ALIBABA CLOUD

Alibaba Cloud

PolarDB PostgreSQL

Spatio-temporal Database

Document Version: 20220705

Alibaba Cloud

Legal disclaimer

Alibaba Cloud reminds you to carefully read and fully understand the terms and conditions of this legal disclaimer before you read or use this document. If you have read or used this document, it shall be deemed as your total acceptance of this legal disclaimer.

1. You shall download and obtain this document from the Alibaba Cloud website or other Alibaba Cloud-authorized channels, and use this document for your own legal business activities only. The content of this document is considered confidential information of Alibaba Cloud. You shall strictly abide by the confidentiality obligations. No part of this document shall be disclosed or provided to any third party for use without the prior written consent of Alibaba Cloud.

2. No part of this document shall be excerpted, translated, reproduced, transmitted, or disseminated by any organization, company or individual in any form or by any means without the prior written consent of Alibaba Cloud.

3. The content of this document may be changed because of product version upgrade, adjustment, or other reasons. Alibaba Cloud reserves the right to modify the content of this document without notice and an updated version of this document will be released through Alibaba Cloud-authorized channels from time to time. You should pay attention to the version changes of this document as they occur and download and obtain the most up-to-date version of this document from Alibaba Cloud-authorized channels.

4. This document serves only as a reference guide for your use of Alibaba Cloud products and services. Alibaba Cloud provides this document based on the "status quo", "being defective", and "existing functions" of its products and services. Alibaba Cloud makes every effort to provide relevant operational guidance based on existing technologies. However, Alibaba Cloud hereby makes a clear statement that it in no way guarantees the accuracy, integrity, applicability, and reliability of the content of this document, either explicitly or implicitly. Alibaba Cloud shall not take legal responsibility for any errors or lost profits incurred by any organization, company, or individual arising from download, use, or trust in this document. Alibaba Cloud shall not, under any circumstances, take responsibility for any indirect, consequential, punitive, contingent, special, or punitive damages, including lost profits arising from the use or trust in this document (even if Alibaba Cloud has been notified of the possibility of such a loss).

5. By law, all the contents in Alibaba Cloud documents, including but not limited to pictures, architecture design, page layout, and text description, are intellectual property of Alibaba Cloud and/or its affiliates. This intellectual property includes, but is not limited to, trademark rights, patent rights, copyrights, and trade secrets. No part of this document shall be used, modified, reproduced, publicly transmitted, changed, disseminated, distributed, or published without the prior written consent of Alibaba Cloud and/or its affiliates. The names owned by Alibaba Cloud shall not be used, published, or reproduced for marketing, advertising, promotion, or other purposes without the prior written consent of Alibaba Cloud. The names owned by Alibaba Cloud include, but are not limited to, "Alibaba Cloud", "Aliyun", "HiChina", and other brands of Alibaba Cloud and/or its affiliates, which appear separately or in combination, as well as the auxiliary signs and patterns of the preceding brands, or anything similar to the company names, trade names, trademarks, product or service names, domain names, patterns, logos, marks, signs, or special descriptions that third parties identify as Alibaba Cloud and/or its affiliates.

6. Please directly contact Alibaba Cloud for any errors of this document.

Document conventions

Style	Description	Example
Danger	A danger notice indicates a situation that will cause major system changes, faults, physical injuries, and other adverse results.	**Danger:** Resetting will result in the loss of user configuration data.
Warning	A warning notice indicates a situation that may cause major system changes, faults, physical injuries, and other adverse results.	**Warning:** Restarting will cause business interruption. About 10 minutes are required to restart an instance.
Notice	A caution notice indicates warning information, supplementary instructions, and other content that the user must understand.	**Notice:** If the weight is set to 0, the server no longer receives new requests.
Note	A note indicates supplemental instructions, best practices, tips, and other content.	**Note:** You can use Ctrl + A to select all files.
>	Closing angle brackets are used to indicate a multi-level menu cascade.	Click **Settings**> **Network**> **Set network type**.
Bold	Bold formatting is used for buttons , menus, page names, and other UI elements.	Click **OK**.
`Courier font`	Courier font is used for commands	Run the `cd /d C:/window` command to enter the Windows system folder.
Italic	Italic formatting is used for parameters and variables.	`bae log list --instanceid` *Instance_ID*
[] or [a|b]	This format is used for an optional value, where only one item can be selected.	`ipconfig [-all|-t]`
{} or {a|b}	This format is used for a required value, where only one item can be selected.	`switch {active|stand}`

Table of Contents

1.Overview 27
2.Models 28
2.1. Geometry model 28
2.2. Raster model 32
2.3. Path model 33
2.4. Point cloud model 36
2.5. Trajectory model 39
2.6. Grid model 41
2.7. Vector pyramid 43
3.Advanced usage 47
3.1. Create spatial indexes in parallel 47
3.2. Enable spatio-temporal two-phase query optimization 47
3.3. Enable spatio-temporal parallel query 48
3.4. Enable GPU-accelerated computing 50
3.5. Spatial-temporal object storage optimization 52
3.5.1. Feature signature-based storage optimization for large ... 52
3.5.2. ST_SetTypeStorage 53
3.5.3. Use the Simple storage policy 53
4.Geometry SQL reference 55
4.1. Output functions 55
4.1.1. ST_AsBinary 55
4.1.2. ST_AsEncodedPolyline 56
4.1.3. ST_AsEWKB 56
4.1.4. ST_AsEWKT 57
4.1.5. ST_AsGeoJSON 58
4.1.6. ST_AsGML 59

4.1.7. ST_AsHEXEWKB 61
4.1.8. ST_AsKML 62
4.1.9. ST_AsLatLonText 63
4.1.10. ST_AsSVG 64
4.1.11. ST_AsText 65
4.1.12. ST_AsTWKB 66
4.1.13. ST_AsX3D 67
4.1.14. ST_GeoHash 68
4.1.15. ST_AsGeobuf 69
4.2. Editor functions 69
4.2.1. ST_AddPoint 69
4.2.2. ST_Affine 70
4.2.3. ST_Force2D 72
4.2.4. ST_Force3D 73
4.2.5. ST_Force3DZ 73
4.2.6. ST_Force3DM 74
4.2.7. ST_Force4D 75
4.2.8. ST_ForcePolygonCCW 75
4.2.9. ST_ForceCollection 76
4.2.10. ST_ForcePolygonCW 77
4.2.11. ST_ForceSFS 77
4.2.12. ST_ForceRHR 78
4.2.13. ST_ForceCurve 79
4.2.14. ST_LineMerge 80
4.2.15. ST_CollectionExtract 80
4.2.16. ST_CollectionHomogenize 81
4.2.17. ST_Multi 82
4.2.18. ST_Normalize 82

4.2.19. ST_QuantizeCoordinates 83
4.2.20. ST_RemovePoint 84
4.2.21. ST_Reverse 85
4.2.22. ST_Rotate 86
4.2.23. ST_RotateX 88
4.2.24. ST_RotateY 88
4.2.25. ST_RotateZ 89
4.2.26. ST_Scale 90
4.2.27. ST_Segmentize 91
4.2.28. ST_SetPoint 92
4.2.29. ST_SetSRID 93
4.2.30. ST_Transform 93
4.2.31. ST_Translate 94
4.2.32. ST_TransScale 95
4.3. Geometry constructors 96
4.3.1. ST_BdPolyFromText 96
4.3.2. ST_BdMPolyFromText 97
4.3.3. ST_Box2dFromGeoHash 98
4.3.4. ST_GeogFromText 98
4.3.5. ST_GeographyFromText 99
4.3.6. ST_GeogFromWKB 100
4.3.7. ST_GeomFromTWKB 101
4.3.8. ST_GeomCollFromText 101
4.3.9. ST_GeomFromEWKB 102
4.3.10. ST_GeomFromEWKT 103
4.3.11. ST_GeometryFromText 103
4.3.12. ST_GeomFromText 104
4.3.13. ST_GeomFromGeoHash 105

4.3.14. ST_GeomFromGML 106
4.3.15. ST_GeomFromGeoJSON 107
4.3.16. ST_GeomFromKML 107
4.3.17. ST_GMLToSQL 108
4.3.18. ST_GeomFromWKB 109
4.3.19. ST_LineFromEncodedPolyline 109
4.3.20. ST_LineFromMultiPoint 110
4.3.21. ST_LineFromText 111
4.3.22. ST_LineFromWKB 111
4.3.23. ST_LinestringFromWKB 112
4.3.24. ST_MakeBox2D 113
4.3.25. ST_3DMakeBox 113
4.3.26. ST_MakeLine 114
4.3.27. ST_MakeEnvelope 115
4.3.28. ST_MakePolygon 116
4.3.29. ST_MakePoint 117
4.3.30. ST_MakePointM 118
4.3.31. ST_MPointFromText 119
4.3.32. ST_MLineFromText 119
4.3.33. ST_MPolyFromText 120
4.3.34. ST_Point 121
4.3.35. ST_PointFromGeoHash 121
4.3.36. ST_PointFromText 122
4.3.37. ST_PointFromWKB 123
4.3.38. ST_Polygon 124
4.3.39. ST_PolygonFromText 125
4.3.40. ST_WKBToSQL 125
4.3.41. ST_WKTToSQL 126

4.4. Access functions 127
4.4.1. GeometryType 127
4.4.2. ST_Boundary 128
4.4.3. ST_CoordDim 129
4.4.4. ST_Dimension 130
4.4.5. ST_EndPoint 131
4.4.6. ST_Envelope 131
4.4.7. ST_BoundingDiagonal 132
4.4.8. ST_ExteriorRing 133
4.4.9. ST_GeometryN 134
4.4.10. ST_GeometryType 135
4.4.11. ST_InteriorRingN 136
4.4.12. ST_IsPolygonCCW 137
4.4.13. ST_IsPolygonCW 137
4.4.14. ST_IsClosed 138
4.4.15. ST_IsCollection 139
4.4.16. ST_IsEmpty 140
4.4.17. ST_IsRing 141
4.4.18. ST_IsSimple 141
4.4.19. ST_IsValid 142
4.4.20. ST_IsValidReason 143
4.4.21. ST_IsValidDetail 144
4.4.22. ST_M 144
4.4.23. ST_NDims 145
4.4.24. ST_NPoints 146
4.4.25. ST_NRings 147
4.4.26. ST_NumGeometries 147
4.4.27. ST_NumInteriorRings 148

