Band des Ausgangsrasters einen neuen Raster mit bis zu vier 8BUI-Bändern (Grauwert, RGB, RGBA). Wenn kein Band angegeben ist, wird Band 1 angenommen.

- **ST\_Contains** - Verfügbarkeit: 2.1.0 Gibt TRUE zurück, wenn kein Punkt des Rasters "rastB" im Äußeren des Rasters "rastA" liegt und zumindest ein Punkt im Inneren von "rastB" auch im Inneren von "rastA" liegt.
  - **ST\_ContainsProperly** - Verfügbarkeit: 2.1.0 Gibt TRUE zurück, wenn "rastB" das Innere von "rastA" schneidet, aber nicht die Begrenzung oder das Äußere von "rastA".
  - **ST\_CoveredBy** - Verfügbarkeit: 2.1.0 Gibt TRUE zurück, wenn kein Punkt des Rasters "rastA" außerhalb des Rasters "rastB" liegt.
  - **ST\_Covers** - Verfügbarkeit: 2.1.0 Gibt TRUE zurück, wenn kein Punkt des Rasters "rastB" außerhalb des Rasters "rastA" liegt.
  - **ST\_DFullyWithin** - Verfügbarkeit: 2.1.0 Gibt TRUE zurück, wenn die Raster "rastA" und "rastB" zur Gänze innerhalb der angegebenen Distanz zueinander liegen.
  - **ST\_DWithin** - Verfügbarkeit: 2.1.0 Gibt TRUE zurück, wenn die Raster "rastA" und "rastB" innerhalb der angegebenen Entfernung voneinander liegen.
  - **ST\_DelaunayTriangles** - Verfügbarkeit: 2.1.0 Gibt die Delaunay-Triangulation der Scheitelpunkte einer Geometrie zurück.
  - **ST\_Disjoint** - Verfügbarkeit: 2.1.0 Gibt TRUE zurück, wenn sich die Raster "rastA" und "rastB" räumlich nicht überschneiden.
  - **ST\_DumpValues** - Verfügbarkeit: 2.1.0 Gibt die Werte eines bestimmten Bandes als 2-dimensionales Feld aus.
  - **ST\_Extrude** - Verfügbarkeit: 2.1.0 Extrudieren einer Oberfläche in ein zugehöriges Volumen
  - **ST\_ForceLHR** - Verfügbarkeit: 2.1.0 LHR-Ausrichtung erzwingen
  - **ST\_FromGDALRaster** - Verfügbarkeit: 2.1.0 Erzeugt einen Raster aus einer von GDAL unterstützten Rasterdatei.
  - **ST\_GeomFromGeoHash** - Verfügbarkeit: 2.1.0 Gibt die Geometrie einer GeoHash Zeichenfolge zurück.
  - **ST\_InvDistWeight4ma** - Verfügbarkeit: 2.1.0 Funktion zur Rasterdatenverarbeitung, die den Wert eines Pixel aus den Pixel der Nachbarschaft interpoliert.
  - **ST\_MapAlgebra (callback function version)** - Verfügbarkeit: 2.1.0 Die Version mit der Rückruffunktion - Gibt für einen oder mehrere Eingaberaster einen Raster mit einem Band, den Bandindizes und einer vom Anwender vorgegebenen Rückruffunktion zurück.
  - **ST\_MapAlgebra (expression version)** - Verfügbarkeit: 2.1.0 Version mit Ausdrücken - Gibt für einen oder zwei Ausgangsraster, Bandindizes und einer oder mehreren vom Anwender vorgegebenen SQL-Ausdrücken, einen Raster mit einem Band zurück.
  - **ST\_MinConvexHull** - Verfügbarkeit: 2.1.0 Gibt die Geometrie der konvexen Hülle des Raster aus, wobei Pixel mit NODATA ausgenommen werden.
  - **ST\_MinDist4ma** - Verfügbarkeit: 2.1.0 Funktion zur Rasterdatenverarbeitung, welche die kürzeste Entfernung (in Pixel) zwischen dem Pixel von Interesse und einem benachbarten Pixel mit Zellwert zurückgibt.
  - **ST\_MinkowskiSum** - Verfügbarkeit: 2.1.0 Führt die Minkowski-Summe aus
  - **ST\_NearestValue** - Verfügbarkeit: 2.1.0 Gibt den nächstgelegenen nicht NODATA Wert eines bestimmten Pixels aus, das über "columnx" und "rowy" oder durch eine Punktgeometrie - im gleichen Koordinatenreferenzsystem wie der Raster - ausgewählt wird.
  - **ST\_Neighborhood** - Verfügbarkeit: 2.1.0 Gibt ein 2-D Feld in "Double Precision" aus, das sich aus nicht NODATA Werten um ein bestimmtes Pixel herum zusammensetzt. Das Pixel Kann über "columnx" und "rowy" oder über eine Punktgeometrie - im gleichen Koordinatenreferenzsystem wie der Raster - ausgewählt werden.
  - **ST\_NotSameAlignmentReason** - Verfügbarkeit: 2.1.0 Gibt eine Meldung aus, die angibt ob die Raster untereinander ausgerichtet sind oder nicht und warum wenn nicht.
  - **ST\_Orientation** - Verfügbarkeit: 2.1.0 Bestimmung der Oberflächenausrichtung
  - **ST\_Overlaps** - Verfügbarkeit: 2.1.0 Gibt TRUE zurück, wenn sich die Raster "rastA" und "rastB" schneiden, aber ein Raster den anderen nicht zur Gänze enthält.
-

- **ST\_PixelAsCentroid** - Verfügbarkeit: 2.1.0 Gibt den geometrischen Schwerpunkt (Punktgeometrie) der Fläche aus, die durch das Pixel repräsentiert wird.
  - **ST\_PixelAsCentroids** - Verfügbarkeit: 2.1.0 Gibt den geometrischen Schwerpunkt (Punktgeometrie) für jedes Pixel des Rasterbandes zurück, zusammen mit dem Zellwert und den X- und Y-Rasterkoordinaten eines jeden Pixels. Die Koordinaten der Punkte entsprechen dem geometrischen Schwerpunkt der Pixel.
  - **ST\_PixelAsPoint** - Verfügbarkeit: 2.1.0 Gibt eine Punktgeometrie der oberen linken Ecke des Rasters zurück.
  - **ST\_PixelAsPoints** - Verfügbarkeit: 2.1.0 Gibt eine Punktgeometrie für jedes Pixel des Rasterbandes zurück, zusammen mit dem Zellwert und den X- und Y-Rasterkoordinaten eines jeden Pixels. Die Koordinaten der Punkte entsprechen dem oberen linken Eck der Pixel.
  - **ST\_PixelOfValue** - Verfügbarkeit: 2.1.0 Gibt die columnx- und rowy-Koordinaten jener Pixel aus, deren Zellwert gleich dem gesuchten Wert ist.
  - **ST\_PointFromGeoHash** - Verfügbarkeit: 2.1.0 Gibt einen Punkt von einer GeoHash Zeichenfolge zurück.
  - **ST\_RasterToWorldCoord** - Verfügbarkeit: 2.1.0 Gibt die obere linke Ecke des Rasters in geodätischem X und Y (Länge und Breite) für eine gegebene Spalte und Zeile aus. Spalte und Zeile wird von 1 aufwärts gezählt.
  - **ST\_Resize** - Verfügbarkeit: 2.1.0 benötigt GDAL 1.6.1+ Ändert die Zellgröße - width/height - eines Rasters
  - **ST\_Roughness** - Verfügbarkeit: 2.1.0 Gibt einen Raster mit der berechneten "Rauhigkeit" des DHM zurück.
  - **ST\_SetValues** - Verfügbarkeit: 2.1.0 Gibt einen Raster zurück, der durch das Setzen der Werte eines bestimmten Bandes verändert wurde.
  - **ST\_Simplify** - Verfügbarkeit: 2.1.0 Gibt für eine TopoGeometry eine "vereinfachte" geometrische Version zurück. Verwendet den Douglas-Peucker Algorithmus.
  - **ST\_StraightSkeleton** - Verfügbarkeit: 2.1.0 Berechnet die konvexe Hülle einer Geometrie.
  - **ST\_Summary** - Verfügbarkeit: 2.1.0 Gibt eine textliche Zusammenfassung des Rasterinhalts zurück.
  - **ST\_TPI** - Verfügbarkeit: 2.1.0 Berechnet den "Topographic Position Index" eines Raster.
  - **ST\_TRI** - Verfügbarkeit: 2.1.0 Gibt einen Raster mit errechneten Geländerauheitsindex aus.
  - **ST\_Tesselate** - Verfügbarkeit: 2.1.0 Führt eine Oberflächentesselierung eines Polygons oder einer Polyederfläche durch und gibt diese als TIN oder Sammlung von TINS zurück
  - **ST\_Tile** - Verfügbarkeit: 2.1.0 Gibt Raster, die aus einer Teilungsoperation des Eingaberasters resultieren, mit den gewünschten Dimensionen aus.
  - **ST\_Touches** - Verfügbarkeit: 2.1.0 Gibt TRUE zurück, wenn rastA und rastB zumindest einen Punkt gemeinsam haben sich aber nicht überschneiden.
  - **ST\_Union** - Verfügbarkeit: 2.1.0 ST\_Union(rast, unionarg) Variante wurde eingeführt. Gibt die Vereinigung mehrerer Rasterkacheln in einem einzelnen Raster mit mehreren Bändern zurück.
  - **ST\_Within** - Verfügbarkeit: 2.1.0 Gibt TRUE zurück, wenn kein Punkt des Rasters "rastA" außerhalb des Rasters "rastB" liegt und zumindest ein Punkt im Inneren von "rastA" auch im Inneren von "rastB" liegt.
  - **ST\_WorldToRasterCoord** - Verfügbarkeit: 2.1.0 Gibt für ein geometrisches X und Y (geographische Länge und Breite) oder für eine Punktgeometrie im Koordinatenreferenzsystem des Rasters, die obere linke Ecke als Spalte und Zeile aus.
  - **Set\_Geocode\_Setting** - Verfügbarkeit: 2.1.0 Setzt die Einstellungen, welche das Verhalten der Funktionen des Geokodierers beeinflussen.
  - **UpdateRasterSRID** - Verfügbarkeit: 2.1.0 Änderung der SRID aller Raster in der vom Anwender angegebenen Spalte und Tabelle.
  - **clearTopoGeom** - Verfügbarkeit: 2.1 Löscht den Inhalt einer TopoGeometry.
-



- **postgis.backend** - Verfügbarkeit: 2.1.0 Dieses Backend stellt eine Funktion zur Auswahl zwischen GEOS und SFCGAL zur Verfügung.
- **postgis\_sfcgal\_version** - Verfügbarkeit: 2.1.0 Gibt die verwendete Version von SFCGAL zurück

#### Erweiterte Funktionen in PostGIS 2.1

- **ST\_AddBand** - Erweiterung: 2.1.0 - Unterstützung für "addbandarg" hinzugefügt. Gibt einen Raster mit den neu hinzugefügten Band(Bändern) aus. Der Typ, der Ausgangswert und der Index für den Speicherort des Bandes kann angegeben werden. Wenn kein Index angegeben ist, wird das Band am Ende hinzugefügt.
  - **ST\_AddBand** - Erweiterung: 2.1.0 Unterstützung für die neuen "out-db" Bänder hinzugefügt. Gibt einen Raster mit den neu hinzugefügten Band(Bändern) aus. Der Typ, der Ausgangswert und der Index für den Speicherort des Bandes kann angegeben werden. Wenn kein Index angegeben ist, wird das Band am Ende hinzugefügt.
  - **ST\_AsBinary/ST\_AsWKB** - Erweiterung: 2.1.0 outasin hinzugefügt Gibt die Well-known-Binary (WKB) Darstellung eines Rasters zurück.
  - **ST\_AsGML** - Erweiterung: 2.1.0 Für GML 3 wurde die Unterstützung einer ID eingeführt. Gibt die Geometrie als GML-Element - Version 2 oder 3 - zurück.
  - **ST\_Aspect** - Erweiterung: 2.1.0 Verwendet ST\_MapAlgebra() und der optionale Funktionsparameter interpolate\_nodata wurde hinzugefügt Gibt die Exposition (standardmäßig in Grad) eines Rasterbandes mit Höhen aus. Nützlich für Terrain-Analysen.
  - **ST\_Boundary** - Erweiterung: mit 2.1.0 wurde die Unterstützung von Dreiecken eingeführt Gibt die abgeschlossene Hülle aus der kombinierten Begrenzung der Geometrie zurück.
  - **ST\_Clip** - Erweiterung: 2.1.0 neu geschrieben in C Returns the raster clipped by the input geometry. If band number is not specified, all bands are processed. If crop is not specified or TRUE, the output raster is cropped. If touched is set to TRUE, then touched pixels are included, otherwise only if the center of the pixel is in the geometry it is included.
  - **ST\_DWithin** - Verbessert: 2.1.0 verbesserte die Geschwindigkeit für Geographie. Siehe Geografie schneller machen für Details. Prüft, ob zwei Geometrien innerhalb eines bestimmten Abstands liegen
  - **ST\_DWithin** - Verbessert: 2.1.0 Unterstützung für gekrümmte Geometrien wurde eingeführt. Prüft, ob zwei Geometrien innerhalb eines bestimmten Abstands liegen
  - **ST\_Distance** - Enhanced: 2.1.0 Geschwindigkeitsverbesserung beim geographischen Datentyp. Siehe Making Geography faster für Details. Gibt die größte 3-dimensionale Distanz zwischen zwei geometrischen Objekten als Linie zurück
  - **ST\_Distance** - Erweiterung: 2.1.0 - Unterstützung für Kurven beim geometrischen Datentyp eingeführt. Gibt die größte 3-dimensionale Distanz zwischen zwei geometrischen Objekten als Linie zurück
  - **ST\_Distinct4ma** - Erweiterung: 2.1.0 Variante 2 wurde hinzugefügt Funktion zur Rasterdatenverarbeitung, welche die Anzahl der einzelnen Pixelwerte in der Nachbarschaft errechnet.
  - **ST\_DumpPoints** - Verbessert: 2.1.0 Höhere Geschwindigkeit. Reimplementiert als natives C. Gibt eine Zusammenfassung des Inhalts einer Geometrie wieder.
  - **ST\_HillShade** - Erweiterung: 2.1.0 Verwendet ST\_MapAlgebra() und der optionale Funktionsparameter interpolate\_nodata wurde hinzugefügt Gibt für gegebenen Horizontalwinkel, Höhenwinkel, Helligkeit und Maßstabsverhältnis die hypothetische Beleuchtung eines Höhenrasterbandes zurück.
  - **ST\_MakeValid** - Verbessert: 2.1.0, Unterstützung für GEOMETRYCOLLECTION und MULTIPOINT hinzugefügt. Versucht, eine ungültige Geometrie gültig zu machen, ohne dass Scheitelpunkte verloren gehen.
  - **ST\_Max4ma** - Erweiterung: 2.1.0 Variante 2 wurde hinzugefügt Funktion zur Rasterdatenverarbeitung, die den maximalen Zellwert in der Nachbarschaft eines Pixel errechnet.
  - **ST\_Mean4ma** - Erweiterung: 2.1.0 Variante 2 wurde hinzugefügt Funktion zur Rasterdatenverarbeitung, die den mittleren Zellwert in der Nachbarschaft von Pixel errechnet.
-