4.4.28. ST_NumInteriorRing 149
4.4.29. ST_NumPatches 149
4.4.30. ST_NumPoints 150
4.4.31. ST_PatchN 151
4.4.32. ST_PointN 151
4.4.33. ST_Points 152
4.4.34. ST_SRID 153
4.4.35. ST_StartPoint 154
4.4.36. ST_Summary 154
4.4.37. ST_X 155
4.4.38. ST_XMax 156
4.4.39. ST_XMin 157
4.4.40. ST_Y 157
4.4.41. ST_YMax 158
4.4.42. ST_YMin 159
4.4.43. ST_Z 160
4.4.44. ST_ZMax 160
4.4.45. ST_Zmflag 161
4.4.46. ST_ZMin 162
4.5. Functions for spatial relationships and measurements 163
4.5.1. ST_3DClosestPoint 163
4.5.2. ST_3DDistance 163
4.5.3. ST_3DDWithin 164
4.5.4. ST_3DDFullyWithin 165
4.5.5. ST_3DIntersects 166
4.5.6. ST_3DLongestLine 167
4.5.7. ST_3DMaxDistance 167
4.5.8. ST_3DShortestLine 168

4.5.9. ST_Area 169
4.5.10. ST_Azimuth 170
4.5.11. ST_Angle 171
4.5.12. ST_Centroid 172
4.5.13. ST_ClosestPoint 173
4.5.14. ST_Contains 173
4.5.15. ST_ContainsProperly 174
4.5.16. ST_Covers 175
4.5.17. ST_CoveredBy 176
4.5.18. ST_Crosses 177
4.5.19. ST_LineCrossingDirection 179
4.5.20. ST_Disjoint 180
4.5.21. ST_Distance 180
4.5.22. ST_MinimumClearance 181
4.5.23. ST_MinimumClearanceLine 182
4.5.24. ST_HausdorffDistance 183
4.5.25. ST_FrechetDistance 183
4.5.26. ST_MaxDistance 184
4.5.27. ST_DistanceSphere 185
4.5.28. ST_DistanceSpheroid 186
4.5.29. ST_DFullyWithin 187
4.5.30. ST_DWithin 188
4.5.31. ST_Equals 189
4.5.32. ST_GeometricMedian 190
4.5.33. ST_HasArc 191
4.5.34. ST_Intersects 192
4.5.35. ST_Length 193
4.5.36. ST_Length2D 194

4.5.37. ST_3DLength 195
4.5.38. ST_LengthSpheroid 195
4.5.39. ST_Length2DSpheroid 196
4.5.40. ST_LongestLine 197
4.5.41. ST_OrderingEquals 197
4.5.42. ST_Overlaps 198
4.5.43. ST_Perimeter 198
4.5.44. ST_Perimeter2D 199
4.5.45. ST_3DPerimeter 200
4.5.46. ST_PointOnSurface 200
4.5.47. ST_Project 201
4.5.48. ST_Relate 202
4.5.49. ST_RelateMatch 203
4.5.50. ST_ShortestLine 204
4.5.51. ST_Touches 204
4.5.52. ST_Within 205
4.6. Processing functions 206
4.6.1. ST_Buffer 206
4.6.2. ST_BuildArea 210
4.6.3. ST_ClipByBox2D 211
4.6.4. ST_Collect 212
4.6.5. ST_ConcaveHull 213
4.6.6. ST_ConvexHull 215
4.6.7. ST_CurveToLine 216
4.6.8. ST_DelaunayTriangles 218
4.6.9. ST_Different 219
4.6.10. ST_Dump 220
4.6.11. ST_DumpPoints 221

4.6.12. ST_DumpRings 222
4.6.13. ST_FlipCoordinates 223
4.6.14. ST_GeneratePoints 223
4.6.15. ST_Intersection 224
4.6.16. ST_LineToCurve 225
4.6.17. ST_MakeValid 225
4.6.18. ST_MemUnion 226
4.6.19. ST_MinimumBoundingCircle 227
4.6.20. ST_MinimumBoundingRadius 228
4.6.21. ST_OrientedEnvelope 228
4.6.22. ST_Polygonize 229
4.6.23. ST_OffsetCurve 230
4.6.24. ST_RemoveRepeatedPoints 232
4.6.25. ST_SharedPaths 233
4.6.26. ST_ShiftLongitude 234
4.6.27. ST_Simplify 234
4.6.28. ST_SimplifyPreserveTopology 236
4.6.29. ST_SimplifyVW 237
4.6.30. ST_ChaikinSmoothing 238
4.6.31. ST_FilterByM 240
4.6.32. ST_SetEffectiveArea 241
4.6.33. ST_Split 242
4.6.34. ST_SymDifference 243
4.6.35. ST_Subdivide 244
4.6.36. ST_SwapOrdinates 245
4.6.37. ST_Union 246
4.6.38. ST_VoronoiLines 247
4.6.39. ST_VoronoiPolygons 248

4.7. Aggregate functions 250
4.7.1. ST_ClusterDBSCAN 250
4.7.2. ST_ClusterIntersecting 251
4.7.3. ST_ClusterKMeans 251
4.7.4. ST_ClusterWithin 252
4.8. Linear referencing functions 253
4.8.1. ST_LineInterpolatePoint 253
4.8.2. ST_LineInterpolatePoints 254
4.8.3. ST_LineLocatePoint 255
4.8.4. ST_LineSubstring 256
4.8.5. ST_LocateAlong 257
4.8.6. ST_LocateBetween 257
4.8.7. ST_LocateBetweenElevations 258
4.8.8. ST_InterpolatePoint 259
4.8.9. ST_AddMeasure 260
5.Raster SQL reference 261
5.1. Basic concepts 261
5.2. Parallel operations 261
5.3. Object storage paths 263
5.4. Raster creation 264
5.4.1. ST_CreateRast 264
5.5. Import and export 265
5.5.1. ST_ImportFrom 265
5.5.2. ST_ExportTo 268
5.5.3. ST_AsImage 269
5.5.4. ST_AsPNG 272
5.5.5. ST_AsJPEG 275
5.5.6. ST_AsDatasetFile 277

5.6. Pyramid operations 279
5.6.1. ST_BuildPyramid 279
5.6.2. ST_deletePyramid 282
5.6.3. ST_BestPyramidLevel 282
5.7. Coordinate system conversion 283
5.7.1. ST_Rast2WorldCoord 283
5.7.2. ST_World2RastCoord 284
5.8. Pixel value operations 285
5.8.1. ST_AddZ 285
5.8.2. ST_ClipDimension 286
5.8.3. ST_Clip 286
5.8.4. ST_ClipToRast 289
5.8.5. ST_Values 291
5.8.6. ST_PointValues 293
5.8.7. ST_Update 294
5.8.8. ST_MosaicFrom 295
5.8.9. ST_MosaicTo 296
5.8.10. ST_PixelAsPolygon 297
5.8.11. ST_PixelAsPolygons 298
5.8.12. ST_PixelAsPoint 298
5.8.13. ST_PixelAsCentroid 299
5.8.14. ST_PixelAsPoints 300
5.8.15. ST_PixelAsCentroids 301
5.9. Overview operations 302
5.9.1. ST_BuildOverview 303
5.9.2. ST_UpdateOverview 303
5.9.3. ST_EraseOverview 304
5.10. DEM operations 305

5.10.1. ST_Aspect 305
5.10.2. ST_Slope 306
5.10.3. ST_Hillshade 306
5.10.4. ST_Overflow 308
5.10.5. ST_Flow_direction 309
5.11. Attribute query and update 310
5.11.1. ST_Name 310
5.11.2. ST_SetName 310
5.11.3. ST_MetaData 311
5.11.4. ST_Width 312
5.11.5. ST_Height 313
5.11.6. ST_NumBands 313
5.11.7. ST_Value 314
5.11.8. ST_RasterID 314
5.11.9. ST_CellDepth 315
5.11.10. ST_CellType 315
5.11.11. ST_InterleavingType 316
5.11.12. ST_TopPyramidLevel 316
5.11.13. ST_Extent 316
5.11.14. ST_ConvexHull 317
5.11.15. ST_Envelope 318
5.11.16. ST_Srid 319
5.11.17. ST_SetSrid 320
5.11.18. ST_ScaleX 320
5.11.19. ST_ScaleY 321
5.11.20. ST_SetScale 321
5.11.21. ST_SkewX 322
5.11.22. ST_SkewY 323

5.11.23. ST_SetSkew 323
5.11.24. ST_UpperLeftX 324
5.11.25. ST_UpperLeftY 324
5.11.26. ST_SetUpperLeft 325
5.11.27. ST_PixelWidth 326
5.11.28. ST_PixelHeight 326
5.11.29. ST_Georeference 327
5.11.30. ST_IsGeoreferenced 328
5.11.31. ST_UnGeoreference 328
5.11.32. ST_SetGeoreference 329
5.11.33. ST_RPCGeoreference 329
5.11.34. ST_SetRPCGeoreference 331
5.11.35. ST_NoData 335
5.11.36. ST_SetNoData 335
5.11.37. ST_ColorTable 336
5.11.38. ST_SetColorTable 337
5.11.39. ST_Statistics 337
5.11.40. ST_SetStatistics 338
5.11.41. ST_SummaryStats 338
5.11.42. ST_ColorInterp 339
5.11.43. ST_SetColorInterp 341
5.11.44. ST_Histogram 342
5.11.45. ST_SetHistogram 343
5.11.46. ST_BuildHistogram 346
5.11.47. ST_StatsQuantile 346
5.11.48. ST_Quantile 347
5.11.49. ST_MD5Sum 348
5.11.50. ST_SetMD5Sum 349

5.11.51. ST_XMin 350
5.11.52. ST_YMin 350
5.11.53. ST_XMax 351
5.11.54. ST_YMax 352
5.11.55. ST_ChunkHeight 352
5.11.56. ST_ChunkWidth 353
5.11.57. ST_ChunkBands 353
5.11.58. ST_MetaItems 354
5.11.59. ST_SetMetaData 354
5.11.60. ST_BeginDateTime 355
5.11.61. ST_EndDateTime 356
5.11.62. ST_SetBeginDateTime 356
5.11.63. ST_SetEndDateTime 357
5.11.64. ST_DateTime 357
5.11.65. ST_SetDateTime 358
5.11.66. ST_BuildPercentiles 359
5.11.67. ST_ComputeStatistics 361
5.11.68. ST_Percentile 362
5.12. Algebra and Analysis Functions 363
5.12.1. ST_Reclassify 363
5.12.2. ST_MapAlgebra 368
5.13. Raster Image Processing 373
5.13.1. ST_SubRaster 373
5.13.2. ST_Transform 376
5.13.3. ST_Rescale 380
5.13.4. ST_Resize 384
5.13.5. ST_RPCRectify 388
5.13.6. ST_LinearStretch 393

5.14. Operators 397
5.14.1. Equal to operator (=) 397
5.14.2. Greater than operator (>) 398
5.14.3. Less than operator (<) 398
5.14.4. Greater than or equal to operator (>=) 399
5.14.5. Less than or equal to operator (<=) 400
5.15. Functions for Identifying Spatial Relationships 400
5.15.1. ST_Intersects 400
5.15.2. ST_Contains 401
5.15.3. ST_ContainsProperly 402
5.15.4. ST_Covers 402
5.15.5. ST_CoveredBy 403
5.15.6. ST_Disjoint 403
5.15.7. ST_overlaps 404
5.15.8. ST_Touches 405
5.15.9. ST_Within 405
5.16. Auxiliary functions 406
5.16.1. ST_CreateChunkTable 406
5.16.2. ST_CheckGPU 407
5.16.3. ST_AKId 407
5.16.4. ST_SetAccessKey 408
5.16.5. ST_SetAKId 408
5.16.6. ST_SetAKSecret 409
5.16.7. ST_RasterDrivers 410
5.17. Variables 412
5.17.1. ganos.parallel.transaction 412
5.17.2. ganos.parallel.degree 413
5.17.3. ganos.raster.calculate_md5 413

5.17.4. ganos.raster.md5sum_chunk_size 414
5.17.5. ganos.raster.mosaic_must_same_nodata 414
5.17.6. ganos.raster.memory_oss_file_max_size 414
6.SpatialRef SQL reference 416
6.1. ST_SrEqual 416
6.2. ST_SrReg 417
6.3. ST_SrFromEsriWkt 418
7.Point cloud SQL reference 419
7.1. Constructors 419
7.1.1. ST_makePoint 419
7.1.2. ST_makePatch 419
7.1.3. ST_Patch 420
7.2. Attribute functions 420
7.2.1. ST_asText 420
7.2.2. ST_pcID 421
7.2.3. ST_get 421
7.2.4. ST_numPoints 422
7.2.5. ST_summary 422
7.3. Object operations 423
7.3.1. ST_boundingDiagonalGeometry 423
7.3.2. ST_compress 424
7.3.3. ST_explode 424
7.3.4. ST_patchAvg 425
7.3.5. ST_envelopeGeometry 426
7.3.6. ST_filterGreaterThan 426
7.3.7. ST_filterLessThan 427
7.3.8. ST_filterEquals 427
7.3.9. ST_filterBetween 428

7.3.10. ST_isSorted 428
7.3.11. ST_patchMax 429
7.3.12. ST_patchMin 429
7.3.13. ST_patchAvg 430
7.3.14. ST_patchMax 430
7.3.15. ST_patchMin 431
7.3.16. ST_pointN 431
7.3.17. ST_sort 432
7.3.18. ST_range 432
7.3.19. ST_setPcid 433
7.3.20. ST_transform 434
7.3.21. ST_unCompress 434
7.3.22. ST_union 435
7.4. OGC WKB operations 435
7.4.1. ST_asBinary 435
7.4.2. ST_envelopeAsBinary 436
7.4.3. ST_boundingDiagonalAsBinary 436
7.5. Spatial relationship judgment 437
7.5.1. ST_intersects 437
7.6. Spatial processing 437
7.6.1. ST_intersection 438
8.Trajectory SQL reference 439
8.1. Terms 439
8.2. Constructors 440
8.2.1. Overview 440
8.2.2. ST_append 441
8.2.3. ST_makeTrajectory 442
8.3. Editing and processing functions 447

8.3.1. ST_attrDeduplicate 447
8.3.2. ST_Compress 448
8.3.3. ST_CompressSED 452
8.3.4. ST_deviation 453
8.3.5. ST_sort 453
8.3.6. ST_removeDriftPoints 454
8.3.7. ST_Split 455
8.4. Attribute metadata 460
8.4.1. ST_attrDefinition 460
8.4.2. ST_attrSize 461
8.4.3. ST_attrName 462
8.4.4. ST_attrType 463
8.4.5. ST_attrLength 464
8.4.6. ST_attrNullable 465
8.5. Event functions 466
8.5.1. ST_addEvent 466
8.5.2. ST_eventTimes 467
8.5.3. ST_eventTime 468
8.5.4. ST_eventTypes 469
8.5.5. ST_eventType 469
8.6. Attribute functions 470
8.6.1. ST_startTime 470
8.6.2. ST_endTime 471
8.6.3. ST_trajectorySpatial 471
8.6.4. ST_trajectoryTemporal 471
8.6.5. ST_trajAttrs 472
8.6.6. ST_attrIntMax 472
8.6.7. ST_attrIntMin 473

8.6.8. ST_attrIntAverage 474
8.6.9. ST_attrFloatMax 474
8.6.10. ST_attrFloatMin 475
8.6.11. ST_attrFloatAverage 476
8.6.12. ST_leafType 477
8.6.13. ST_leafCount 477
8.6.14. ST_duration 477
8.6.15. ST_timeAtPoint 478
8.6.16. ST_pointAtTime 478
8.6.17. ST_velocityAtTime 479
8.6.18. ST_accelerationAtTime 479
8.6.19. ST_timeToDistance 480
8.6.20. ST_timeAtDistance 480
8.6.21. ST_cumulativeDistanceAtTime 481
8.6.22. ST_timeAtCumulativeDistance 481
8.6.23. ST_subTrajectory 482
8.6.24. ST_subTrajectorySpatial 482
8.6.25. ST_samplingInterval 483
8.6.26. ST_trajAttrsAsText 483
8.6.27. ST_trajAttrsAsInteger 484
8.6.28. ST_trajAttrsAsDouble 485
8.6.29. ST_trajAttrsAsBool 486
8.6.30. ST_trajAttrsAsTimestamp 486
8.6.31. ST_attrIntFilter 487
8.6.32. ST_attrFloatFilter 489
8.6.33. ST_attrTimestampFilter 490
8.6.34. ST_attrNullFilter 491
8.6.35. ST_attrNotNullFilter 492

8.6.36. ST_trajAttrsMeanMax 493
8.7. Functions to process bounding boxes 495
8.7.1. ST_MakeBox 495
8.7.2. ST_MakeBox{Z|T|2D|2DT|3D|3DT} 496
8.7.3. ST_BoxndfToGeom 498
8.7.4. ST_Has{xy|z|t} 498
8.7.5. ST_{X|Y|Z|T}Min 499
8.7.6. ST_{X|Y|Z|T}Max 500
8.7.7. ST_ExpandSpatial 501
8.8. Operators to process bounding boxes 501
8.8.1. Basic concepts 501
8.8.2. INTERSECT operators 502
8.8.3. INCLUDE operators 503
8.8.4. INCLUDED operators 505
8.9. Spatial relationship judgment 506
8.9.1. ST_intersects 506
8.9.2. ST_equals 507
8.9.3. ST_distanceWithin 507
8.10. Spatial processing 508
8.10.1. ST_intersection 508
8.10.2. ST_difference 509
8.11. Spatial statistics 509
8.11.1. ST_nearestApproachPoint 509
8.11.2. ST_nearestApproachDistance 510
8.12. Spatio-temporal relationship judgment 510
8.12.1. ST_intersects 510
8.12.2. ST_equals 511
8.12.3. ST_distanceWithin 512

8.12.4. ST_durationWithin 512
8.12.5. ST_{Z|T|2D|2DT|3D|3DT}Intersects 513
8.12.6. ST_{2D|2DT|3D|3DT}Intersects_IndexLeft 515
8.12.7. ST_{2D|2DT|3D|3DT}DWithin 516
8.12.8. ST_{2D|2DT|3D|3DT}DWithin_IndexLeft 519
8.12.9. ST_{T|2D|2DT|3D|3DT}Contains 521
8.12.10. ST_{T|2D|2DT|3D|3DT}Within 523
8.13. Spatio-temporal processing 524
8.13.1. ST_intersection 524
8.14. Spatio-temporal statistics 525
8.14.1. ST_nearestApproachPoint 525
8.14.2. ST_nearestApproachDistance 526
8.15. Distance measurement 526
8.15.1. ST_length 526
8.15.2. ST_euclideanDistance 527
8.15.3. ST_mdistance 528
8.16. Similarity analysis 528
8.16.1. ST_lcsSimilarity 528
8.16.2. ST_lcsDistance 530
8.16.3. ST_lcsSubDistance 533
8.16.4. ST_JaccardSimilarity 534
8.17. Indexing 538
8.17.1. GiST indexing 538
8.17.2. TrajGiST indexing 540
8.18. External storage 541
8.18.1. ST_ExportTo 541
8.18.2. ST_IsExternal 542
8.18.3. ST_importFrom 543

8.18.4. ST_StorageLocation 543
8.18.5. ST_AKID 544
8.18.6. ST_SetAccessKey 545
8.18.7. ST_SetAkId 546
8.18.8. ST_SetAkSecret 547
8.18.9. ST_SetStorageLocation 548
8.18.10. ST_DeleteGTF 549
8.19. Variables 550
8.19.1. ganos.trajectory.attr_string_length 550
8.19.2. ganos.trajectory.index_split_config 550
9.GeomGrid SQL reference 551
9.1. Usage notes 551
9.2. Functions for output 551
9.2.1. ST_AsText 551
9.2.2. ST_AsBinary 552
9.2.3. ST_AsGeometry 552
9.2.4. ST_AsBox 553
9.3. Functions for input 553
9.3.1. ST_GridFromText 554
9.3.2. ST_GridFromBinary 554
9.4. Functions to query spatial relationships 555
9.4.1. ST_Intersects 555
9.4.2. ST_Contains 555
9.4.3. ST_Within 556
9.5. Operators 557
9.5.1. @> 557
9.5.2. <@ 558
9.6. Functions to compute grids 558

9.6.1. ST_AsGrid 558
10.Geometry Pyramid SQL reference 562
10.1. Usage notes 562
10.2. Functions to build pyramids 562
10.2.1. ST_BuildPyramid 562
10.3. Functions to delete pyramids 564
10.3.1. ST_DeletePyramid 564
10.4. Functions to view pyramids 565
10.4.1. ST_Tile 565
10.4.2. ST_AsPng 566
11.Map services 568
11.1. Overview 568
11.2. Publish geometry data 568
11.3. Publish raster data 568
12.Desktop applications 575
12.1. Use OpenJump to access Ganos data 575
12.2. Use QGIS to access Ganos data 575
12.3. Use uDig to access Ganos data 578
13.FAQ 583
13.1. Load raster data 583
13.2. Load vector data 583
13.3. Trajectory FAQ 588
13.4. How do I update the Ganos plug-ins? 592
14.Best practice 594
14.1. Trajectory best practices 594

1.Overview

Spatio-temporal data is graphic and image data that contains both space and time information. Spatio-temporal data consists of multidimensional information about objects, such as the location, shape, change, and size distribution.

Description

Ganos is a spatio-temporal engine developed by Alibaba Cloud. Ganos supports a series of data types, functions, and stored procedures for Apsara PolarDB to store, index, query, analyze, and compute spatio-temporal data in a cost-effective manner.

This topic describes how to use Ganos in Apsara PolarDB to manage and analyze spatio-temporal data.

To request further assistance, you can log on to the Apsara PolarDB console. In the top navigation bar, choose **Ticket > New Ticket**.

Pricing

The Ganos engine is supported by . Ganos is provided free of charge by Alibaba Cloud.