- **ST\_Min4ma** - Erweiterung: 2.1.0 Variante 2 wurde hinzugefügt Funktion zur Rasterdatenverarbeitung, die den minimalen Zellwert in der Nachbarschaft von Pixel errechnet.
- **ST\_PixelAsPolygons** - Erweiterung: 2.1.0 Der optionale Übergabewert "exclude\_nodata\_value" wurde hinzugefügt. Gibt die umhüllende Polygoneometrie, den Zellwert, sowie die X- und Y-Rasterkoordinate für jedes Pixel aus.
- **ST\_Polygon** - Erweiterung: 2.1.0 Geschwindigkeit verbessert (zur Gänze C-basiert) und es wird sichergestellt, dass das zurück-gegebenen Mehrfachpolygon valide ist. Gibt eine Geometrie mit Mehrfachpolygone zurück, die aus der Vereinigung von Pixel mit demselben Zellwert gebildet werden. Pixel mit NODATA Werten werden nicht berücksichtigt. Wenn keine Band angegeben ist, wird die Bandnummer standardmäßig auf 1 gesetzt.
- **ST\_Range4ma** - Erweiterung: 2.1.0 Variante 2 wurde hinzugefügt Funktion zur Rasterdatenverarbeitung, die den Wertebereich der Pixel in einer Nachbarschaft errechnet.
- **ST\_SameAlignment** - Erweiterung: 2.1.0 die Variante mit der Aggregatfunktion hinzugefügt Gibt TRUE zurück, wenn die Raster die selbe Rotation, Skalierung, Koordinatenreferenzsystem und Versatz (Pixel können auf dasselbe Gitter gelegt werden, ohne dass die Gitterlinien durch die Pixel schneiden) aufweisen. Wenn nicht, wird FALSE und eine Beschreibung des Problems ausgegeben.
- **ST\_Segmentize** - Erweiterung: mit 2.1.0 wurde die Unterstützung des geographischen Datentyps eingeführt. Gibt eine geänderte Geometrie/Geografie zurück, bei der kein Segment länger als eine bestimmte Entfernung ist.
- **ST\_SetGeoReference** - Erweiterung: 2.1.0 ST\_SetGeoReference(raster, double precision, ...) Variante hinzugefügt Georeferenziert einen Raster über 6 Parameter in einem einzigen Aufruf. Die Zahlen müssen durch Leerzeichen getrennt sein. Die Funktion akzeptiert die Eingabe im Format von 'GDAL' und von 'ESRI'. Der Standardwert ist GDAL.
- **ST\_SetValue** - Erweiterung: 2.1.0 Die geometrische Variante von ST\_SetValue() unterstützt nun jeden geometrischen Datentyp, nicht nur POINT. Die geometrische Variante ist ein Adapter für die geomval[] Variante von ST\_SetValues() Setzt den Wert für ein Pixel eines Bandes, das über columnx und rowy festgelegt wird, oder für die Pixel die eine bestimmte Geometrie schneiden, und gibt den veränderten Raster zurück. Die Bandnummerierung beginnt mit 1; wenn die Bandnummer nicht angegeben ist, wird 1 angenommen.
- **ST\_Slope** - Erweiterung: 2.1.0 Verwendet ST\_MapAlgebra() und es wurden die optionalen Funktionsparameter units, scale und interpolate\_nodata hinzugefügt Gibt die Neigung (standardmäßig in Grad) eines Höhenrasterbandes zurück. Nützlich für Terrain-Analysen.
- **ST\_StdDev4ma** - Erweiterung: 2.1.0 Variante 2 wurde hinzugefügt Funktion zur Rasterdatenverarbeitung, welche die Standardabweichung der Zellwerte in der Nachbarschaft von Pixel errechnet.
- **ST\_Sum4ma** - Erweiterung: 2.1.0 Variante 2 wurde hinzugefügt Funktion zur Rasterdatenverarbeitung, die die Summe aller Zellwerte in der Nachbarschaft von Pixel errechnet.
- **ST\_Summary** - Erweiterung: 2.1.0 S-Flag, diese zeigt an ob das Koordinatenreferenzsystem bekannt ist Gibt eine Zusammenfassung des Inhalts einer Geometrie wieder.
- **ST\_Transform** - Erweiterung: 2.1.0 Variante ST\_Transform(rast, alignto) hinzugefügt Projiziert einen Raster von einem bekannten Koordinatenreferenzsystem in ein anderes bekanntes Koordinatenreferenzsystem um. Die Optionen für die Skalierung sind NearestNeighbor, Bilinear, Cubisch, CubicSpline und der Lanczos-Filter, die Standardeinstellung ist NearestNeighbor.
- **ST\_Union** - Erweiterung: 2.1.0 Geschwindigkeit verbessert (zur Gänze C-basiert) Gibt die Vereinigung mehrerer Rasterkacheln in einem einzelnen Raster mit mehreren Bändern zurück.
- **ST\_Union** - Erweiterung: 2.1.0 ST\_Union(rast) (Variante 1) vereinigt alle Bänder aller Ausgangsraster. Vorherige Versionen von PostGIS setzten das erste Band voraus. Gibt die Vereinigung mehrerer Rasterkacheln in einem einzelnen Raster mit mehreren Bändern zurück.
- **ST\_Union** - Erweiterung: 2.1.0 ST\_Union(rast, uniontype) (Variante 4) vereinigt alle Bänder aller Ausgangsraster. Gibt die Vereinigung mehrerer Rasterkacheln in einem einzelnen Raster mit mehreren Bändern zurück.
- **toTopoGeom** - Erweiterung: 2.1.0 die Version, welche eine bestehende TopoGeometry entgegennimmt, wurde hinzugefügt. Wandelt eine einfache Geometrie in eine TopoGeometry um.

- **ST\_Aspect** - Änderung: 2.1.0 In Vorgängerversionen wurden die Werte in Radiant ausgegeben. Nun werden die Werte standardmäßig in Grad ausgegeben. Gibt die Exposition (standardmäßig in Grad) eines Rasterbandes mit Höhen aus. Nützlich für Terrain-Analysen.
- **ST\_EstimatedExtent** - Geändert: 2.1.0. Bis zu 2.0.x hieß dies ST\_Estimated\_Extent. Gibt die geschätzte Ausdehnung einer räumlichen Tabelle zurück.
- **ST\_Force2D** - Änderung: 2.1.0. Bis zu 2.0.x wurde diese Funktion mit ST\_Force\_2D bezeichnet. Die Geometrien in einen "2-dimensionalen Modus" zwingen.
- **ST\_Force3D** - Änderung: 2.1.0. Bis zu 2.0.x wurde diese Funktion mit ST\_Force\_3D bezeichnet. Zwingt die Geometrien in einen XYZ Modus. Dies ist ein Alias für ST\_Force3DZ.
- **ST\_Force3DM** - Änderung: 2.1.0. Bis zu 2.0.x wurde diese Funktion mit ST\_Force\_3DM bezeichnet. Zwingt die Geometrien in einen XYM Modus.
- **ST\_Force3DZ** - Änderung: 2.1.0. Bis zu 2.0.x wurde diese Funktion mit ST\_Force\_3DZ bezeichnet. Zwingt die Geometrien in einen XYZ Modus.
- **ST\_Force4D** - Änderung: 2.1.0. Bis zu 2.0.x wurde diese Funktion mit ST\_Force\_4D bezeichnet. Zwingt die Geometrien in einen XYZM Modus.
- **ST\_ForceCollection** - Änderung: 2.1.0. Bis zu 2.0.x wurde diese Funktion mit ST\_Force\_Collection bezeichnet. Wandelt eine Geometrie in eine GEOMETRYCOLLECTION um.
- **ST\_HillShade** - Änderung: 2.1.0 In Vorgängerversionen wurden Richtungswinkel und Höhenwinkel in Radiant angegeben. Nun werden die Werte in Grad ausgedrückt. Gibt für gegebenen Horizontalwinkel, Höhenwinkel, Helligkeit und Maßstabsverhältnis die hypothetische Beleuchtung eines Höhenrasterbandes zurück.
- **ST\_LineInterpolatePoint** - Änderung: 2.1.0. Bis zu 2.0.x wurde diese Funktion mit ST\_Line\_Interpolate\_Point bezeichnet. Gibt einen oder mehrere, entlang einer Linie interpolierte Punkte zurück.
- **ST\_LineLocatePoint** - Änderung: 2.1.0. Bis zu 2.0.x wurde diese Funktion mit ST\_Line\_Locate\_Point bezeichnet. Liefert die gebrochene Position des Punktes auf einer Linie, der einem Punkt am nächsten liegt.
- **ST\_LineSubstring** - Änderung: 2.1.0. Bis zu 2.0.x wurde diese Funktion mit ST\_Line\_Substring bezeichnet. Gibt den Teil einer Linie zwischen zwei gebrochenen Stellen zurück.
- **ST\_PixelAsCentroids** - Änderung: 2.1.1 Die Verhaltensweise von "exclude\_nodata\_value" wurde geändert. Gibt den geometrischen Schwerpunkt (Punktgeometrie) für jedes Pixel des Rasterbandes zurück, zusammen mit dem Zellwert und den X- und Y-Rasterkoordinaten eines jeden Pixels. Die Koordinaten der Punkte entsprechen dem geometrischen Schwerpunkt der Pixel.
- **ST\_PixelAsPoints** - Änderung: 2.1.1 Die Verhaltensweise von "exclude\_nodata\_value" wurde geändert. Gibt eine Punktgeometrie für jedes Pixel des Rasterbandes zurück, zusammen mit dem Zellwert und den X- und Y-Rasterkoordinaten eines jeden Pixels. Die Koordinaten der Punkte entsprechen dem oberen linken Eck der Pixel.
- **ST\_PixelAsPolygons** - Änderung: 2.1.1 Die Verhaltensweise von "exclude\_nodata\_value" wurde geändert. Gibt die umhüllende Polygoneometrie, den Zellwert, sowie die X- und Y-Rasterkoordinate für jedes Pixel aus.
- **ST\_Polygon** - Änderung: 2.1.0 In Vorgängerversionen wurde manchmal ein Polygon zurückgegeben; dies wurde geändert so dass jetzt immer ein Mehrfachpolygon zurückgegeben wird. Gibt eine Geometrie mit Mehrfachpolygonen zurück, die aus der Vereinigung von Pixel mit demselben Zellwert gebildet werden. Pixel mit NODATA Werten werden nicht berücksichtigt. Wenn keine Band angegeben ist, wird die Bandnummer standardmäßig auf 1 gesetzt.
- **ST\_RasterToWorldCoordX** - Änderung: 2.1.0 Vorgängerversionen haben dies als "ST\_Raster2WorldCoordX" bezeichnet. Gibt die geodätische X Koordinate links oberhalb des Rasters, der Spalte und der Zeile aus. Die Nummerierung der Spalten und Zeilen beginnt mit 1.
- **ST\_RasterToWorldCoordY** - Änderung: 2.1.0 In Vorgängerversionen wurde dies als ST\_Raster2WorldCoordY bezeichnet. Gibt die geodätische Y Koordinate links oberhalb des Rasters, der Spalte und der Zeile aus. Die Nummerierung der Spalten und Zeilen beginnt mit 1.