2.Models

2.1. Geometry model

Ganos Geometry is a spatial geometry extension of ApsaraDB for PolarDB. Ganos Geometry complies with OpenGIS specifications and enables ApsaraDB for PolarDB to store and manage 2D (X, Y), 3D (X, Y, Z), and 4D (X, Y, Z, M) spatial geometry data. In addition, Ganos Geometry provides a wide range of features such as spatial geometry objects, indexes, functions, and operators.

Overview

The geometry model is fully compatible with PostGIS functions and allows you to smoothly migrate existing applications.

Quick start

- Create extensions.

```
-- Create a geometry extension.
Create extension ganos_geometry cascade;
-- Create a geometry topology extension.
Create extension ganos_geometry_topology;
-- Create an SFCGAL plug-in extension.
Create extension ganos_geometry_sfcgal;
```

- Create a geometry table.

```
-- Method 1: Create a table with a geometry field.
CREATE TABLE ROADS (ID int4, ROAD_NAME varchar(25), geom geometry(LINESTRING,3857));
-- Method 2: Create a common table and then add a geometry field.
CREATE TABLE ROADS (ID int4, ROAD_NAME varchar(25));
SELECT AddGeometryColumn('roads', 'geom', 3857, 'LINESTRING', 2);
```

- Add geometry constraints.

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

- Import geometry data.

```
INSERT INTO roads (id, geom, road_name)
  VALUES (1,ST_GeomFromText('LINESTRING(191232 243118,191108 243242)',3857),'North Fifth-
Ring Road');
INSERT INTO roads (id, geom, road_name)
  VALUES (2,ST_GeomFromText('LINESTRING(189141 244158,189265 244817)',3857),'East Fifth-R
ing Road');
INSERT INTO roads (id, geom, road_name)
  VALUES (3,ST_GeomFromText('LINESTRING(192783 228138,192612 229814)',3857),'South Fifth-
Ring Road');
INSERT INTO roads (id, geom, road_name)
  VALUES (4,ST_GeomFromText('LINESTRING(189412 252431,189631 259122)',3857),'West Fifth-R
ing Road');
INSERT INTO roads (id, geom, road_name)
  VALUES (5,ST_GeomFromText('LINESTRING(190131 224148,190871 228134)',3857),'East Chang''
an Avenue');
INSERT INTO roads (id, geom, road_name)
  VALUES (6,ST_GeomFromText('LINESTRING(198231 263418,198213 268322)',3857),'West Chang''
an Avenue');
```

- Query geometry object information.

```
SELECT id, ST_AsText(geom) AS geom, road_name FROM roads;
--------------------------------
     id | geom                                     | road_name
--------+------------------------------------------+-----------
      1 | LINESTRING(191232 243118,191108 243242) | North Fifth-Ring Road
      2 | LINESTRING(189141 244158,189265 244817) | East Fifth-Ring Road
      3 | LINESTRING(192783 228138,192612 229814) | South Fifth-Ring Road
      4 | LINESTRING(189412 252431,189631 259122) | West Fifth-Ring Road
      5 | LINESTRING(190131 224148,190871 228134) | East Chang'an Avenue
      6 | LINESTRING(198231 263418,198213 268322) | West Chang'an Avenue
(6 rows)
```

- Create indexes.

```
-- Create a GiST index.
CREATE INDEX [indexname] ON [tablename] USING GIST ([geometryfield]);
CREATE INDEX [indexname] ON [tablename] USING GIST ([geometryfield] gist_geometry_ops_nd)
;
VACUUM ANALYZE [table_name] [(column_name)];
-- Example
Create INDEX sp_geom_index ON ROADS USING GIST(geom);
VACUUM ANALYZE ROADS (geom);
-- Create a block range index (BRIN).
CREATE INDEX [indexname] ON [tablename] USING BRIN ([geometryfield]);
CREATE INDEX [indexname] ON [tablename] USING BRIN ([geometryfield] brin_geometry_inclusi
on_ops_3d);
CREATE INDEX [indexname] ON [tablename] USING BRIN ([geometryfield] brin_geometry_inclusi
on_ops_4d);
-- Create a BRIN index with a specified range.
CREATE INDEX [indexname] ON [tablename] USING BRIN ([geometryfield]) WITH (pages_per_rang
e = [number]);
```

- Access geometry objects.

```
-- Determine whether a spatial geometry object consists of only simple elements.
 SELECT ST_IsSimple(ST_GeomFromText('POLYGON((1 2, 3 4, 5 6, 1 2))'));
 st_issimple
-------------
 t
(1 row)
 SELECT ST_IsSimple(ST_GeomFromText('LINESTRING(1 1,2 2,2 3.5,1 3,1 2,2 1)'));
 st_issimple
-------------
 f
(1 row)
-- Query the largest city with roundabouts in the terrain.
SELECT gid, name, ST_Area(the_geom) AS area
FROM bc_municipality
WHERE ST_NRings(the_geom) > 1
ORDER BY area DESC LIMIT 1;
gid  | name         | area
-----+--------------+------------------
12   | Anning        | 257374619.430216
(1 row)
```

- Measure and analyze spatial data, and judge spatial relationships.

```
-- Create the bc_roads table.
Create table bc_roads (gid serial, name varchar, the_geom geometry);
-- Create the bc_municipality table.
Create table bc_municipality(gid serial, code integer, name varchar, the_geom geometry);
-- Compute the length.
SELECT sum(ST_Length(the_geom))/1000 AS km_roads FROM bc_roads;
km_roads
------------------
70842.1243039643
(1 row)
-- Compute the area.
SELECT ST_Area(the_geom)/10000 AS hectares FROM bc_municipality WHERE name = 'PRINCE GEOR
GE';
hectares
------------------
32657.9103824927
(1 row)
-- Use the ST_Contains function.
SELECT m.name, sum(ST_Length(r.the_geom))/1000 as roads_km
FROM
  bc_roads AS r, bc_municipality AS m
WHERE
  ST_Contains(m.the_geom,r.the_geom)
GROUP BY m.name
ORDER BY roads_km;
name                        | roads_km
----------------------------+------------------
SURREY                      | 1539.47553551242
VANCOUVER                   | 1450.33093486576
LANGLEY DISTRICT            | 833.793392535662
BURNABY                     | 773.769091404338
```

```
PRINCE GEORGE               | 694.37554369147
-- Use the ST_Covers function.
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)
-- Use the ST_Disjoint function.
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)
-- Use the ST_Overlaps function.
SELECT ST_Overlaps(a,b) As a_overlap_b,
    ST_Crosses(a,b) As a_crosses_b,
        ST_Intersects(a, b) As a_intersects_b, ST_Contains(b,a) As b_contains_a
FROM (SELECT ST_GeomFromText('POINT(1 0.5)') As a, ST_GeomFromText('LINESTRING(1 0, 1 1,
3 5)') As b)
    As foo
a_overlap_b | a_crosses_b | a_intersects_b | b_contains_a
------------+-------------+----------------+--------------
f           | f           | t              | t
-- Use the ST_Relate function.
SELECT ST_Relate(ST_GeometryFromText('POINT(1 2)'), ST_Buffer(ST_GeometryFromText('POINT(
1 2)'),2), '0FFFFF212');
St_relate
-----------
t
-- Use the ST_Touches function.
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)
-- Use the ST_Within function.
SELECT ST_Within(smallc,smallc) As smallinsmall,
    ST_Within(smallc, bigc) As smallinbig,
```

```
    ST_Within(bigc,smallc) As biginsmall,
    ST_Within(ST_Union(smallc, bigc), bigc) as unioninbig,
    ST_Within(bigc, ST_Union(smallc, bigc)) as biginunion,
    ST_Equals(bigc, ST_Union(smallc, bigc)) as bigisunion
FROM
(
SELECT ST_Buffer(ST_GeomFromText('POINT(50 50)'), 20) As smallc,
    ST_Buffer(ST_GeomFromText('POINT(50 50)'), 40) As bigc) As foo;
-- Result
 smallinsmall | smallinbig | biginsmall | unioninbig | biginunion | bigisunion
--------------+------------+------------+------------+------------+------------
 t            | t          | f          | t          | t          | t
(1 row)
```

- Delete extensions.

```
Drop extension ganos_geometry cascade;
```

SQL reference

For more information, see the official PostGIS reference.

2.2. Raster model

Raster data consists of a matrix of cells or pixels. The cells or pixels are organized into rows and columns to form a grid. Each cell contains a value that represents information, such as temperature.

Overview

Raster data can be digital aerial photographs, satellite images, digital images, or scanned maps.

implement the raster model. This allows Ganos to store and analyze raster data in a cost-effective manner with the help of database technologies and methods.

Quick start

- Create an extension.

```
Create Extension Ganos_Raster cascade;
```

- Create a raster table.

```
Create Table raster_table(id integer, raster_obj raster);
```

- Import raster data from Object Storage Service (OSS).

```
Insert into raster_table Values(1, ST_ImportFrom('chunk_table','OSS://ABCDEFG:1234567890@
oss-cn.aliyuncs.com/mybucket/data/4.tif'))
```

- Query raster object information.

```
Select ST_Height(raster_obj),ST_Width(raster_obj) From raster_table Where id = 1;
```

- Create a pyramid.

```
Update raster_table Set raster_obj = ST_BuildPyramid(raster_obj) Where id = 1;
```

- Calculate the optimal pyramid level based on the world space, width, and height of a viewport.

```
Select ST_BestPyramidLevel(raster_obj, '((128.0, 30.0),(128.5, 30.5))', 800, 600) from ra
ster_table where id = 10;
---------------------
3
```

- Retrieve a pixel matrix from the specified space of a raster object.

```
Select ST_Clip(raster_obj, 0, '((128.980,30.0),(129.0,30.2))', 'World') From raster_table
Where id = 1;
```

- Calculate the raster space of a clipped area.

```
Select ST_ClipDimension(raster_obj, 2, '((128.0, 30.0),(128.5, 30.5))') from raster_table
where id = 10;
-----------------------------------
'((600, 720),(200, 300))'
```

- Delete the extension.

```
Drop Extension Ganos_raster cascade;
```

SQL reference

For more information, see Raster SQL reference.

2.3. Path model

Path data is a geometric network diagram that consists of edges and nodes. It is used to build road and traffic networks.

Overview

Ganos Networking is an extension of ApsaraDB for PolarDB. Ganos Networking provides a series of functions and stored procedures to find the fastest, shortest, and optimal paths based on cost models. If the cost is time, the fastest path is the optimal path. If the cost is distance, the shortest path is the optimal path. The path model can be used for path planning on a road network, path search and planning in GPS navigation on an electronic map, and routing. The path model is fully compatible with pgRouting functions and allows you to smoothly migrate existing applications.