- **ST\_Rescale** - Änderung: 2.1.0 Funktioniert jetzt auch mit Raster ohne SRID Neuabtastung eines Rasters, indem nur die Skala (oder Pixelgröße) angepasst wird. Die neuen Pixelwerte werden mit den Algorithmen NearestNeighbor (englische oder amerikanische Schreibweise), Bilinear, Cubic, CubicSpline, Lanczos, Max oder Min resampling berechnet. Die Voreinstellung ist NearestNeighbor.
- **ST\_Reskew** - Änderung: 2.1.0 Funktioniert jetzt auch mit Raster ohne SRID Skaliert einen Raster, indem lediglich der Versatz (oder Rotationsparameter) angepasst wird. Neue Pixelwerte werden über NearestNeighbor, bilinear, kubisch, CubicSpline oder mit dem Lanczos-Filter errechnet. Die Standardeinstellung ist NearestNeighbor.
- **ST\_Segmentize** - Geändert: 2.1.0 Infolge der Einführung der Geographie-Unterstützung verursacht die Verwendung ST\_Segmentize('1 2, 3 4)', 0.5) einen mehrdeutigen Funktionsfehler. Die Eingabe muss korrekt als Geometrie oder Geografie eingegeben werden. Verwenden Sie ST\_GeomFromText, ST\_GeogFromText oder eine Umwandlung in den erforderlichen Typ (z.B. ST\_Segmentize('LINESTRING(1 2, 3 4)::geometry, 0.5) ) Gibt eine geänderte Geometrie/Geografie zurück, bei der kein Segment länger als eine bestimmte Entfernung ist.
- **ST\_Slope** - Änderung: 2.1.0 In Vorgängerversionen wurden die Werte in Radiant ausgegeben. Nun werden die Werte standardmäßig in Grad ausgegeben. Gibt die Neigung (standardmäßig in Grad) eines Höhenrasterbandes zurück. Nützlich für Terrain-Analysen.
- **ST\_SnapToGrid** - Änderung: 2.1.0 Funktioniert jetzt auch mit Raster ohne SRID Skaliert einen Raster durch Fangen an einem Führungsgitter. Neue Pixelwerte werden über NearestNeighbor, bilinear, kubisch, CubicSpline oder mit dem Lanczos-Filter errechnet. Die Standardeinstellung ist NearestNeighbor.
- **ST\_WorldToRasterCoordX** - Änderung: 2.1.0 In Vorgängerversionen wurde dies als ST\_World2RasterCoordX bezeichnet Gibt für eine Punktgeometrie (pt) oder eine globale X- und Y-Koordinate (xw, yw) die Rasterpalette im globalen Koordinatenreferenzsystem des Rasters aus.
- **ST\_WorldToRasterCoordY** - Änderung: 2.1.0 In Vorgängerversionen wurde dies als ST\_World2RasterCoordY bezeichnet Gibt für eine Punktgeometrie (pt) oder eine globale X- und Y-Koordinate (xw, yw) die Rasterzeile im globalen Koordinatenreferenzsystem des Rasters aus.

### 13.12.12 PostGIS-Funktionen neu oder erweitert in 2.0

Die unten aufgeführten Funktionen sind PostGIS-Funktionen, die hinzugefügt oder erweitert wurden.

Neue Funktionen in PostGIS 2.0

- **&&** - Verfügbarkeit: 2.0.0 Gibt TRUE zurück, wenn das umschreibende Rechteck von A das umschreibende Rechteck von B schneidet.
- **&&&** - Verfügbarkeit: 2.0.0 Gibt TRUE zurück, wenn A's n-D bounding box B's n-D bounding box schneidet.
- **<#>** - Verfügbarkeit: 2.0.0 -- KNN steht erst ab PostgreSQL 9.1+ zur Verfügung Gibt die 2D Entfernung zwischen den Bounding Boxes von A und B zurück
- **<->** - Verfügbarkeit: 2.0.0 -- Weak KNN liefert nearest neighbors, welche sich auf die Entfernung der Centroide der Geometrien, anstatt auf den tatsächlichen Entfernungen, stützen. Genaue Ergebnisse für Punkte, ungenau für alle anderen Geometrietypen. Verfügbar ab PostgreSQL 9.1+. Gibt die 2D Entfernung zwischen A und B zurück.
- **@** - Verfügbarkeit: 2.0.0 raster @ raster, und raster @ geometry eingeführt Gibt TRUE zurück, wenn das umschreibende Rechteck von A in jenem von B enthalten ist. Das umschreibende Rechteck ist in Double Precision.
- **@** - Verfügbarkeit: 2.0.5 "geometry @ raster" eingeführt Gibt TRUE zurück, wenn das umschreibende Rechteck von A in jenem von B enthalten ist. Das umschreibende Rechteck ist in Double Precision.
- **AddEdge** - Verfügbarkeit: 2.0.0 Fügt die Kante eines Linienzugs in der Tabelle "edge", und die zugehörigen Anfangs- und Endpunkte in die Knotenpunktabelle, des jeweiligen topologischen Schemas ein. Dabei wird die übergebene Linienzuggeometrie verwendet und die edgeid der neuen (oder bestehenden) Kante ausgegeben.
- **AddFace** - Verfügbarkeit: 2.0.0 Registriert die Elementarstruktur einer Masche in einer Topologie und gibt den Identifikator der Masche aus.

- **AddNode** - Verfügbarkeit: 2.0.0 Fügt einen Knotenpunkt zu der Tabelle "node" in dem vorgegebenen topologischen Schema hinzu und gibt die "nodeid" des neuen Knotens aus. Falls der Punkt bereits als Knoten existiert, wird die vorhandene nodeid zurückgegeben.
  - **AddOverviewConstraints** - Verfügbarkeit: 2.0.0 Eine Rasterspalte als Übersicht für eine andere Rasterspalte kennzeichnen.
  - **AddRasterConstraints** - Verfügbarkeit: 2.0.0 Fügt die Raster-Constraints zu einer bestimmten Spalte einer bereits geladenen Rastertabelle hinzu. Diese Constraints beschränken das Koordinatentransformationssystem, den Maßstab, die Blockgröße, die Ausrichtung, die Bänder, den Bandtyp und eine Flag, die anzeigt ob die Rasterspalte regelmäßig geblockt ist. Es müssen bereits Daten in die Tabelle geladen sein, damit die Constraints abgeleitet werden können. Gibt TRUE zurück, wenn das Setzen der Constraints ausgeführt wurde; bei Problemen wird eine Meldung angezeigt.
  - **AsGML** - Verfügbarkeit: 2.0.0 Gibt die GML-Darstellung einer TopoGeometry zurück.
  - **CopyTopology** - Verfügbarkeit: 2.0.0 Erstellt eine Kopie einer Topologie (Knoten, Kanten, Flächen, Ebenen und TopoGeometrien) in ein neues Schema
  - **DropOverviewConstraints** - Verfügbarkeit: 2.0.0 Löscht die Markierung einer Rasterspalte, die festlegt dass sie als Übersicht für eine andere Spalte dient.
  - **DropRasterConstraints** - Verfügbarkeit: 2.0.0 Löscht die Constraints eines PostGIS Rasters die sich auf eine Rastertabellenspalte beziehen. Nützlich um Daten erneut zu laden oder um die Daten einer Rasterspalte zu aktualisieren.
  - **Drop\_Indexes\_Generate\_Script** - Verfügbarkeit: 2.0.0 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.
  - **Drop\_State\_Tables\_Generate\_Script** - Verfügbarkeit: 2.0.0 Erzeugt ein Skript, dass alle Tabellen in dem angegebenen Schema löscht, die als Präfix einen Ländercode haben. Wenn kein Schema angegeben ist wird standardmäßig auf das tiger\_data Schema zugegriffen.
  - **Geocode\_Intersection** - Verfügbarkeit: 2.0.0 Nimmt 2 sich kreuzende Straßen, einen Bundesstaat, eine Stadt und einen ZIP-Code entgegen und gibt die möglichen Punktlagen an der ersten Querstraße an der Kreuzung zurück. Die Ausgabe beinhaltet auch die Geometrie "geomout" in NAD 83 Länge/Breite, eine standardisierte Adresse normalized\_address (addy) für jede Punktlage, sowie die Rangfolge. Umso niedriger die Rangfolge ist, um so wahrscheinlicher ist die Übereinstimmung. Die Ergebnisse werden mit aufsteigender Rangfolge sortiert - dar niedrigste Rang zuerst. Optional kann die maximale Anzahl der Ergebnisse angegeben werden (Standardeinstellung ist 10). Verwendet TIGER Daten (Kanten, Maschen, Adressen) und Fuzzy String Matching (soundex, levenshtein) von PostgreSQL.
  - **GetEdgeByPoint** - Verfügbarkeit: 2.0.0 Findet die edge-id einer Kante die einen gegebenen Punkt schneidet.
  - **GetFaceByPoint** - Verfügbarkeit: 2.0.0 Findet eine Fläche, die einen bestimmten Punkt schneidet.
  - **GetNodeByPoint** - Verfügbarkeit: 2.0.0 Findet zu der Lage eines Punktes die node-id eines Knotens.
  - **GetNodeEdges** - Verfügbarkeit: 2.0 Gibt für einen Knoten die sortierte Menge der einfallenden Kanten aus.
  - **GetRingEdges** - Verfügbarkeit: 2.0.0 Gibt eine sortierte Liste von mit Vorzeichen versehenen Identifikatoren der Kanten zurück, die angetroffen werden, wenn man an der Seite der gegebenen Kante entlangwandert.
  - **GetTopoGeomElements** - Verfügbarkeit: 2.0.0 Gibt für eine TopoGeometry (Elementarstrukturen) einen Satz an topoelement Objekten zurück, welche die topologische element\_id und den element\_type beinhalten.
  - **GetTopologySRID** - Verfügbarkeit: 2.0.0 Gibt für den Namen einer Topologie, die SRID der Topologie in der Tabelle "topology.topology" aus.
  - **Get\_Tract** - Verfügbarkeit: 2.0.0 Gibt für die Lage einer Geometrie die Census Area oder ein Feld der tract-Tabelle zurück. Standardmäßig wird die Kurzbezeichnung der Census Area ausgegeben.
  - **Install\_Missing\_Indexes** - Verfügbarkeit: 2.0.0 Findet alle Tabellen mit Spalten, die für JOINS und Filterbedingungen vom Geokodierer verwendet werden und keinen Index aufweisen; die fehlenden Indizes werden hinzugefügt.
-

- **Loader\_Generate\_Census\_Script** - Verfügbarkeit: 2.0.0 Erzeugt für gegebene Plattform und Bundesstaaten ein Shellskript, das die TIGER Datentabellen "tract", "bg" und "tabblocks" herunterlädt, bereitstellt und in das Schema tiger\_data importiert. Jedes Bundesstaat-Skript wird in einem eigenen Datensatz ausgegeben.
  - **Loader\_Generate\_Script** - Verfügbarkeit: 2.0.0 unterstützt die strukturierten Daten von Tiger 2010 und ladet die Tabellen "tract" (Census Area), "bg" (Census Block Groups) und "tabblocks" (Census Blocks). Erzeugt für gegebene Plattform und Bundesstaaten ein Shellskript, das die TIGER Daten herunterlädt, bereitstellt und in das Schema tiger\_data importiert. Jedes Bundesstaat-Skript wird in einem eigenen Datensatz ausgegeben. Die neueste Version unterstützt die geänderte Struktur von Tiger 2010 und lädt ebenfalls die Census Tract, Block Groups und Blocks Tabellen.
  - **Missing\_Indexes\_Generate\_Script** - Verfügbarkeit: 2.0.0 Findet alle Tabellen mit Schlüsselspalten, die für JOINS vom Geokodierer verwendet werden und keinen Index aufweisen; gibt ein DDL (SQL) aus, das die Indizes für diese Tabellen festlegt.
  - **Polygonize** - Verfügbarkeit: 2.0.0 Findet und registriert alle Maschen, die durch die Kanten der Topologie festgelegt sind.
  - **Reverse\_Geocode** - Verfügbarkeit: 2.0.0 Nimmt einen geometrischen Punkt in einem bekannten Koordinatenreferenzsystem entgegen und gibt einen Datensatz zurück, das ein Feld mit theoretisch möglichen Adressen und ein Feld mit Straßenkreuzungen beinhaltet. Wenn include\_strnum\_range = true, dann beinhalten die Straßenkreuzungen den "Street Range" (Kennung des Straßenabschnitts).
  - **ST\_3DClosestPoint** - Verfügbarkeit: 2.0.0 Gibt den 3-dimensionalen Punkt auf g1 zurück, der den kürzesten Abstand zu g2 hat. Dies ist der Anfangspunkt des kürzesten Abstands in 3D.
  - **ST\_3DDFullyWithin** - Verfügbarkeit: 2.0.0 Prüft, ob zwei 3D-Geometrien vollständig innerhalb eines bestimmten 3D-Abstands liegen
  - **ST\_3DDWithin** - Verfügbarkeit: 2.0.0 Prüft, ob zwei 3D-Geometrien innerhalb eines bestimmten 3D-Abstands liegen
  - **ST\_3DDistance** - Verfügbarkeit: 2.0.0 Für den geometrischen Datentyp. Es wird der geringste 3-dimensionale kartesische Abstand (basierend auf dem Koordinatenreferenzsystem) zwischen zwei geometrischen Objekten in projizierten Einheiten zurückgegeben.
  - **ST\_3DIntersects** - Verfügbarkeit: 2.0.0 Prüft, ob sich zwei Geometrien in 3D räumlich schneiden - nur für Punkte, Linienzüge, Polygone, polyedrische Flächen (Bereich)
  - **ST\_3DLongestLine** - Verfügbarkeit: 2.0.0 Gibt die größte 3-dimensionale Distanz zwischen zwei geometrischen Objekten als Linie zurück
  - **ST\_3DMaxDistance** - Verfügbarkeit: 2.0.0 Für den geometrischen Datentyp. Gibt die maximale 3-dimensionale kartesische Distanz (basierend auf dem Koordinatenreferenzsystem) zwischen zwei geometrischen Objekten in projizierten Einheiten zurück.
  - **ST\_3DShortestLine** - Verfügbarkeit: 2.0.0 Gibt den kürzesten 3-dimensionalen Abstand zwischen zwei geometrischen Objekten als Linie zurück
  - **ST\_AddEdgeModFace** - Verfügbarkeit: 2.0 Fügt eine Kante hinzu. Falls dabei eine Masche aufgetrennt wird, so wird die ursprüngliche Masche angepasst und eine weitere Masche hinzugefügt.
  - **ST\_AddEdgeNewFaces** - Verfügbarkeit: 2.0 Fügt eine Kante hinzu. Falls dabei eine Masche aufgetrennt wird, so wird die ursprüngliche Masche gelöscht und durch zwei neue Maschen ersetzt.
  - **ST\_AsGDALRaster** - Verfügbarkeit: 2.0.0 - GDAL >= 1.6.0. Gibt die Rasterkachel in dem ausgewiesenen Rasterformat von GDAL aus. Sie können jedes Rasterformat angeben, das von Ihrer Bibliothek unterstützt wird. Um eine Liste mit den unterstützten Formaten auszugeben, verwenden Sie bitte ST\_GDALDrivers().
  - **ST\_AsJPEG** - Verfügbarkeit: 2.0.0 - GDAL >= 1.6.0. Gibt die ausgewählten Bänder der Rasterkachel als einzelnes Bild (Byte-Array) im Format "Joint Photographic Exports Group" (JPEG) aus. Wenn kein Band angegeben ist und 1 oder mehr als 3 Bänder ausgewählt wurden, dann wird nur das erste Band verwendet. Wenn 3 Bänder ausgewählt wurden, werden alle 3 Bänder verwendet und auf RGB abgebildet.
  - **ST\_AsLatLonText** - Verfügbarkeit: 2.0 Gibt die "Grad, Minuten, Sekunden"-Darstellung für den angegebenen Punkt aus.
-

- **ST\_AsPNG** - Verfügbarkeit: 2.0.0 - GDAL  $\geq$  1.6.0. Gibt die ausgewählten Bänder der Rasterkachel als einzelnes, übertragbares Netzwerkgraphik (PNG) Bild (Byte-Feld) aus. Wenn der Raster 1,3 oder 4 Bänder hat und keine Bänder angegeben sind, dann werden alle Bänder verwendet. Wenn der Raster 2 oder mehr als 4 Bänder hat und keine Bänder angegeben sind, dann wird nur Band 1 verwendet. Die Bänder werden in den RGB- oder den RGBA-Raum abgebildet.
  - **ST\_AsRaster** - Verfügbarkeit: 2.0.0 - GDAL  $\geq$  1.6.0. Konvertiert den geometrischen Datentyp von PostGIS in einen PostGIS Raster.
  - **ST\_AsTIFF** - Verfügbarkeit: 2.0.0 - GDAL  $\geq$  1.6.0. Gibt die ausgewählten Bänder des Raster als einzelnes TIFF Bild (Byte-Feld) zurück. Wenn kein Band angegeben ist oder keines der angegebenen Bänder im Raster existiert, werden alle Bänder verwendet.
  - **ST\_AsX3D** - Verfügbarkeit: 2.0.0: ISO-IEC-19776-1.2-X3DEncodings-XML Gibt eine Geometrie im X3D XML Knotenelement-Format zurück: ISO-IEC-19776-1.2-X3DEncodings-XML
  - **ST\_Aspect** - Verfügbarkeit: 2.0.0 Gibt die Exposition (standardmäßig in Grad) eines Rasterbandes mit Höhen aus. Nützlich für Terrain-Analysen.
  - **ST\_Band** - Verfügbarkeit: 2.0.0 Gibt einen oder mehrere Bänder eines bestehenden Rasters als neuen Raster aus. Nützlich um neue Raster aus bestehenden Rastern abzuleiten.
  - **ST\_BandIsNoData** - Verfügbarkeit: 2.0.0 Gibt TRUE aus, wenn das Band ausschließlich aus NODATA Werten besteht.
  - **ST\_Clip** - Verfügbarkeit: 2.0.0 Returns the raster clipped by the input geometry. If band number is not specified, all bands are processed. If crop is not specified or TRUE, the output raster is cropped. If touched is set to TRUE, then touched pixels are included, otherwise only if the center of the pixel is in the geometry it is included.
  - **ST\_CollectionHomogenize** - Verfügbarkeit: 2.0.0 Gibt die einfachste Darstellung einer Geometriesammlung zurück.
  - **ST\_ConcaveHull** - Verfügbarkeit: 2.0.0 Berechnet eine möglicherweise konkave Geometrie, die alle Eckpunkte der Eingabegeometrie enthält
  - **ST\_Count** - Verfügbarkeit: 2.0.0 Gibt die Anzahl der Pixel für ein Band eines Rasters oder eines Raster-Coverage zurück. Wenn kein Band angegeben ist, wird standardmäßig Band 1 gewählt. Wenn der Parameter "exclude\_nodata\_value" auf TRUE gesetzt ist, werden nur Pixel mit Werten ungleich NODATA gezählt.
  - **ST\_CreateTopoGeo** - Verfügbarkeit: 2.0 Fügt eine Sammlung von Geometrien an eine leere Topologie an und gibt eine Bestätigungsmeldung aus.
  - **ST\_Distinct4ma** - Verfügbarkeit: 2.0.0 Funktion zur Rasterdatenverarbeitung, welche die Anzahl der einzelnen Pixelwerte in der Nachbarschaft errechnet.
  - **ST\_FlipCoordinates** - Verfügbarkeit: 2.0.0 Gibt eine Version einer Geometrie mit gespiegelter X- und Y-Achse zurück.
  - **ST\_GDALDrivers** - Verfügbarkeit: 2.0.0 - GDAL  $\geq$  1.6.0. Gibt eine Liste der Rasterformate aus, die von PostGIS über die Bibliothek GDAL unterstützt werden. Nur die Formate mit can\_write=True können von ST\_AsGDALRaster verwendet werden.
  - **ST\_GeomFromGeoJSON** - Verfügbarkeit: 2.0.0 benötigt - JSON-C  $\geq$  0.9 Nimmt als Eingabe eine GeoJSON-Darstellung der Geometrie und gibt ein geometrisches PostGIS-Objekt aus.
  - **ST\_GetFaceEdges** - Verfügbarkeit: 2.0 Gibt die Kanten, die aface begrenzen, sortiert aus.
  - **ST\_HasNoBand** - Verfügbarkeit: 2.0.0 Gibt TRUE aus, wenn kein Band mit der angegebenen Bandnummer existiert. Gibt den Pixeltyp des angegebenen Bandes aus. Wenn keine Bandnummer angegeben ist, wird das 1ste Band angenommen.
  - **ST\_HillShade** - Verfügbarkeit: 2.0.0 Gibt für gegebenen Horizontalwinkel, Höhenwinkel, Helligkeit und Maßstabsverhältnis die hypothetische Beleuchtung eines Höhenrasterbandes zurück.
  - **ST\_Histogram** - Verfügbarkeit: 2.0.0 Gibt Datensätze aus, welche die Verteilung der Daten eines Rasters oder eines Rastercoverage darstellen. Dabei wird die Wertemenge in Klassen aufgeteilt und für jede Klasse zusammengefasst. Wenn die Anzahl der Klassen nicht angegeben ist, wird sie automatisch berechnet.
-

- **ST\_InterpolatePoint** - Verfügbarkeit: 2.0.0 Für einen gegebenen Punkt wird die Kilometrierung auf dem nächstliegenden Punkt einer Geometrie zurück.
  - **ST\_IsEmpty** - Verfügbarkeit: 2.0.0 Gibt TRUE zurück, wenn der Raster leer ist (width = 0 and height = 0). Andernfalls wird FALSE zurückgegeben.
  - **ST\_IsValidDetail** - Verfügbarkeit: 2.0.0 Gibt eine Zeile valid\_detail zurück, die angibt, ob eine Geometrie gültig ist oder, falls nicht, einen Grund und einen Ort.
  - **ST\_IsValidReason** - Verfügbarkeit: Version 2.0 mit Flaggen. Gibt einen Text zurück, der angibt, ob eine Geometrie gültig ist, oder einen Grund für die Ungültigkeit.
  - **ST\_MakeLine** - Verfügbarkeit: 2.0.0 - Unterstützung zur Eingabe von LineString Elementen eingeführt Erzeugt einen Linienzug aus einer Punkt-, Mehrfachpunkt- oder Liniengeometrie.
  - **ST\_MakeValid** - Verfügbarkeit: 2.0.0 Versucht, eine ungültige Geometrie gültig zu machen, ohne dass Scheitelpunkte verloren gehen.
  - **ST\_MapAlgebraExpr** - Verfügbarkeit: 2.0.0 Version mit 1 Rasterband: Erzeugt ein neues Rasterband, dass über eine gültige, algebraische PostgreSQL Operation für ein Rasterband mit gegebenen Pixeltyp erstellt wird. Wenn kein Band bestimmt ist, wird Band 1 angenommen.
  - **ST\_MapAlgebraExpr** - Verfügbarkeit: 2.0.0 Version mit 2 Rasterbändern: Erstellt einen neuen Einzelbandraster, indem eine gültige algebraische PostgreSQL Funktion auf die zwei Ausgangsrasterbänder und den entsprechenden Pixeltyp angewendet wird. Wenn keine Bandnummern angegeben sind, wird von jedem Raster Band 1 angenommen. Der Ergebnisraster wird nach dem Gitter des ersten Raster ausgerichtet (Skalierung, Versatz und Eckpunkte der Pixel) und hat die Ausdehnung, welche durch den Parameter "extenttype" definiert ist. Der Parameter "extenttype" kann die Werte INTERSECTION, UNION, FIRST, SECOND annehmen.
  - **ST\_MapAlgebraFct** - Verfügbarkeit: 2.0.0 Version mit 1 Rasterband: Erzeugt ein neues Rasterband, dass über eine gültige PostgreSQL Funktion für ein gegebenes Rasterband und Pixeltyp erstellt wird. Wenn kein Band bestimmt ist, wird Band 1 angenommen.
  - **ST\_MapAlgebraFct** - Verfügbarkeit: 2.0.0 Version mit 2 Rasterbändern: Erstellt einen neuen Einzelbandraster, indem eine gültige PostgreSQL Funktion auf die 2 gegebenen Rasterbänder und den entsprechenden Pixeltyp angewendet wird. Wenn kein Band bestimmt ist, wird Band 1 angenommen. Wenn der "Extent"-Typ nicht angegeben ist, wird standardmäßig INTERSECTION angenommen.
  - **ST\_MapAlgebraFctNgb** - Verfügbarkeit: 2.0.0 Version mit 1em Band: Map Algebra Nearest Neighbor mit einer benutzerdefinierten PostgreSQL Funktion. Gibt einen Raster zurück, dessen Werte sich aus einer benutzerdefinierte PL/pgsql Funktion ergeben, welche die Nachbarschaftswerte des Ausgangsrasterbandes einbezieht.
  - **ST\_Max4ma** - Verfügbarkeit: 2.0.0 Funktion zur Rasterdatenverarbeitung, die den maximalen Zellwert in der Nachbarschaft eines Pixel errechnet.
  - **ST\_Mean4ma** - Verfügbarkeit: 2.0.0 Funktion zur Rasterdatenverarbeitung, die den mittleren Zellwert in der Nachbarschaft von Pixel errechnet.
  - **ST\_Min4ma** - Verfügbarkeit: 2.0.0 Funktion zur Rasterdatenverarbeitung, die den minimalen Zellwert in der Nachbarschaft von Pixel errechnet.
  - **ST\_ModEdgeHeal** - Verfügbarkeit: 2.0 "Heilt" zwei Kanten, indem der verbindende Knoten gelöscht wird, die erste Kante modifiziert und die zweite Kante gelöscht wird. Gibt die ID des gelöschten Knoten zurück.
  - **ST\_MoveIsoNode** - Verfügbarkeit: 2.0.0 Verschiebt einen isolierten Knoten in einer Topologie von einer Stelle an eine andere. Falls die neue Geometrie apoint bereits als Knoten existiert, wird eine Fehlermeldung ausgegeben. Gibt eine Beschreibung der Verschiebung aus.
  - **ST\_NewEdgeHeal** - Verfügbarkeit: 2.0 "Heilt" zwei Kanten, indem der verbindende Knoten und beide Kanten gelöscht werden. Die beiden Kanten werden durch eine Kante ersetzt, welche dieselbe Ausrichtung wie die erste Kante hat.
  - **ST\_Node** - Verfügbarkeit: 2.0.0 Knoten eine Sammlung von Linien.
-