Quick start

- Create an extension.

```
Create Extension Ganos_Networking cascade;
```

- Create a table.

```
CREATE TABLE edge_table (
    id BIGSERIAL,
    dir character varying,
    source BIGINT,
    target BIGINT,
    cost FLOAT,
    reverse_cost FLOAT,
    capacity BIGINT,
    reverse_capacity BIGINT,
    category_id INTEGER,
    reverse_category_id INTEGER,
    x1 FLOAT,
    y1 FLOAT,
    x2 FLOAT,
    y2 FLOAT,
    the_geom geometry
);
```

- Insert data.

```
INSERT INTO edge_table (
    category_id, reverse_category_id,
    cost, reverse_cost,
    capacity, reverse_capacity,
    x1, y1,
    x2, y2) VALUES
(3, 1,    1,  1,  80, 130,   2,   0,    2, 1),
(3, 2,   -1,  1,  -1, 100,   2,   1,    3, 1),
(2, 1,   -1,  1,  -1, 130,   3,   1,    4, 1),
(2, 4,    1,  1, 100,  50,   2,   1,    2, 2),
(1, 4,    1, -1, 130,  -1,   3,   1,    3, 2),
(4, 2,    1,  1,  50, 100,   0,   2,    1, 2),
(4, 1,    1,  1,  50, 130,   1,   2,    2, 2),
(2, 1,    1,  1, 100, 130,   2,   2,   3, 2),
(1, 3,    1,  1, 130,  80,   3,   2,    4, 2),
(1, 4,    1,  1, 130,  50,   2,   2,    2, 3),
(1, 2,    1, -1, 130,  -1,   3,   2,   3, 3),
(2, 3,    1, -1, 100,  -1,   2,   3,   3, 3),
(2, 4,    1, -1, 100,  -1,   3,   3,   4, 3),
(3, 1,    1,  1,  80, 130,   2,   3,   2, 4),
(3, 4,    1,  1,  80,  50,   4,   2,    4, 3),
(3, 3,    1,  1,  80,  80,   4,   1,    4, 2),
(1, 2,    1,  1, 130, 100,   0.5, 3.5,  1.999999999999,3.5),
(4, 1,    1,  1,  50, 130,   3.5, 2.3,  3.5,4);
```

- Update data.

```
UPDATE edge_table SET the_geom = st_makeline(st_point(x1,y1),st_point(x2,y2)),
dir = CASE WHEN (cost>0 AND reverse_cost>0) THEN 'B'
           WHEN (cost>0 AND reverse_cost<0) THEN 'FT'
           WHEN (cost<0 AND reverse_cost>0) THEN 'TF'
           ELSE '' END;
```

- Create a topology.

```
SELECT pgr_createTopology('edge_table',0.001);
```

- Query the shortest path.

```
-- Use Dijkstra's algorithm to query the shortest path.
SELECT * FROM pgr_dijkstra(
    'SELECT id, source, target, cost, reverse_cost FROM edge_table',
    2, 3
);
 seq | path_seq | node | edge | cost | agg_cost
-----+----------+------+------+------+----------
   1 |        1 |    2 |    4 |    1 |        0
   2 |        2 |    5 |    8 |    1 |        1
   3 |        3 |    6 |    9 |    1 |        2
   4 |        4 |    9 |   16 |    1 |        3
   5 |        5 |    4 |    3 |    1 |        4
   6 |        6 |    3 |   -1 |    0 |        5
(6 rows)
-- Use the A* algorithm to query the shortest path.
SELECT * FROM pgr_astar(
    'SELECT id, source, target, cost, reverse_cost, x1, y1, x2, y2 FROM edge_table',
    2, 12,
    directed := false, heuristic := 2);
 seq | path_seq | node | edge | cost | agg_cost
-----+----------+------+------+------+----------
   1 |        1 |    2 |    2 |    1 |        0
   2 |        2 |    3 |    3 |    1 |        1
   3 |        3 |    4 |   16 |    1 |        2
   4 |        4 |    9 |   15 |    1 |        3
   5 |        5 |   12 |   -1 |    0 |        4
(5 rows)
-- Use the turn restricted shortest path (TRSP) algorithm to query the shortest path.
SELECT * FROM pgr_trsp(
        'SELECT id::INTEGER, source::INTEGER, target::INTEGER, cost FROM edge_table',
        2, 7, false, false,
        'SELECT to_cost, target_id::int4,
        from_edge || coalesce('','' || via_path, '''') AS via_path
        FROM restrictions'
    );
 seq | id1 | id2 | cost
-----+-----+-----+------
   0 |   2 |   4 |    1
   1 |   5 |  10 |    1
   2 |  10 |  12 |    1
   3 |  11 |  11 |    1
   4 |   6 |   8 |    1
   5 |   5 |   7 |    1
   6 |   8 |   6 |    1
   7 |   7 |  -1 |    0
(8 rows)
```

- Delete the extension.

```
Drop Extension Ganos_Networking cascade;
```

SQL reference

For more information, see the official pgRouting manual.

2.4. Point cloud model

Point cloud data consists of the points generated by a 3D scanner when it scans an object. Each point is represented by a set of 3D coordinates, and some may contain RGB or intensity information. Point cloud data contains spatial coordinate information, and has a large number of points and complex attribute dimensions.

Overview

Ganos PointCloud is an extension of ApsaraDB for PolarDB. It enables ApsaraDB for PolarDB to store and manage point cloud data efficiently and fast. Ganos PointCloud provides features such as point cloud data compression, decompression, and attribute statistics collection. It can also work with Ganos Geometry to support spatial analysis of point cloud data.

Point cloud data types

Point cloud data types include pcpoint and pcpatch. For the pcpoint type, each point is stored as a row. The dimensions of a point are defined in the point cloud schema. For the pcpatch type, points are stored as a collection. Point cloud data of the pcpatch type can be compressed to reduce storage space and supports spatial queries. The compression method is specified by the compression parameter in the point cloud schema.

Point cloud schema

The pointcloud_formats table stores point cloud schemata. Each schema lists the attribute dimensions of a point cloud object and provides the information about each dimension, such as the size, type, name, and description.

Getting started

- Enable Ganos PointCloud.

```
Create extension ganos_pointcloud cascade;
Create extension ganos_pointcloud_geometry cascade;
```

- Insert a point cloud schema.

```
INSERT INTO pointcloud_formats (pcid, srid, schema) VALUES (1, 4326,
'<?xml version="1.0" encoding="UTF-8"?>
<pc:PointCloudSchema xmlns:pc="http://example.org/schemas/PC/1.1"
    xmlns:xsi="http://www.example.org/2001/XMLSchema-instance">
  <pc:dimension>
    <pc:position>1</pc:position>
    <pc:size>4</pc:size>
    <pc:description>X coordinate as a long integer. You must use the
                    scale and offset information of the header to
                    determine the double value.</pc:description>
    <pc:name>X</pc:name>
    <pc:interpretation>int32_t</pc:interpretation>
    <pc:scale>0.01</pc:scale>
  </pc:dimension>
  <pc:dimension>
    <pc:position>2</pc:position>
    <pc:size>4</pc:size>
    <pc:description>Y coordinate as a long integer. You must use the
                    scale and offset information of the header to
                    determine the double value.</pc:description>
    <pc:name>Y</pc:name>
    <pc:interpretation>int32_t</pc:interpretation>
    <pc:scale>0.01</pc:scale>
  </pc:dimension>
  <pc:dimension>
    <pc:position>3</pc:position>
    <pc:size>4</pc:size>
    <pc:description>Z coordinate as a long integer. You must use the
                    scale and offset information of the header to
                    determine the double value.</pc:description>
    <pc:name>Z</pc:name>
    <pc:interpretation>int32_t</pc:interpretation>
    <pc:scale>0.01</pc:scale>
  </pc:dimension>
  <pc:dimension>
    <pc:position>4</pc:position>
    <pc:size>2</pc:size>
    <pc:description>The intensity value is the integer representation
                    of the pulse return magnitude. This value is optional
                    and system specific. However, it should always be
                    included if available.</pc:description>
    <pc:name>Intensity</pc:name>
    <pc:interpretation>uint16_t</pc:interpretation>
    <pc:scale>1</pc:scale>
  </pc:dimension>
  <pc:metadata>
    <Metadata name="compression">dimensional</Metadata>
  </pc:metadata>
</pc:PointCloudSchema>');
```

- Create a point cloud table.

```
-- Use the pcpoint data type.
CREATE TABLE points (
    id SERIAL PRIMARY KEY,
    pt PCPOINT(1)
);
-- Use the pcpatch data type.
CREATE TABLE patches (
    id SERIAL PRIMARY KEY,
    pa PCPATCH(1)
);
```

- Insert data of pcpoint data type.

```
INSERT INTO points (pt)
SELECT ST_MakePoint(1, ARRAY[x,y,z,intensity])
FROM (
  SELECT
  -127+a/100.0 AS x,
    45+a/100.0 AS y,
         1.0*a AS z,
          a/10 AS intensity
  FROM generate_series(1,100) AS a
) AS values;
SELECT ST_MakePoint(1, ARRAY[-127, 45, 124.0, 4.0]);
-------------------------
010100000064CEFFFF94110000703000000400
SELECT ST_AsText('010100000064CEFFFF94110000703000000400'::pcpoint);
-------------------------
null
```

- Insert data of pcpatch data type.

```
INSERT INTO patches (pa)
SELECT ST_Patch(pt) FROM points GROUP BY id/10;
SELECT ST_AsText(ST_MakePatch(1, ARRAY[-126.99,45.01,1,0, -126.98,45.02,2,0, -126.97,45.0
3,3,0]));
-------------------------
{"pcid":1,"pts":[
 [-126.99,45.01,1,0],[-126.98,45.02,2,0],[-126.97,45.03,3,0]
]}
```

- Calculate the average values of all attribute dimensions in a pcpatch object.

```
SELECT ST_AsText(ST_PatchAvg(pa)) FROM patches WHERE id = 7;
-------------------------
null
```

- Disable Ganos PointCloud.

```
Drop extension ganos_pointcloud_geometry;
Drop extension ganos_pointcloud cascade;
```

SQL reference

For more information, see Point cloud SQL reference.

2.5. Trajectory model

Trajectory data records the continuous location update information about a moving feature, such as a vehicle or a person. Trajectory data is a type of typical spatio-temporal data. You can analyze and use trajectory data to study many important issues.

Overview

Ganos Trajectory is an extension of PolarDB. Ganos Trajectory provides a series of data types, functions, and stored procedures. This allows you to manage, query, and analyze spatio-temporal trajectory data.