- **ST\_NumPatches** - Verfügbarkeit: 2.0.0 Gibt die Anzahl der Maschen einer polyedrischen Oberfläche aus. Gibt NULL zurück, wenn es sich nicht um polyedrische Geometrien handelt.
  - **ST\_OffsetCurve** - Verfügbarkeit: 2.0 Gibt eine versetzte Linie in einem bestimmten Abstand und einer bestimmten Seite von einer Eingabelinie zurück.
  - **ST\_PatchN** - Verfügbarkeit: 2.0.0 Gibt den Geometrietyt des ST\_Geometry Wertes zurück.
  - **ST\_Perimeter** - Verfügbarkeit: Mit 2.0.0 wurde die Unterstützung für geographischen Koordinaten eingeführt Gibt die Länge der Begrenzung einer polygonalen Geometrie oder Geografie zurück.
  - **ST\_PixelAsPolygon** - Verfügbarkeit: 2.0.0 Gibt die Polyongeometrie aus, die das Pixel einer bestimmten Zeile und Spalte begrenzt.
  - **ST\_PixelAsPolygons** - Verfügbarkeit: 2.0.0 Gibt die umhüllende Polyongeometrie, den Zellwert, sowie die X- und Y-Rasterkoordinate für jedes Pixel aus.
  - **ST\_Project** - Verfügbarkeit: 2.0.0 Gibt einen Punkt zurück, der von einem Startpunkt um eine bestimmte Entfernung und Peilung (Azimut) projiziert wird.
  - **ST\_Quantile** - Verfügbarkeit: 2.0.0 Berechnet die Quantile eines Rasters oder einer Rastercoverage Tabelle im Kontext von Stichproben oder Bevölkerung. Dadurch kann untersucht werden, ob ein Wert bei 25%, 50% oder 75% Perzentil des Rasters liegt.
  - **ST\_Range4ma** - Verfügbarkeit: 2.0.0 Funktion zur Rasterdatenverarbeitung, die den Wertebereich der Pixel in einer Nachbarschaft errechnet.
  - **ST\_Reclass** - Verfügbarkeit: 2.0.0 Erstellt einen neuen Raster, der aus neu klassifizierten Bändern des Originalraster besteht. Das Band "nband" ist jenes das verändert werden soll. Wenn "nband" nicht angegeben ist, wird "Band 1" angenommen. Alle anderen Bänder bleiben unverändert. Anwendungsfall: zwecks einfacherer Visualisierung ein 16BUI-Band in ein 8BUI-Band konvertieren und so weiter.
  - **ST\_RelateMatch** - Verfügbarkeit: 2.0.0 Testet, ob eine DE-9IM Schnittpunktmatrix mit einem Schnittpunktmuster übereinstimmt
  - **ST\_RemEdgeModFace** - Verfügbarkeit: 2.0 Entfernt eine Kante, und wenn die Kante zwei Flächen trennt, wird eine Fläche gelöscht und die andere Fläche so verändert, dass sie den Raum beider Flächen abdeckt.
  - **ST\_RemEdgeNewFace** - Verfügbarkeit: 2.0 Entfernt eine Kante. Falls die gelöschte Kante zwei Maschen voneinander getrennt hat, werden die ursprünglichen Maschen gelöscht und durch einer neuen Masche ersetzt.
  - **ST\_Resample** - Verfügbarkeit: 2.0.0 benötigt GDAL 1.6.1+ Skaliert einen Raster mit einem bestimmten Algorithmus, neuen Dimensionen, einer beliebigen Gitterecke und über Parameter zur Georeferenzierung des Rasters, die angegeben oder von einem anderen Raster übernommen werden können.
  - **ST\_Rescale** - Verfügbarkeit: 2.0.0 benötigt GDAL 1.6.1+ Neuabtastung eines Rasters, indem nur die Skala (oder Pixelgröße) angepasst wird. Die neuen Pixelwerte werden mit den Algorithmen NearestNeighbor (englische oder amerikanische Schreibweise), Bilinear, Cubic, CubicSpline, Lanczos, Max oder Min resampling berechnet. Die Voreinstellung ist NearestNeighbor.
  - **ST\_Reskew** - Verfügbarkeit: 2.0.0 benötigt GDAL 1.6.1+ Skaliert einen Raster, indem lediglich der Versatz (oder Rotationsparameter) angepasst wird. Neue Pixelwerte werden über NearestNeighbor, bilinear, kubisch, CubicSpline oder mit dem Lanczos-Filter errechnet. Die Standardeinstellung ist NearestNeighbor.
  - **ST\_SameAlignment** - Verfügbarkeit: 2.0.0 Gibt TRUE zurück, wenn die Raster die selbe Rotation, Skalierung, Koordinatenreferenzsystem und Versatz (Pixel können auf dasselbe Gitter gelegt werden, ohne dass die Gitterlinien durch die Pixel schneiden) aufweisen. Wenn nicht, wird FALSE und eine Beschreibung des Problems ausgegeben.
  - **ST\_SetBandIsNoData** - Verfügbarkeit: 2.0.0 Setzt die Flag "isnodata" für das Band auf TRUE.
  - **ST\_SharedPaths** - Verfügbarkeit: 2.0.0 Gibt eine Sammelgeometrie zurück, welche die gemeinsamen Strecken der beiden eingegebenen LineStrings/MultiLinestrings enthält.
  - **ST\_Slope** - Verfügbarkeit: 2.0.0 Gibt die Neigung (standardmäßig in Grad) eines Höhenrasterbandes zurück. Nützlich für Terrain-Analysen.
-

- **ST\_Snap** - Verfügbarkeit: 2.0.0 Fängt die Segmente und Knoten einer Eingabegeometrie an den Knoten einer Referenzgeometrie.
- **ST\_SnapToGrid** - Verfügbarkeit: 2.0.0 benötigt GDAL 1.6.1+ Skaliert einen Raster durch Fangen an einem Führungsgitter. Neue Pixelwerte werden über NearestNeighbor, bilinear, kubisch, CubicSpline oder mit dem Lanczos-Filter errechnet. Die Standardeinstellung ist NearestNeighbor.
- **ST\_Split** - Verfügbarkeit: 2.0.0 erfordert GEOS Gibt eine Sammlung von Geometrien zurück, die durch Aufteilung einer Geometrie durch eine andere Geometrie entstanden sind.
- **ST\_StdDev4ma** - Verfügbarkeit: 2.0.0 Funktion zur Rasterdatenverarbeitung, welche die Standardabweichung der Zellwerte in der Nachbarschaft von Pixel errechnet.
- **ST\_Sum4ma** - Verfügbarkeit: 2.0.0 Funktion zur Rasterdatenverarbeitung, die die Summe aller Zellwerte in der Nachbarschaft von Pixel errechnet.
- **ST\_SummaryStats** - Verfügbarkeit: 2.0.0 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\_Transform** - Verfügbarkeit: 2.0.0 benötigt GDAL 1.6.1+ Projiziert einen Raster von einem bekannten Koordinatenreferenzsystem in ein anderes bekanntes Koordinatenreferenzsystem um. Die Optionen für die Skalierung sind NearestNeighbor, Bilinear, Kubisch, CubicSpline und der Lanczos-Filter, die Standardeinstellung ist NearestNeighbor.
- **ST\_UnaryUnion** - Verfügbarkeit: 2.0.0 Berechnet die Vereinigung der Komponenten einer einzelnen Geometrie.
- **ST\_Union** - Verfügbarkeit: 2.0.0 Gibt die Vereinigung mehrerer Rasterkacheln in einem einzelnen Raster mit mehreren Bändern zurück.
- **ST\_ValueCount** - Verfügbarkeit: 2.0.0 Gibt Datensätze aus, die den Zellwert und die Anzahl der Pixel eines Rasterbandes (oder Rastercoveragebandes) für gegebene Werte enthalten. Wenn kein Band angegeben ist, wird Band 1 angenommen. Pixel mit dem Wert NODATA werden standardmäßig nicht gezählt; alle anderen Pixelwerte des Bandes werden ausgegeben und auf die nächste Ganzzahl gerundet.
- **TopoElementArray\_Agg** - Verfügbarkeit: 2.0.0 Gibt für eine Menge an element\_id, type Feldern (topoelements) ein topoelementarray zurück.
- **TopoGeo\_AddLineString** - Verfügbarkeit: 2.0.0 Adds a linestring to an existing topology using a tolerance and possibly splitting existing edges/faces.
- **TopoGeo\_AddPoint** - Verfügbarkeit: 2.0.0 Fügt einen Punkt, unter Berücksichtigung einer Toleranz, an eine bestehende Topologie an. Existierende Kanten werden eventuell aufgetrennt.
- **TopoGeo\_AddPolygon** - Verfügbarkeit: 2.0.0 Fügt ein Polygon, unter Berücksichtigung einer Toleranz, an eine bestehende Topologie an. Existierende Kanten/Maschen werden eventuell aufgetrennt. Gibt den Identifikator der Masche zurück.
- **TopologySummary** - Verfügbarkeit: 2.0.0 Nimmt den Namen einer Topologie und liefert eine Zusammenfassung der Gesamtsummen der Typen und Objekte in der Topologie.
- **Topology\_Load\_Tiger** - Verfügbarkeit: 2.0.0 Lädt die Tiger-Daten einer bestimmte Region in die PostGIS Topologie, transformiert sie in das Koordinatenreferenzsystem der Topologie und fängt sie entsprechend der Genauigkeitstoleranz der Topologie.
- **toTopoGeom** - Verfügbarkeit: 2.0 Wandelt eine einfache Geometrie in eine TopoGeometry um.
- **~** - Verfügbarkeit: 2.0.0 Gibt TRUE zurück, wenn das umschreibende Rechteck von A jenes von B enthält. Das umschreibende Rechteck ist in Double Precision.
- **~=** - Verfügbarkeit: 2.0.0 Gibt TRUE zurück, wenn die bounding box von A ident mit jener von B ist.

#### Erweiterte Funktionen in PostGIS 2.0

- **&&** - Erweiterung: Mit 2.0.0 wurde die Unterstützung für polyedrische Oberflächen eingeführt. Gibt TRUE zurück, wenn die 2D Bounding Box von A die 2D Bounding Box von B schneidet.

- **AddGeometryColumn** - Verbesserung: 2.0.0 use\_typmod Argument eingeführt. Standardmäßig wird eine typmod Geometrie anstelle einer Constraint-basierten Geometrie erzeugt. Entfernt eine Geometriespalte aus einer räumlichen Tabelle.
  - **Box2D** - Erweiterung: Mit 2.0.0 wurde die Unterstützung für polyedrische Oberflächen, Dreiecke und TIN eingeführt. Gibt ein BOX2D zurück, das die 2D-Ausdehnung einer Geometrie darstellt.
  - **Box3D** - Erweiterung: Mit 2.0.0 wurde die Unterstützung für polyedrische Oberflächen, Dreiecke und TIN eingeführt. Gibt ein BOX3D zurück, das die 3D-Ausdehnung einer Geometrie darstellt.
  - **CreateTopology** - Verbessert: 2.0 fügte die Signatur hinzu, die hasZ akzeptiert Erstellt ein neues Topologie-Schema und trägt es in die Tabelle topology.topology ein.
  - **Geocode** - Erweiterung: 2.0.0 Unterstützung von strukturierten Daten von TIGER 2010. Weiters wurde die Logik überarbeitet, um die Rechengeschwindigkeit und die Genauigkeit der Geokodierung zu erhöhen, und den Versatz von der Mittellinie auf die Straßenseite zu ermöglichen. Der neue Parameter max\_results kann verwendet werden, um die Anzahl der besten Ergebnisse zu beschränken oder um nur das beste Ergebnis zu erhalten. Nimmt eine Adresse als Zeichenkette (oder eine bereits standardisierte Adresse) entgegen und gibt die möglichen Punktlagen zurück. Die Ausgabe beinhaltet eine Punktgeometrie in NAD 83 Länge/Breite, eine standardisierte Adresse und eine Rangfolge (Rating) für jede Punktlage. Umso niedriger die Rangfolge ist, um so wahrscheinlicher ist die Übereinstimmung. Die Ergebnisse werden mit aufsteigender Rangfolge sortiert - dar niedrigste Rang zuerst. Optional kann die maximale Anzahl der Ergebnisse angegeben werden (Standardeinstellung ist 10) und der Bereich mit restrict\_region beschränkt werden (Standardeinstellung ist NULL)
  - **GeometryType** - Erweiterung: Mit 2.0.0 wurde die Unterstützung für polyedrische Oberflächen, Dreiecke und TIN eingeführt. Gibt den Geometriety des ST\_Geometry Wertes zurück.
  - **Populate\_Geometry\_Columns** - Erweiterung: 2.0.0 Der optionale Übergabewert use\_typmod wurde eingeführt, um bestimmen zu können, ob die Spalten mit Typmodifikatoren oder mit Check-Constraints erstellt werden sollen. Stellt sicher, dass Geometriespalten mit Typmodifikatoren definiert sind oder geeignete räumliche Beschränkungen haben.
  - **ST\_3DExtent** - Erweiterung: Mit 2.0.0 wurde die Unterstützung für polyedrische Oberflächen, Dreiecke und TIN eingeführt. Aggregatfunktion, die den 3D-Begrenzungsrahmen von Geometrien zurückgibt.
  - **ST\_Affine** - Erweiterung: Mit 2.0.0 wurde die Unterstützung für polyedrische Oberflächen, Dreiecke und TIN eingeführt. Wenden Sie eine affine 3D-Transformation auf eine Geometrie an.
  - **ST\_Area** - Erweiterung: Mit 2.0.0 wurde 2D-Unterstützung für polyedrische Oberflächen eingeführt. Gibt den geometrischen Schwerpunkt einer Geometrie zurück.
  - **ST\_AsBinary** - Erweiterung: Mit 2.0.0 wurde die Unterstützung für polyedrische Oberflächen, Dreiecke und TIN eingeführt. Rückgabe der OGC/ISO Well-Known Binary (WKB)-Darstellung der Geometrie/Geografie ohne SRID-Metadaten.
  - **ST\_AsBinary** - Erweiterung: 2.0.0 - Unterstützung für höherdimensionale Koordinatensysteme eingeführt. Rückgabe der OGC/ISO Well-Known Binary (WKB)-Darstellung der Geometrie/Geografie ohne SRID-Metadaten.
  - **ST\_AsBinary** - Erweiterung: 2.0.0 Unterstützung zum Festlegen des Endian beim geographischen Datentyp eingeführt. Rückgabe der OGC/ISO Well-Known Binary (WKB)-Darstellung der Geometrie/Geografie ohne SRID-Metadaten.
  - **ST\_AsEWKB** - Erweiterung: Mit 2.0.0 wurde die Unterstützung für polyedrische Oberflächen, Dreiecke und TIN eingeführt. Rückgabe der Extended Well-Known Binary (EWKB) Darstellung der Geometrie mit SRID-Metadaten.
  - **ST\_AsEWKT** - Erweiterung: Mit 2.0.0 wurde die Unterstützung für den geographischen Datentyp, polyedrische Oberflächen, Dreiecke und TIN eingeführt. Gibt die Well-known-Text(WKT) Darstellung der Geometrie mit den SRID-Metadaten zurück.
  - **ST\_AsGML** - Erweiterung: 2.0.0 Unterstützung durch Präfix eingeführt. Für GML3 wurde die Option 4 eingeführt, um die Verwendung von LineString anstatt von Kurven für Linien zu erlauben. Ebenfalls wurde die GML3 Unterstützung für polyedrische Oberflächen und TINS eingeführt, sowie die Option 32 zur Ausgabe der BBox. Gibt die Geometrie als GML-Element - Version 2 oder 3 - zurück.
  - **ST\_AsKML** - Erweiterung: 2.0.0 - Präfix Namensraum hinzugefügt. Standardmäßig kein Präfix Gibt die Geometrie als GML-Element - Version 2 oder 3 - zurück.
  - **ST\_Azimuth** - Erweiterung: mit 2.0.0 wurde die Unterstützung des geographischen Datentyps eingeführt. Gibt die 2-dimensionale kürzeste Strecke zwischen zwei Geometrien als Linie zurück
-



- **ST\_Dimension** - Erweiterung: 2.0.0 - Unterstützung für polyedrische Oberflächen und TIN eingeführt. Gibt die Dimension der Koordinaten für den Wert von ST\_Geometry zurück.
  - **ST\_Dump** - Erweiterung: Mit 2.0.0 wurde die Unterstützung für polyedrische Oberflächen, Dreiecke und TIN eingeführt. Gibt einen Satz von geometry\_dump Zeilen für die Komponenten einer Geometrie zurück.
  - **ST\_DumpPoints** - Erweiterung: Mit 2.0.0 wurde die Unterstützung für polyedrische Oberflächen, Dreiecke und TIN eingeführt. Gibt eine Zusammenfassung des Inhalts einer Geometrie wieder.
  - **ST\_Expand** - Erweiterung: Mit 2.0.0 wurde die Unterstützung für polyedrische Oberflächen, Dreiecke und TIN eingeführt. Gibt einen Begrenzungsrahmen zurück, der aus einem anderen Begrenzungsrahmen oder einer Geometrie erweitert wurde.
  - **ST\_Extent** - Erweiterung: Mit 2.0.0 wurde die Unterstützung für polyedrische Oberflächen, Dreiecke und TIN eingeführt. Aggregatfunktion, die den Begrenzungsrahmen von Geometrien zurückgibt.
  - **ST\_Force2D** - Erweiterung: Mit 2.0.0 wurde die Unterstützung für polyedrische Oberflächen eingeführt. Die Geometrien in einen "2-dimensionalen Modus" zwingen.
  - **ST\_Force3D** - Erweiterung: Mit 2.0.0 wurde die Unterstützung für polyedrische Oberflächen eingeführt. Zwingt die Geometrien in einen XYZ Modus. Dies ist ein Alias für ST\_Force3DZ.
  - **ST\_Force3DZ** - Erweiterung: Mit 2.0.0 wurde die Unterstützung für polyedrische Oberflächen eingeführt. Zwingt die Geometrien in einen XYZ Modus.
  - **ST\_ForceCollection** - Erweiterung: Mit 2.0.0 wurde die Unterstützung für polyedrische Oberflächen eingeführt. Wandelt eine Geometrie in eine GEOMETRYCOLLECTION um.
  - **ST\_ForceRHR** - Erweiterung: Mit 2.0.0 wurde die Unterstützung für polyedrische Oberflächen eingeführt. Orientiert die Knoten in einem Polygon so, dass sie der Drei-Finger-Regel folgen.
  - **ST\_GMLToSQL** - Erweiterung: Mit 2.0.0 wurde die Unterstützung für polyedrische Oberflächen und TIN eingeführt. Gibt einen spezifizierten ST\_Geometry Wert aus einer GML-Darstellung zurück. Dies ist ein Aliasname für ST\_GeomFromGML
  - **ST\_GMLToSQL** - Erweiterung: 2.0.0 Standardwert für den optionalen Parameter SRID eingefügt. Gibt einen spezifizierten ST\_Geometry Wert aus einer GML-Darstellung zurück. Dies ist ein Aliasname für ST\_GeomFromGML
  - **ST\_GeomFromEWKB** - Erweiterung: Mit 2.0.0 wurde die Unterstützung für polyedrische Oberflächen und TIN eingeführt. Gibt einen geometrischen Datentyp (ST\_Geometry) aus einer Well-known-Binary (WKB) Darstellung zurück.
  - **ST\_GeomFromEWKT** - Erweiterung: Mit 2.0.0 wurde die Unterstützung für polyedrische Oberflächen und TIN eingeführt. Gibt einen spezifizierten ST\_Geometry-Wert von einer erweiterten Well-known-Text Darstellung (EWKT) zurück.
  - **ST\_GeomFromGML** - Erweiterung: Mit 2.0.0 wurde die Unterstützung für polyedrische Oberflächen und TIN eingeführt. Nimmt als Eingabe eine GML-Darstellung der Geometrie und gibt ein geometrisches PostGIS-Objekt aus.
  - **ST\_GeomFromGML** - Erweiterung: 2.0.0 Standardwert für den optionalen Parameter SRID eingefügt. Nimmt als Eingabe eine GML-Darstellung der Geometrie und gibt ein geometrisches PostGIS-Objekt aus.
  - **ST\_GeometryN** - Erweiterung: Mit 2.0.0 wurde die Unterstützung für polyedrische Oberflächen, Dreiecke und TIN eingeführt. Gibt den Geometriertyp des ST\_Geometry Wertes zurück.
  - **ST\_GeometryType** - Erweiterung: Mit 2.0.0 wurde die Unterstützung für polyedrische Oberflächen eingeführt. Gibt den Geometriertyp des ST\_Geometry Wertes zurück.
  - **ST\_IsClosed** - Erweiterung: Mit 2.0.0 wurde die Unterstützung für polyedrische Oberflächen eingeführt. Gibt den Wert TRUE zurück, wenn die Anfangs- und Endpunkte des LINESTRING's zusammenfallen. Bei polyedrischen Oberflächen, wenn sie geschlossen (volumetrisch) sind.
  - **ST\_MakeEnvelope** - Erweiterung: 2.0: es wurde die Möglichkeit eingeführt, eine Einhüllende/Envelope festzulegen, ohne dass die SRID spezifiziert ist. Erzeugt ein rechteckiges Polygon aus den gegebenen Minimum- und Maximumwerten. Die Eingabewerte müssen in dem Koordinatenreferenzsystem sein, welches durch die SRID angegeben wird.
  - **ST\_MakeValid** - Verbessert: 2.0.1, Geschwindigkeitsverbesserungen Versucht, eine ungültige Geometrie gültig zu machen, ohne dass Scheitelpunkte verloren gehen.
-

- **ST\_NPoints** - Erweiterung: Mit 2.0.0 wurde die Unterstützung für polyedrische Oberflächen eingeführt. Gibt die Anzahl der Punkte (Knoten) einer Geometrie zurück.
- **ST\_NumGeometries** - Erweiterung: Mit 2.0.0 wurde die Unterstützung für polyedrische Oberflächen, Dreiecke und TIN eingeführt. Gibt die Anzahl der Punkte einer Geometrie zurück. Funktioniert für alle Geometrien.
- **ST\_Relate** - Verbessert: 2.0.0 - Unterstützung für die Angabe von Grenzknotenregeln hinzugefügt. Prüft, ob zwei Geometrien eine topologische Beziehung haben, die einem Schnittpunktmatrixmuster entspricht, oder berechnet ihre Schnittpunktmatrix
- **ST\_Rotate** - Erweiterung: Mit 2.0.0 wurde die Unterstützung für polyedrische Oberflächen, Dreiecke und TIN eingeführt. Dreht eine Geometrie um einen Ursprungspunkt.
- **ST\_Rotate** - Verbessert: In Version 2.0.0 wurden zusätzliche Parameter zur Angabe des Ursprungs der Drehung hinzugefügt. Dreht eine Geometrie um einen Ursprungspunkt.
- **ST\_RotateX** - Erweiterung: Mit 2.0.0 wurde die Unterstützung für polyedrische Oberflächen, Dreiecke und TIN eingeführt. Dreht eine Geometrie um die X-Achse.
- **ST\_RotateY** - Erweiterung: Mit 2.0.0 wurde die Unterstützung für polyedrische Oberflächen, Dreiecke und TIN eingeführt. Dreht eine Geometrie um die Y-Achse.
- **ST\_RotateZ** - Erweiterung: Mit 2.0.0 wurde die Unterstützung für polyedrische Oberflächen, Dreiecke und TIN eingeführt. Dreht eine Geometrie um die Z-Achse.
- **ST\_Scale** - Erweiterung: Mit 2.0.0 wurde die Unterstützung für polyedrische Oberflächen, Dreiecke und TIN eingeführt. Skaliert eine Geometrie um bestimmte Faktoren.
- **ST\_ShiftLongitude** - Erweiterung: Mit 2.0.0 wurde die Unterstützung für polyedrische Oberflächen und TIN eingeführt. Verschiebt die Längenkoordinaten einer Geometrie zwischen -180..180 und 0..360.
- **ST\_Summary** - Erweiterung: 2.0.0 Unterstützung für geographische Koordinaten hinzugefügt Gibt eine Zusammenfassung des Inhalts einer Geometrie wieder.
- **ST\_Transform** - Erweiterung: Mit 2.0.0 wurde die Unterstützung für polyedrische Oberflächen eingeführt. Rückgabe einer neuen Geometrie mit in ein anderes räumliches Bezugssystem transformierten Koordinaten.
- **ST\_Value** - Erweiterung: 2.0.0 Der optionale Übergabewert "exclude\_nodata\_value" wurde hinzugefügt. Gibt den Zellwert eines Pixels aus, das über columnx und rowy oder durch einen bestimmten geometrischen Punkt angegeben wird. Die Bandnummern beginnen mit 1 und wenn keine Bandnummer angegeben ist, dann wird Band 1 angenommen. Wenn exclude\_nodata\_value auf FALSE gesetzt ist, werden auch die Pixel mit einem nodata Wert mit einbezogen. Wenn exclude\_nodata\_value nicht übergeben wird, dann wird er über die Metadaten des Rasters ausgelesen.
- **ValidateTopology** - Erweiterung: 2.0.0 effizientere Ermittlung sich überkreuzender Kanten. Falsch positive Fehlermeldungen von früheren Versionen fixiert. Liefert eine Menge validate\_topology\_return\_type Objekte, die Probleme mit der Topologie beschreiben.