Quick start

- Create the Ganos Trajectory extension.

```
Create Extension Ganos_trajectory cascade;
```

- Create the enumeration type for a trajectory.

```
CREATE TYPE leaftype AS ENUM ('STPOINT', 'STPOLYGON');
```

- Create a trajectory table.

```
Create Table traj_table (id integer, traj trajectory);
```

- Insert trajectory data records into the trajectory table.

```
insert into traj_table values
(1, ST_MakeTrajectory('STPOINT'::leaftype, st_geomfromtext('LINESTRING (114 35, 115 36, 1
16 37)', 4326), '[2010-01-01 14:30, 2010-01-01 15:30)'::tsrange, '{"leafcount": 3,"attrib
utes" : {"velocity" : {"type":"integer","length":4,"nullable":false,"value":[120, 130, 14
0]},"accuracy":{"type":"integer","length":4,"nullable":false,"value":[120, 130, 140]},"be
aring":{"type":"float","length":4,"nullable":false,"value":[120, 130, 140]},"acceleration
":{"type":"float","length":4,"nullable":false,"value":[120, 130, 140]}}}')),
(2, ST_MakeTrajectory('STPOINT'::leaftype, st_geomfromtext('LINESTRING (114 35, 115 36, 1
16 37)', 4326), '2010-01-01 14:30'::timestamp, '2010-01-01 15:30'::timestamp, '{"leafcoun
t": 3,"attributes" : {"velocity" : {"type":"integer","length":4,"nullable":false,"value":
[120, 130, 140]},"accuracy":{"type":"integer","length":4,"nullable":false,"value":[120, 1
30, 140]},"bearing":{"type":"float","length":4,"nullable":false,"value":[120, 130, 140]},
"acceleration":{"type":"float","length":4,"nullable":false,"value":[120, 130, 140]}}}')),

(3, ST_MakeTrajectory('STPOINT'::leaftype, st_geomfromtext('LINESTRING (114 35, 115 36, 1
16 37)', 4326),ARRAY['2010-01-01 14:30'::timestamp, '2010-01-01 15:00'::timestamp, '2010-
01-01 15:30'::timestamp], '{"leafcount": 3,"attributes" : {"velocity" : {"type":"integer"
,"length":4,"nullable":false,"value":[120, 130, 140]},"accuracy":{"type":"integer","lengt
h":4,"nullable":false,"value":[120, 130, 140]},"bearing":{"type":"float","length":4,"null
able":false,"value":[120, 130, 140]},"acceleration":{"type":"float","length":4,"nullable"
:false,"value":[120, 130, 140]}}}')),
(4, ST_MakeTrajectory('STPOINT'::leaftype, st_geomfromtext('LINESTRING (114 35, 115 36, 1
16 37)', 4326), '[2010-01-01 14:30, 2010-01-01 15:30)'::tsrange, null));
```

- Create a trajectory index on the trajectory table.

```
--Create a trajectory index to accelerate the process of filtering spatio-temporal data r
ecords.
create index tr_index on traj_table using trajgist (traj);
--Run spatial data queries. You can find that the trajectory index accelerates the proces
s of filtering spatial data records.
select id, traj_id from traj_table where st_3dintersects(traj, ST_GeomFromText('POLYGON((
116.46747851805917 39.92317964155052,116.4986540687358 39.92317964155052,116.498654068735
8 39.94452401711516,116.46747851805917 39.94452401711516,116.46747851805917 39.9231796415
5052))'));
--Run temporal data queries. You can find that the trajectory index accelerates the proce
ss of filtering temporal data records.
select id, traj_id from traj_table where st_TWithin(traj, ST_ToBox('2008-02-02 13:30:44':
:timestamp,'2008-02-03 17:30:44'::timestamp));
--Run spatio-temporal data queries. You can find that the trajectory index accelerates th
e process of filtering spatio-temporal data records.
select id, traj_id from traj_table where st_3dintersects(traj, ST_GeomFromText('POLYGON((
116.46747851805917 39.92317964155052,116.4986540687358 39.92317964155052,116.498654068735
8 39.94452401711516,116.46747851805917 39.94452401711516,116.46747851805917 39.9231796415
5052))'),'2008-02-02 13:30:44'::timestamp,'2008-02-03 17:30:44'::timestamp);
```

- Create a trajectory index in a specific dimension.

```
--If you want to analyze trajectory data only from a specific dimension, create a traject
ory index in that dimension. For example, if you do not want to analyze trajectory data i
n the z dimension, create a two-dimensional temporal trajectory index by using trajgist_o
p_2dt.
create index tr_timespan_time_index on traj_table using trajgist (traj trajgist_op_2dt);
--Query two-dimensional temporal data. The queries are run at a fast rate.
select id, traj_id from traj_table where st_2dintersects(traj, ST_GeomFromText('POLYGON((
116.46747851805917 39.92317964155052,116.4986540687358 39.92317964155052,116.498654068735
8 39.94452401711516,116.46747851805917 39.94452401711516,116.46747851805917 39.9231796415
5052))'),'2008-02-02 13:30:44'::timestamp,'2008-02-03 17:30:44'::timestamp);
--Create more trajectory indexes based on your business requirements. If you create one t
rajectory index, all queries run based on the trajectory index. If you create multiple tr
ajectory indexes, ApsaraDB for RDS selects the optimal trajectory index when you run quer
ies.
create index tr_timespan_time_index on traj_table using trajgist (traj trajgist_op_2d);
select id, traj_id from traj_table where st_2dintersects(traj, ST_GeomFromText('POLYGON((
116.46747851805917 39.92317964155052,116.4986540687358 39.92317964155052,116.498654068735
8 39.94452401711516,116.46747851805917 39.94452401711516,116.46747851805917 39.9231796415
5052))'));
```

- Query the start time and end time of trajectories.

```
select st_startTime(traj), st_endTime(traj) from traj_table ;
    st_starttime      |      st_endtime
---------------------+---------------------
 2010-01-01 14:30:00 | 2010-01-01 15:30:00
 2010-01-01 14:30:00 | 2010-01-01 15:30:00
 2010-01-01 14:30:00 | 2010-01-01 15:30:00
 2010-01-01 14:30:00 | 2010-01-01 15:30:00
 2010-01-01 14:30:00 | 2010-01-01 15:30:00
 2010-01-01 11:30:00 | 2010-01-01 15:00:00
 2010-01-01 11:30:00 | 2010-01-01 15:00:00
 2010-01-01 11:30:00 | 2010-01-01 15:00:00
(8 rows)
```

- Query the trajectories of moving objects.

```
--Query the attributes of a trajectory point by using an interpolation function.
Select ST_velocityAtTime(traj, '2010-01-01 12:45') from traj_table  where id > 5;
st_velocityattime
-------------------
5
5
4.16666666666667
(3 rows)
```

- Analyze the proximity among trajectories.

```
postgres=# Select ST_euclideanDistance((Select traj From traj_table Where id = 6), (Selec
t traj From traj_table Where id = 7));
 st_euclideandistance
----------------------
   0.0334968923954815
(1 row)
```

- Delete the Ganos Trajectory extension.

```
Drop Extension Ganos_trajectory cascade;
```

SQL reference

For more information, see Trajectory SQL reference.

2.6. Grid model

Grid models are discrete, multi-scale location identification systems that are developed from GeoSOT.

Overview

A grid model allows you to assign globally unique codes to various objects within the earth space (from the earth's core to the earth's surface). The globally unique grid code of an object can be used to associate the object with various data that is collected from the same location as the object.

Quick start

- Create an extension.

```
CREATE EXTENSION Ganos_GeomGrid CASCADE;
```

- Create a table with a grid code.

```
CREATE TABLE t_grid(id integer,
                    geom geometry,  -- A geometry.
                    grid1 geomgrid[], -- A grid code with a precision of 1.
                    grid2 geomgrid[], -- A grid code with a precision of 2.
                    grid3 geomgrid[] -- A grid code with a precision of 3.
                   );
```

- Compute the grid code of the table.

```
--Insert data into the table.
INSERT INTO t_grid(id, geom)
VALUES (1, ST_GeomFromText('POINT(116.31522216796875 39.910277777777778)', 4490)),
       (2, ST_GeomFromText('POINT(116.31522217796875 39.910277776777778)', 4490)),
       (3, ST_GeomFromText('POINT(116.31522217797875 39.910277776787778)', 4490)),
       (4, ST_GeomFromText('POINT(116.31522227796875 39.910277776775778)', 4490));
--Create grid codes of different precision levels.
UPDATE t_grid
SET grid1 = ST_AsGrid(geom, 10),
    grid2 = ST_AsGrid(geom, 15),
    grid3 = ST_AsGrid(geom, 26);
```

- Create indexes on grid codes.

```
--Create GIN indexes on grid codes of different precision levels.
CREATE INDEX idx_grid_gin1
ON t_grid
USING GIN(grid1);
CREATE INDEX idx_grid_gin2
ON t_grid
USING GIN(grid2);
CREATE INDEX idx_grid_gin3
ON t_grid
USING GIN(grid3);
```

- Query data.

```
--Query data that resides in a grid.
SELECT id
FROM t_grid
WHERE grid2 = ARRAY[ST_GridFromText('G001310322230230')];
--Query data that intersects with a grid.
SELECT id
FROM t_grid
WHERE grid3 @> ARRAY[ST_GridFromText('G00131032223023031031033223')];
--Query data that intersects with more than one grid.
SELECT id
FROM t_grid
WHERE grid3 && ARRAY[ST_GridFromText('G00131032223023031031211001'),
                        ST_GridFromText('G00131032223023031031211111')];
--Query data that intersects with more than one geometry in a grid.
SELECT id
FROM t_grid
WHERE grid3 &&
ST_AsGrid(
  ST_GeomFromText('LINESTRING(116.31522216796875 39.910277777777778, 116.31522217797875 3
9.910277776787778)', 4490), 26);
```

- Delete the extension that you created.

```
DROP EXTENSION Ganos_GeomGrid CASCADE;
```

2.7. Vector pyramid

Vector pyramids are structures that are designed to display tens of millions of spatial geometry data records.

Overview

Vector pyramids allow you to create sparse indexes on spatial geometry data records and to preprocess data-intensive areas based on the specified rules. This way, you can obtain standard data records in the MVT and PBF formats.

Vector pyramids provided by Ganos allow you to preprocess hundreds of millions of spatial geometry data records in minutes and to display the preprocessing results in seconds.

Quick start

- Create an extension.

```
CREATE EXTENSION ganos_geometry_pyramid CASCADE;
```

- Create test data.

```
CREATE TABLE test(id int primary key, geom geometry(Point,4326));
INSERT INTO test(id, geom)
    SELECT  i,
            ST_GeomFromText(
              format('POINT(%s %s)', floor(i / 100) * 0.01, (i % 100) * 0.01),
              4326)
    FROM generate_series(1,10000) i;
CREATE INDEX ON test USING gist(geom);
```

- Create a pyramid for a spatial table.

```
--Create a pyramid for the data table named test.
--Specify the name of the spatial fields in the test table. Before you specify the name o
f a spatial field, you must create a spatial index for the field.
SELECT ST_BuildPyramid('test', 'geom', 'id', '');
```

- Read MVT-formatted data records from the pyramid that you created.

```
--Read the data record with the tile ID 0_0_0 from the pyramid. You can specify any tile
ID. The system returns a data record regardless of whether the specified tile ID can be f
ound in the pyramid.
--Make sure that the specified tile ID follows the z_x_y format and the EPSG:3857 coordin
ate system.
SELECT ST_Tile('test', '0_0_0');
```

- Delete the pyramid that you created.

```
--Delete the pyramid named test.
SELECT ST_DeletePyramid('test');
```

- Delete the extension that you created.

```
DROP EXTENSION ganos_geometry_pyramid CASCADE;
```

Parameter settings

- Pyramid name

  The default name of a pyramid is the same as the data table to which the pyramid belongs. You can change the default name to map one data table to multiple pyramids.

```
--Create a pyramid named hello for the data table named test.
SELECT ST_BuildPyramid('test', 'geom', 'id', '{"name": "hello"}');
```

- Number of parallel tasks

  You can specify the maximum number of tasks that can run in parallel to create pyramids. The default value is 0. The value 0 indicates that the number of parallel tasks is not limited. The value cannot exceed four times the number of cores that are configured.

  Pyramids are created in parallel in compliance with the two-phase commit (2PC) protocol. You must set the max_prepared_transactions parameter to a value that is greater than or equal to 100. After you set the parameter, you must restart your PolarDB cluster to make the new parameter setting take effect.

```
--Configure four cores to create pyramids in parallel.
SELECT ST_BuildPyramid('test', 'geom', 'id','{"parallel": 4}');
```

- Tile parameters

  You can specify the tile size and the tile extension size.

  - Valid tile sizes are multiples of 256 within the range of 0 to 4096.
  - Valid tile extension sizes are within the range of 0 to 256.

```
--Set the tile size to 512 and the tile extension size to 8.
SELECT ST_BuildPyramid('test', 'geom', 'id','{
                        "tileSize": 512,
                        "tileExtend": 8
                       }');
```

- Maximum number of pyramid layers

  You can specify the maximum number of layers in a pyramid. The default value is 16. If the number of zoom levels is greater than the maximum number of pyramid layers, the layers beyond the maximum number are no longer included in the pyramid. If you do not set this parameter, the system provides a maximum number based on data density.

```
--Set the maximum number of pyramid layers to 12. If the number of zoom levels exceeds 12
, the system reads data in real time to generate data records that are in the MVT format.

SELECT ST_BuildPyramid('test', 'geom', 'id', '{"maxLevel": 12}');
```

- Layered processing

  You can specify unique criteria that are used to process each layer of a pyramid. The criteria include the fields that you want to display and the filter conditions.

  You can specify the criteria by using the `buildRules` element.

  The system requires a long period of time to generate the top layer of a pyramid. You can use layered processing rules to skip the top layer.

```
--Enable layered processing.
--Specify only the "1!=1" filter condition for Layers 0 through 5.The system generates em
pty data records in the MVT format and does not display the data of Layers 0 through 5.
--Specify the name field that you want to display and the "code = 1" filter condition for
Layers 6 through 9.
--Specify only the name and width fields that you want to display for Layers 10 through 1
5.
SELECT ST_BuildPyramid('test', 'geom', 'id', '{
                        "buildRules":[
                          {
                            "level":[0,1,2,3,4,5],
                            "value": {
                              "filter": "1 != 1"
                            }
                          },
                          {
                            "level":[6,7,8,9],
                            "value": {
                              "filter": "code = 1",
                              "attrFields": ["name"]
                            }
                          },
                          {
                            "level":[10,11,12,13,14,15],
                            "value": {
                              "attrFields": ["name", "width"]
                            }
                          }
                        ]
                      }');
```

Advanced features

You can merge the data records within a specific tile based on attributes to reduce the total number of data records.

```
--Separately merge the data records whose codes are 1 and those whose codes are 2.
SELECT ST_BuildPyramid('test', 'geom', 'id', '{
                        "buildRules":[
                                {
                                    "level":[0,1,2,3,4,5],
                                    "value": {
                                        "merge": ["code=1","code=2"]
                                    }
                                }
                            ]
                        }');
```

3.Advanced usage

3.1. Create spatial indexes in parallel

This feature uses the GiST Sort method to parallelize the index creation process, which significantly reduces the read/write operations on disks and accelerates index creation.

Precautions

The GiST Sort method is only applicable to points. Using this method for other types of spatial data will affect the query performance of indexes.

Usage

> **Note** The number of workers and the total memory usage of workers in the following statements are provided for reference. You can configure them as needed.

1. Enable the GiST Sort feature.

```
set polar_enable_gist_sort=on;
```

2. Configure the number of parallel workers.

   The more parallel workers that you use to scan data tables, the higher the CPU load is during query. Therefore, we recommend that the number of workers you set do not exceed the number of physical CPU cores.

```
set max_parallel_maintenance_workers=4;
```

3. Configure the total memory usage of parallel workers. This parameter is recommended to be at least 1G.

```
set maintenance_work_mem='1GB';
```

4. Create a GiST index.

```
create index on t using gist(geom);
```

5. Execute the following statement to disable the Gist Sort feature:

```
set polar_enable_gist_sort=off;
```

3.2. Enable spatio-temporal two-phase query optimization

This topic describes the spatio-temporal two-phase query optimization method that is provided by Ganos. This method optimizes a query in two phases based on the spatio-temporal index framework. This allows you to reduce extra data I/O and computing overheads.

Context

A conventional spatio-temporal query runs in two phases: coarse filtering and precise filtering. In the coarse filtering phase, the database system coarsely filters data records based on multi-dimensional spatio-temporal indexes to obtain an intermediate result set. In the precise filtering phase, the database system invokes an appropriate function to further filter data records from the intermediate result set. This allows the database system to obtain the final result set.

Example

The database system coarsely filters data records from the test table based on the spatial index on the test table and based on the query object. This allows the database system to obtain an intermediate result set. Then, the database system invokes the _st_intersects function to precisely filter data records from the intermediate result set. This allows the database system to obtain the final result set that consists of qualified data records.

In this example, the intermediate result set that you obtain in the coarse filtering phase still contains a large number of irrelevant data records. These irrelevant data records increase data I/O and computing overheads in the precise filtering phase.

```
select id from test where st_intersects(ST_GeomFromText('POLYGON((250000 -268000, 250000 27
0000, 280000 270000,280000 -268000, 250000 -268000))', 3857),geom)=true;
```

Two-phase query optimization

The two-phase query optimization method is suitable for spatio-temporal range queries and point in polygon (PIP) queries. This method is controlled by using the guc parameter. However, this parameter is invisible to users. This parameter is set to off by default.

You can run the following command to enable or disable the two-phase query optimization method for a session on your RDS instance:

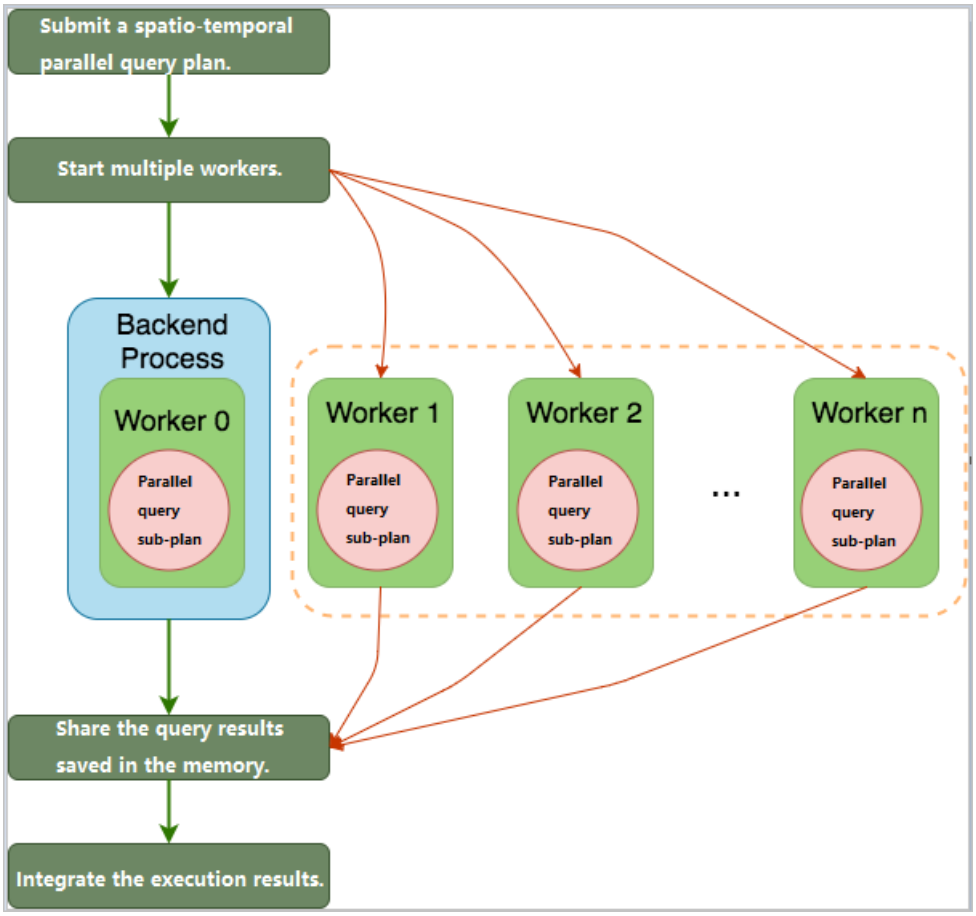
```
# Enable the two-phase query optimization method.
set polar_enable_gist_refine = true;
# Disable the two-phase query optimization method.
set polar_enable_gist_refine = false;
```

3.3. Enable spatio-temporal parallel query

Ganos can use the parallel query capability of PostgreSQL to accelerate complex spatio-temporal data queries that involve a large amount of data.

Parallel query principles

Parallel query is performed at the table level. The following figure shows the parallel query process.



Precautions

- A larger number of workers used for parallel query indicate a higher CPU load. If the CPU load is already high, we recommend that you set the number of workers to 2 by setting the max_parallel_workers_per_gather parameter to 2.
- When enabling parallel query for highly concurrent access requests on a server with limited memory, you must properly set the work_mem parameter (to a minimum of 64 KB). You must ensure that the number of concurrent access requests multiplied by the number of parallel workers multiplied by the value of the work_mem parameter does not exceed 60% of the server memory.

Usage

To enable Ganos parallel query, perform the following operations:

1. Set parallel query parameters in the postgresql.conf profile.
   - Set the max_parallel_workers parameter to specify the total number of parallel workers that can be enabled, in the range of 8 to 32. The value of this parameter must be smaller than that of the max_worker_processes parameter.
   - Set the max_parallel_workers_per_gather parameter to specify the maximum number of parallel workers for a single query gather, in the range of 2 to 4. The value of this parameter must be smaller than that of the max_parallel_workers parameter.
   - To forcibly enable parallel query, set the force_parallel_mode parameter to on.
   - To set the number of parallel workers for a table, execute the following SQL statement: alter

table table_name set (parallel_workers=n).