#### Geänderte Funktionen in PostGIS 2.0

- **AddGeometryColumn** - Änderung: 2.0.0 Diese Funktion aktualisiert die geometry\_columns Tabelle nicht mehr, da geometry\_columns jetzt ein View ist, welcher den Systemkatalog ausliest. Standardmäßig werden auch keine Bedingungen/constraints erzeugt, sondern es wird der in PostgreSQL integrierte Typmodifikaor verwendet. So entspricht zum Beispiel die Erzeugung einer wgs84 POINT Spalte mit dieser Funktion: ALTER TABLE some\_table ADD COLUMN geom geometry(Point,4326); Entfernt eine Geometriespalte aus einer räumlichen Tabelle.
- **AddGeometryColumn** - Änderung: 2.0.0 Falls Sie das alte Verhalten mit Constraints wünschen, setzen Sie bitte use\_typmod vom standardmäßigen true auf false. Entfernt eine Geometriespalte aus einer räumlichen Tabelle.
- **AddGeometryColumn** - Änderung: 2.0.0 Views können nicht mehr händisch in "geometry\_columns" registriert werden. Views auf eine Geometrie in Typmod-Tabellen, bei denen keine Adapterfunktion verwendet wird, registrieren sich selbst auf korrekte Weise, da sie die Typmod-Verhaltensweise von der Spalte der Stammtabelle erben. Views die 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 . Entfernt eine Geometriespalte aus einer räumlichen Tabelle.

- **Box3D** - Änderung: 2.0.0 In Versionen vor 2.0 war dies üblicherweise Box2D anstelle von Box3D. Da Box2D überholt ist, wurde dies zu Box3D geändert. Stellt das umschreibende Rechteck eines Raster als Box3D dar.
  - **DropGeometryColumn** - Änderung: 2.0.0 Diese Funktion wurde zwecks Abwärtskompatibilität eingeführt. Seit geometry\_columns ein View auf den Systemkatalog ist, können Sie die Geometriespalte, so wie jede andere Tabellenspalte, mit ALTER TABLE löschen. Entfernt eine Geometriespalte aus einer räumlichen Tabelle.
  - **DropGeometryTable** - Änderung: 2.0.0 Diese Funktion wurde zwecks Abwärtskompatibilität eingeführt. Seit geometry\_columns ein View auf den Systemkatalog ist, können Sie eine Tabelle mit einer Geometriespalte, so wie jede andere Tabelle, mit DROP TABLE löschen. Löscht eine Tabelle und alle Referenzen in dem geometry\_columns View.
  - **Populate\_Geometry\_Columns** - Änderung: 2.0.0 Standardmäßig werden nun Typmodifikatoren anstelle von Check-Constraints für die Beschränkung des Geometrietyps verwendet. Sie können nach wie vor stattdessen die Verhaltensweise mit Check-Constraints verwenden, indem Sie die neu eingeführte Variable use\_typmod auf FALSE setzen. Stellt sicher, dass Geometriespalten mit Typmodifikatoren definiert sind oder geeignete räumliche Beschränkungen haben.
  - **ST\_3DExtent** - Geändert: 2.0.0 In früheren Versionen wurde dies als ST\_Extent3D bezeichnet. Aggregatfunktion, die den 3D-Begrenzungsrahmen von Geometrien zurückgibt.
  - **ST\_3DLength** - Änderung: 2.0.0 In Vorgängerversionen als ST\_Length3D bezeichnet. Gibt den geometrischen Schwerpunkt einer Geometrie zurück.
  - **ST\_3DMakeBox** - Geändert: 2.0.0 In früheren Versionen hieß diese Funktion ST\_MakeBox3D Erzeugt einen BOX3D, der durch zwei 3D-Punktgeometrien definiert ist.
  - **ST\_3DPerimeter** - Änderung: 2.0.0 In Vorgängerversionen als ST\_Perimeter3D bezeichnet. Gibt den geometrischen Schwerpunkt einer Geometrie zurück.
  - **ST\_AsBinary** - Änderung: 2.0.0 - Eingabewerte für diese Funktion dürfen nicht "unknown" sein -- es muss sich um eine Geometrie handeln. Konstrukte, wie ST\_AsBinary('POINT(1 2)'), sind nicht länger gültig und geben folgende Fehlermeldung aus: n st\_asbinary(unknown) is not unique error. Dieser Code muss in ST\_AsBinary('POINT(1 2)::geometry'); geändert werden. Falls dies nicht möglich ist, so installieren Sie bitte legacy.sql. Rückgabe der OGC/ISO Well-Known Binary (WKB)-Darstellung der Geometrie/Geografie ohne SRID-Metadaten.
  - **ST\_AsGML** - Änderung: 2.0.0 verwendet standardmäßig benannte Argumente. Gibt die Geometrie als GML-Element - Version 2 oder 3 - zurück.
  - **ST\_AsGeoJSON** - Änderung: 2.0.0 Unterstützung für Standardargumente und benannte Argumente. Rückgabe einer Geometrie oder eines Merkmals im GeoJSON-Format.
  - **ST\_AsSVG** - Änderung: 2.0.0 verwendet Standardargumente und unterstützt benannte Argumente. Gibt eine Geometrie als SVG-Pfad aus.
  - **ST\_EndPoint** - Änderung: 2.0.0 unterstützt die Verarbeitung von MultiLinestring's die nur aus einer einzelnen Geometrie bestehen, nicht mehr. In früheren Versionen von PostGIS gab die Funktion bei einem aus einer einzelnen Linie bestehender MultiLinestring den Anfangspunkt zurück. Ab 2.0.0 gibt sie nur NULL zurück, so wie bei jedem anderen MultiLinestring. Die alte Verhaltensweise war undokumentiert, aber Anwender, die annahmen, dass Sie Ihre Daten als LINESTRING vorliegen haben, könnten in 2.0 dieses zurückgegebene NULL bemerken. Gibt die Anzahl der Stützpunkte eines ST\_LineString oder eines ST\_CircularString zurück.
  - **ST\_GDALDrivers** - Änderung: 2.0.6, 2.1.3 - standardmäßig ist kein Treiber aktiviert, solange die GUC oder die Umgebungsvariable "gdal\_enabled\_drivers" nicht gesetzt sind. Gibt eine Liste der Rasterformate aus, die von PostGIS über die Bibliothek GDAL unterstützt werden. Nur die Formate mit can\_write=True können von ST\_AsGDALRaster verwendet werden.
  - **ST\_GeomFromText** - Änderung: 2.0.0 - In Vorgängerversionen von PostGIS war ST\_GeomFromText('GEOMETRYCOLLECTION 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. Gibt einen spezifizierten ST\_Geometry Wert aus einer Well-known-Text Darstellung (WKT) zurück.
  - **ST\_GeometryN** - Änderung: 2.0.0 Vorangegangene Versionen geben bei Einzelgeometrien NULL zurück. Dies wurde geändert um die Geometrie für den ST\_GeometryN(...,1) Fall zurückzugeben. Gibt den Geometriety des ST\_Geometry Wertes zurück.
-

- **ST\_IsEmpty** - Änderung: 2.0.0 - In Vorgängerversionen von PostGIS war `ST_GeomFromText('GEOMETRYCOLLECTION(EMPTY` erlaubt. Um eine bessere Übereinstimmung mit der SQL/MM Norm zu erreichen, ist dies nun nicht mehr gestattet. Prüft, ob eine Geometrie leer ist.
- **ST\_Length** - Änderung: 2.0.0 Wesentliche Änderung -- In früheren Versionen ergab die Anwendung auf ein MULTI/POLYGON vom geographischen Datentyp den Umfang des POLYGON/MULTIPOLYGON. In 2.0.0 wurde dies geändert und es wird jetzt 0 zurückgegeben, damit es mit der Verhaltensweise beim geometrischen Datentyp übereinstimmt. Verwenden Sie bitte `ST_Perimeter`, wenn Sie den Umfang eines Polygons wissen wollen. Gibt den geometrischen Schwerpunkt einer Geometrie zurück.
- **ST\_LocateAlong** - Änderung: 2.0.0 In Vorgängerversionen als `ST_Locate_Along_Measure` bezeichnet. Der alte Name ist überholt und wird in der Zukunft entfernt ist aber noch verfügbar. Gibt die Punkte auf einer Geometrie zurück, die einem Messwert entsprechen.
- **ST\_LocateBetween** - Änderung: 2.0.0 In Vorgängerversionen als `ST_Locate_Along_Measure` bezeichnet. Der alte Name ist überholt und wird in der Zukunft entfernt ist aber noch verfügbar. Gibt die Teile einer Geometrie zurück, die einem Messbereich entsprechen.
- **ST\_ModEdgeSplit** - Änderung: 2.0 - In Vorgängerversionen fälschlicherweise als `ST_ModEdgesSplit` bezeichnet. Trennt eine Kante auf, indem ein neuer Knoten entlang einer bestehenden Kante erstellt wird. Ändert die ursprüngliche Kante und fügt eine neue Kante hinzu.
- **ST\_NumGeometries** - Änderung: 2.0.0 Bei früheren Versionen wurde NULL zurückgegeben, wenn die Geometrie nicht vom Typ `GEOMETRYCOLLECTION/MULTI` war. 2.0.0+ gibt nun 1 für Einzelgeometrien, wie `POLYGON`, `LINestring`, `POINT` zurück. Gibt die Anzahl der Punkte einer Geometrie zurück. Funktioniert für alle Geometrien.
- **ST\_NumInteriorRings** - Änderung: 2.0.0 - In früheren Versionen war ein `MULTIPOLYGON` als Eingabe erlaubt, wobei die Anzahl der inneren Ringe des ersten Polygons ausgegeben wurde. Gibt die Anzahl der inneren Ringe einer Polygoneometrie aus.
- **ST\_PointN** - Änderung: 2.0.0 arbeitet nicht mehr mit `MultiLinestring`'s, die nur eine einzelne Geometrie enthalten. In früheren Versionen von PostGIS gab die Funktion bei einem, aus einer einzelnen Linie bestehender `MultiLinestring`, den Anfangspunkt zurück. Ab 2.0.0 wird, so wie bei jedem anderen `MultiLinestring` auch, NULL zurückgegeben. Gibt die Anzahl der Stützpunkte eines `ST_LineString` oder eines `ST_CircularString` zurück.
- **ST\_ScaleX** - Änderung: 2.0.0 In `WKTRaster` Versionen wurde dies als `ST_PixelSizeX` bezeichnet. Gibt die X-Komponente der Pixelbreite in den Einheiten des Koordinatenreferenzsystems aus.
- **ST\_ScaleY** - Änderung: 2.0.0. Versionen von `WKTRaster` haben dies als "`ST_PixelSizeY`" bezeichnet. Gibt die Y-Komponente der Pixelhöhe in den Einheiten des Koordinatenreferenzsystems aus.
- **ST\_SetScale** - Änderung: 2.0.0. Versionen von `WKTRaster` haben dies als "`ST_SetPixelSizeY`" bezeichnet. Dies wurde mit 2.0.0 geändert. Setzt die X- und Y-Größe der Pixel in den Einheiten des Koordinatenreferenzsystems. Entweder eine Zahl pro Pixel oder Breite und Höhe.
- **ST\_StartPoint** - Änderung: 2.0.0 unterstützt die Verarbeitung von `MultiLinestring`'s die nur aus einer einzelnen Geometrie bestehen, nicht mehr. In früheren Versionen von PostGIS gab die Funktion bei einem aus einer einzelnen Linie bestehender `MultiLinestring` den Anfangspunkt zurück. Ab 2.0.0 gibt sie nur NULL zurück, so wie bei jedem anderen `MultiLinestring`. Die alte Verhaltensweise war undokumentiert, aber Anwender, die annahmen, dass Sie Ihre Daten als `LINestring` vorliegen haben, könnten in 2.0 dieses zurückgegebene NULL bemerken. Gibt den ersten Punkt eines `LineString` zurück.

### 13.12.13 PostGIS-Funktionen neu oder erweitert in 1.5

Die unten aufgeführten Funktionen sind PostGIS-Funktionen, die hinzugefügt oder erweitert wurden.

Neue Funktionen in PostGIS 1.5

- **&&** - Verfügbarkeit: Mit 1.5.0 wurde die Unterstützung von geographischen Koordinaten eingeführt. Gibt TRUE zurück, wenn die 2D Bounding Box von A die 2D Bounding Box von B schneidet.
- **PostGIS\_LibXML\_Version** - Verfügbarkeit: 1.5 Gibt die Versionsnummer der libxml2-Bibliothek zurück.

- **ST\_AddMeasure** - Verfügbarkeit: 1.5.0 Interpoliert Maße entlang einer linearen Geometrie.
  - **ST\_AsBinary** - Verfügbarkeit: 1.5.0 Unterstützung von geographischen Koordinaten. Rückgabe der OGC/ISO Well-Known Binary (WKB)-Darstellung der Geometrie/Geografie ohne SRID-Metadaten.
  - **ST\_AsGML** - Verfügbarkeit: 1.5.0 Unterstützung von geographischen Koordinaten. Gibt die Geometrie als GML-Element - Version 2 oder 3 - zurück.
  - **ST\_AsGeoJSON** - Verfügbarkeit: 1.5.0 Unterstützung von geographischen Koordinaten. Rückgabe einer Geometrie oder eines Merkmals im GeoJSON-Format.
  - **ST\_AsText** - Verfügbarkeit: 1.5 - Unterstützung von geographischen Koordinaten. Gibt die Well-known-Text(WKT) Darstellung der Geometrie/Geographie ohne die SRID Metadaten zurück.
  - **ST\_Buffer** - Verfügbarkeit: 1.5 - ST\_Buffer wurde um die Unterstützung von Abschlusstücken/endcaps und Join-Typen erweitert. Diese können zum Beispiel dazu verwendet werden, um Linienzüge von Straßen in Straßenpolygone mit flachen oder rechtwinkligen Abschlüssen anstatt mit runden Enden umzuwandeln. Ein schlanker Adapter für den geographischen Datentyp wurde hinzugefügt. Berechnet eine Geometrie, die alle Punkte innerhalb eines bestimmten Abstands zu einer Geometrie umfasst.
  - **ST\_ClosestPoint** - Verfügbarkeit: 1.5.0 Gibt den 2D-Punkt auf g1 zurück, der g2 am nächsten ist. Dies ist der erste Punkt der kürzesten Linie von einer Geometrie zur anderen.
  - **ST\_CollectionExtract** - Verfügbarkeit: 1.5.0 Gibt bei einer Geometriesammlung eine Multi-Geometrie zurück, die nur Elemente eines bestimmten Typs enthält.
  - **ST\_Covers** - Verfügbarkeit: 1.5 - Unterstützung von geographischen Koordinaten. Prüft, ob jeder Punkt von B in A liegt
  - **ST\_DFullyWithin** - Verfügbarkeit: 1.5.0 Tests if a geometry is entirely inside a distance of another
  - **ST\_DWithin** - Verfügbarkeit: Mit Version 1.5.0 wurde die Unterstützung für Geografie eingeführt. Prüft, ob zwei Geometrien innerhalb eines bestimmten Abstands liegen
  - **ST\_Distance** - Verfügbarkeit: 1.5.0 die Unterstützung des geographischen Datentyps wurde eingeführt. Geschwindigkeitsverbesserungen bei einer umfangreichen Geometrie und bei einer Geometrie mit vielen Knoten Gibt die größte 3-dimensionale Distanz zwischen zwei geometrischen Objekten als Linie zurück
  - **ST\_DistanceSphere** - Verfügbarkeit: 1.5 die Unterstützung für weitere geometrische Datentypen neben Punkten eingeführt. Bei Vorgängerversionen wurden nur Punkte unterstützt. Gibt die kürzeste Distanz zwischen zwei geometrischen Objekten zurück, die über Länge, Breite und ein bestimmtes Referenzellipsoid gegeben sind. Vorgängerversionen von PostGIS 1.5 unterstützten nur Punkte.
  - **ST\_DistanceSpheroid** - Verfügbarkeit: 1.5 die Unterstützung für weitere geometrische Datentypen neben Punkten eingeführt. Bei Vorgängerversionen wurden nur Punkte unterstützt. Gibt die kürzeste Distanz zwischen zwei geometrischen Objekten zurück, die über Länge, Breite und ein bestimmtes Referenzellipsoid gegeben sind. Vorgängerversionen von PostGIS 1.5 unterstützten nur Punkte.
  - **ST\_DumpPoints** - Verfügbarkeit: 1.5.0 Gibt eine Zusammenfassung des Inhalts einer Geometrie wieder.
  - **ST\_Envelope** - Verfügbarkeit: 1.5.0 Änderung der Verhaltensweise insofern, das die Ausgabe in Double Precision anstelle von Float4 erfolgt Gibt eine Geometrie in doppelter Genauigkeit (float8) zurück, welche das Umgebungsrechteck der beigestellten Geometrie darstellt.
  - **ST\_Expand** - Verfügbarkeit: 1.5.0 Verhalten geändert, um double precision statt float4 Koordinaten auszugeben. Gibt einen Begrenzungsrahmen zurück, der aus einem anderen Begrenzungsrahmen oder einer Geometrie erweitert wurde.
  - **ST\_GMLToSQL** - Verfügbarkeit: 1.5, benötigt libxml2 1.6+ Gibt einen spezifizierten ST\_Geometry Wert aus einer GML-Darstellung zurück. Dies ist ein Aliasname für ST\_GeomFromGML
  - **ST\_GeomFromGML** - Verfügbarkeit: 1.5, benötigt libxml2 1.6+ Nimmt als Eingabe eine GML-Darstellung der Geometrie und gibt ein geometrisches PostGIS-Objekt aus.
  - **ST\_GeomFromKML** - Verfügbarkeit: 1.5, benötigt libxml2 2.6+ Nimmt als Eingabe eine KML-Darstellung der Geometrie und gibt ein geometrisches PostGIS-Objekt aus.
-



- **ST\_HausdorffDistance** - Verfügbarkeit: 1.5.0 Gibt den kürzesten 3-dimensionalen Abstand zwischen zwei geometrischen Objekten als Linie zurück
- **ST\_Intersection** - Verfügbarkeit: Mit Version 1.5 wurde die Unterstützung für den Datentyp Geographie eingeführt. Berechnet eine Geometrie, die den gemeinsamen Teil der Geometrien A und B darstellt.
- **ST\_Intersects** - Verfügbarkeit: Mit Version 1.5 wurde die Unterstützung für Geografie eingeführt. Prüft, ob sich zwei Geometrien schneiden (sie haben mindestens einen Punkt gemeinsam)
- **ST\_Length** - Verfügbarkeit: 1.5.0 Unterstützung von geographischen Koordinaten. Gibt den geometrischen Schwerpunkt einer Geometrie zurück.
- **ST\_LongestLine** - Verfügbarkeit: 1.5.0 Gibt die größte 3-dimensionale Distanz zwischen zwei geometrischen Objekten als Linie zurück
- **ST\_MakeEnvelope** - Verfügbarkeit: 1.5 Erzeugt ein rechteckiges Polygon aus den gegebenen Minimum- und Maximumwerten. Die Eingabewerte müssen in dem Koordinatenreferenzsystem sein, welches durch die SRID angegeben wird.
- **ST\_MaxDistance** - Verfügbarkeit: 1.5.0 Gibt die größte 2-dimensionale Distanz zwischen zwei geometrischen Objekten in projizierten Einheiten zurück.
- **ST\_ShortestLine** - Verfügbarkeit: 1.5.0 Gibt die 2-dimensionale kürzeste Strecke zwischen zwei Geometrien als Linie zurück
- **~=** - Verfügbarkeit: 1.5.0 "Verhaltensänderung" Gibt TRUE zurück, wenn die bounding box von A ident mit jener von B ist.

#### 13.12.14 PostGIS-Funktionen neu oder erweitert in 1.4

Die unten aufgeführten Funktionen sind PostGIS-Funktionen, die hinzugefügt oder erweitert wurden.

Neue Funktionen in PostGIS 1.4

- **Populate\_Geometry\_Columns** - Verfügbarkeit: 1.4.0 Stellt sicher, dass Geometriespalten mit Typmodifikatoren definiert sind oder geeignete räumliche Beschränkungen haben.
- **ST\_Collect** - Verfügbarkeit: 1.4.0 - **ST\_MakeLine**(geomarray) wurde eingeführt. **ST\_MakeLine** Aggregatfunktion wurde verbessert, um mehr Punkte schneller handhaben zu können. Erzeugt eine GeometryCollection oder Multi\*-Geometrie aus einer Reihe von Geometrien.
- **ST\_ContainsProperly** - Verfügbarkeit: 1.4.0 Prüft, ob jeder Punkt von B im Inneren von A liegt
- **ST\_GeoHash** - Verfügbarkeit: 1.4.0 Gibt die Geometrie in der GeoHash Darstellung aus.
- **ST\_IsValidReason** - Verfügbarkeit: 1.4 Gibt einen Text zurück, der angibt, ob eine Geometrie gültig ist, oder einen Grund für die Ungültigkeit.
- **ST\_LineCrossingDirection** - Verfügbarkeit: 1.4 Gibt eine Zahl zurück, die das Kreuzungsverhalten von zwei LineStrings angibt
- **ST\_LocateBetweenElevations** - Verfügbarkeit: 1.4.0 Gibt die Teile einer Geometrie zurück, die in einem Höhenbereich (Z) liegen.
- **ST\_MakeLine** - Verfügbarkeit: 1.4.0 - **ST\_MakeLine**(geomarray) wurde eingeführt. **ST\_MakeLine** Aggregatfunktion wurde verbessert, um mehr Punkte schneller handhaben zu können. Erzeugt einen Linienzug aus einer Punkt-, Mehrfachpunkt- oder Liniengeometrie.
- **ST\_MinimumBoundingCircle** - Verfügbarkeit: 1.4.0 Gibt das kleinste Kreispolygon zurück, das eine Geometrie enthält.
- **ST\_Union** - Verfügbarkeit: 1.4.0 - **ST\_Union** wurde verbessert. **ST\_Union**(geomarray) wurde eingeführt und auch schnellere Aggregat-Sammlung in PostgreSQL. Berechnet eine Geometrie, die die Punktmengenvereinigung der Eingabegerometrien darstellt.

### 13.12.15 PostGIS-Funktionen neu oder erweitert in 1.3

Die unten aufgeführten Funktionen sind PostGIS-Funktionen, die hinzugefügt oder erweitert wurden.

Neue Funktionen in PostGIS 1.3

- **ST\_AsGML** - Verfügbarkeit: 1.3.2 Gibt die Geometrie als GML-Element - Version 2 oder 3 - zurück.
  - **ST\_AsGeoJSON** - Verfügbarkeit: 1.3.4 Rückgabe einer Geometrie oder eines Merkmals im GeoJSON-Format.
  - **ST\_CurveToLine** - Verfügbarkeit: 1.3.0 Konvertiert eine Geometrie mit Kurven in eine lineare Geometrie.
  - **ST\_LineToCurve** - Verfügbarkeit: 1.3.0 Konvertiert eine lineare Geometrie in eine gekrümmte Geometrie.
  - **ST\_SimplifyPreserveTopology** - Verfügbarkeit: 1.3.3 Gibt eine vereinfachte und gültige Darstellung einer Geometrie zurück, die den Douglas-Peucker-Algorithmus verwendet.
-

## Chapter 14

# Meldung von Problemen

### 14.1 Software Bugs melden

Effektive Fehlerberichte sind ein wesentlicher Beitrag zur Weiterentwicklung von PostGIS. Am wirksamsten ist ein Fehlerbericht dann, wenn er von den PostGIS-Entwicklern reproduziert werden kann. Idealerweise enthält er ein Skript das den Fehler auslöst und eine vollständige Beschreibung der Umgebung in der er aufgetreten ist. Ausreichend gute Information liefert `SELECT postgis_full_version()` [für PostGIS] und `SELECT version()` [für PostgreSQL].

Falls Sie nicht die aktuelle Version verwenden, sollten Sie zuerst unter [release changelog](#) nachsehen, ob Ihr Bug nicht bereits bereinigt wurde.

Die Verwendung des [PostGIS bug tracker](#) stellt sicher dass Ihre Berichte nicht verworfen werden, und dass Sie über die Prozessabwicklung am Laufenden gehalten werden. Bevor Sie einen neuen Fehler melden fragen Sie bitte die Datenbank ab ob der Fehler schon bekannt ist. Wenn es ein bekannter Fehler ist, so fügen Sie bitte jegliche neue Information die Sie herausgefunden haben hinzu.

Vielleicht möchten Sie zuvor Simon Tatham's Artikel über [How to Report Bugs Effectively](#) lesen, bevor Sie einen Fehlerbericht senden.

### 14.2 Probleme mit der Dokumentation melden

Die Dokumentation sollte die Eigenschaften und das Verhalten der Software exakt 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:

```
git clone https://git.osgeo.org/gitea/postgis/postgis.git
```

Dies wird im Verzeichnis `./trunk` gespeichert

2. Erledigen Sie die Änderungen an der Dokumentation mit Ihrem Lieblingstexteditor. Auf Unix, tippen Sie (zum Beispiel):

```
vim doc/postgis.xml
```

Bedenken Sie bitte, dass die Dokumentation in DocBook XML und nicht in HTML geschrieben ist. Falls Sie damit nicht vertraut sind, so folgen Sie bitte dem Beispiel in der restlichen Dokumentation.

---



3. Erzeugung einer Patchdatei, welche die Unterschiede zur Master-Kopie der Dokumentation enthält. Unter Unix tippen Sie bitte:

```
git diff doc/postgis.xml > doc.patch
```

4. Fügen Sie den Patch einem neuen Thema/Issue im Bug Tracker bei.
-

## Appendix A

# Anhang

### A.1 PostGIS 3.6.0

2025/xx/xx