> **Note** In the preceding SQL statement, n indicates the number of parallel workers. For more information about how to set an appropriate value for n, see the description of the max_parallel_workers_per_gather parameter.

2. Increase the cost of Ganos functions.

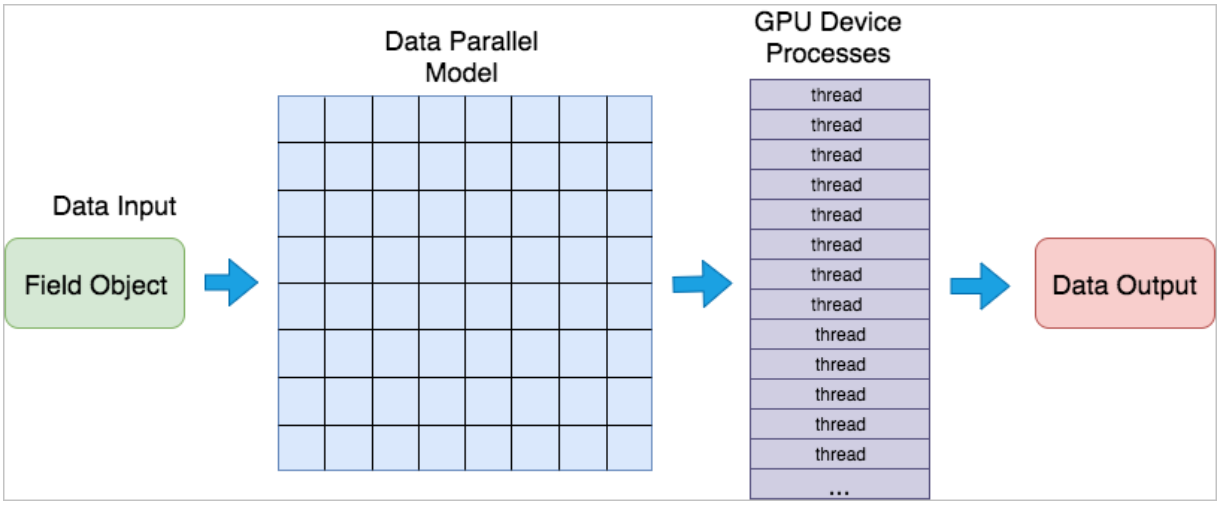
   After a Ganos module extension is created, functions have a default cost. If the data size of the table for the module extension is small, parallel query is disabled by default. If a function is computing-intensive and parallel query is suitable for the function, you must increase the cost of the function before you can enable parallel query.

3.4. Enable GPU-accelerated computing

Due to its special hardware architecture, a GPU has great advantages over a CPU in processing computing-intensive and easy-to-parallel code.

Acceleration principles

In a database, GPU parallel acceleration is performed at the object level. Ganos converts a field object to a model suitable for parallel computing, and uses the multiple cores of the GPU for parallel computing. The following figure shows the parallel computing process.



Notes

A single GPU has limited resources. Therefore, we recommend that you disable GPU-accelerated computing in a session for high-concurrency scenarios.

Procedure

When detecting a GPU, Ganos automatically enables GPU-accelerated computing. You do not need to set any parameters. GPU-accelerated computing is transparent and imperceptible to you. In addition, Ganos allows you to enable or disable GPU-accelerated computing at the session level.

1. Check whether a GPU is available in the current environment.

i. Run the `create extension ganos_raster cascade` statement to create a Ganos Raster extension.

ii. Run the `select st_checkgpu()` statement.

- If Ganos detects a GPU in the current environment, it returns the GPU information.

```
rasterdb=# select st_checkgpu();
                    st_checkgpu

--------------------------------------------------------------------------------
--------------------------------------------
 [GPU(0) prop]multiProcessorCount=20; sharedMemPerBlock=49152; totalGlobalMem=-60
8239616; maxThreadsPerBlock=1024; maxThreadsPerMultiProcessor=2048; cudaThreadGet
Limit=1024.
(1 row)
```

- If Ganos does not detect a GPU in the current environment, it returns an error message.

```
rasterdb=# select st_checkgpu();
                   st_checkgpu

--------------------------------------------------------------------------------
-------------------------------------------------------
There is not gpu device on current environment, cuda_errorcode=35, errormsg=CUDA
driver version is insufficient for CUDA runtime version.
(1 row)
```

> **Note** GPU-accelerated computing can be enabled only in an environment where a GPU is available.

2. Enable or disable GPU-accelerated computing at the session level.

   - In an environment with a GPU, Ganos enables GPU-accelerated computing by default. To disable GPU-accelerated computing to use the original CPU computing mode, perform the following operation in a session:

     Run the set ganos.raster.use_cuda=off statement.

```
rasterdb=# set ganos.raster.use_cuda=off;
SET
rasterdb=# show ganos.raster.use_cuda;
 ganos.raster.use_cuda
-----------------------
 off
(1 row)
```

   - To enable GPU-accelerated computing again, run the set ganos.raster.use_cuda=on statement.

```
rasterdb=# set ganos.raster.use_cuda=on;
SET
rasterdb=# show ganos.raster.use_cuda;
 ganos.raster.use_cuda
-----------------------
 on
(1 row)
```

3. Implement GPU-accelerated computing in a Ganos module.

> **Note** Currently, you can implement GPU-accelerated computing only in the Ganos Raster module. GPU-accelerated computing will be supported by the Ganos Trajectory and Ganos Geometry modules in the future.

3.5. Spatial-temporal object storage optimization

3.5.1. Feature signature-based storage optimization for large spatio-temporal objects

The feature signatures of large spatio-temporal objects are used in Ganos to optimize the storage of these objects. This storage optimization feature allows you to store the metadata of these objects in-row and store the entity data of these objects off-row. This increases processing performance and reduces I/O overhead in scenarios such as spatio-temporal index creation and spatial joins.

Configure the storage optimization feature

After you connect to your PolarDB cluster, you can enable or disable the storage optimization feature by using a Grand Unified Configuration (GUC) parameter. By default, the storage optimization feature is disabled.

- To enable the storage optimization feature, run the following command:

```
set polar_enable_storage_partial = true;
```

- To disable the storage optimization feature, run the following command:

```
set polar_enable_storage_partial = false;
```

Configure the threshold for storage optimization

After you enable the storage optimization feature, PolarDB implements in-row storage and off-row storage on a large spatio-temporal object only when the number of bytes for the object exceeds the value of the `polar_partial_external_threshold` parameter.

The value of the `polar_partial_external_threshold` parameter ranges from 1000 to 8000. The default value is 2000. Before you create a table, you can reconfigure this parameter. Example:

```
set polar_partial_external_threshold = 3000;
```

3.5.2. ST_SetTypeStorage

This topic describes the ST_SetTypeStorage function, which is used to set the storage mode for specific data types. This function allows you to store some types of data in-row and store the other types of data off-row.

Syntax

```
bool ST_SetTypeStorage(cstring typeName, cstring storageStrategy, integer size);
```

Parameters

Parameter	Description
typeName	The types of data whose storage mode you want to specify. The supported data types include geometry and geography.
storageStrategy	The storage mode that you want to specify. Set the value to main.
size	The number of bytes that can be stored in-row.

Description

If the size per object for a specific data type is large, you can store the metadata in-row and store the entity data off-row. For normal query and analysis workloads, PolarDB needs to read only the in-row metadata. This reduces the I/O operations on the disk.

The size parameter specifies the number of in-row bytes. If you set this parameter to 0, data is still stored in main mode. The following table lists the recommended values of this parameter for various data types.

Data type	Dimension	Recommended value
geometry	2d(x,y)	24
	3d(x,y,z) and 3dm(x,y,m)	32
	4d(x,y,z,m)	40
geography	2d(x,y)	32

Example

```
Select ST_SetTypeStorage('geometry','main', 41);
 st_settypestorage
-------------------
 t
(1 row)
```

3.5.3. Use the Simple storage policy

This topic describes how to use the Simple storage policy in the spatio-temporal database Ganos.

Background information

Spatio-temporal data features large data objects and the total amount of spatio-temporal data is large. In most cases, data of the types, such as grids, trajectories, and point clouds, in Ganos must be compressed by using professional compression algorithms and then stored in database tables. The storage policy provided by the current database supports only fixed and simple compression algorithms. Data can also be stored in external tables without being compressed. Due to this policy, spatio-temporal occupies a lot of storage space and affects query performance.

Therefore, a simple storage policy called Simple is provided in Ganos. This policy allows you to use a custom compression algorithm to compress data before the data is stored. The compressed data is stored in basic tables if possible. The policy reduces storage costs, and the impact on query performance as much as possible.

Use the policy

Specify the Grand Unified Configuration (GUC) parameter. By default, this parameter is enabled.

- To enable the GUC parameter, execute the following statement:

```
SET polar_enable_storage_simple=true;
```

- To disable the GUC parameter, execute the following statement:

```
SET polar_enable_storage_simple=false;
```

Set the specified field of the table to `storage simple`, as shown in the following example:

```
ALTER TABLE tbname ALTER colname SET storage simple;
```

After the settings are complete, the colname field is automatically stored based on the Simple storage policy.

4.Geometry SQL reference

4.1. Output functions

4.1.1. ST_AsBinary

This topic describes the ST_AsBinary function. This function returns the Well-Known Binary (WKB) string that represents a geometry object or a geography object. The WKB string that is returned does not contain spatial reference identifier (SRID) metadata.

Syntax

```
bytea  ST_AsBinary(geometry g1);
bytea  ST_AsBinary(geometry g1, text NDRorXDR);
bytea  ST_AsBinary(geography g1);
bytea  ST_AsBinary(geography g1, text NDRorXDR);
```

Parameters

Parameter	Description
g1	The geometry object or geography object whose WKB representation you want to obtain.
NDRorXDR	The byte order that you want to use. Valid values: • XDR: big endian. • NDR: little endian. This is the default value.

Description

- The WKB specification defines that standard WKB strings do not contain SRIDs. If you want to obtain a WKB string that contains an SRID, use the ST_AsEWKB function.
- This function supports circular strings, curves, polyhedral surfaces, triangles, triangulated irregular network (TIN) surfaces, and 3D objects.

Examples

- Obtain a WKB string with the default little endian byte order retained.

```
SELECT ST_AsBinary(ST_GeomFromText('POLYGON((1 1,1 2,2 2,2 1,1 1))',4326));
                          st_asbinary
--------------------------------------------------------------
 \x0103000000010000000500000000000000000f03f000000000000f03f0.
.00000000000f03f0000000000000040000000000000000400000000000000.
.40000000000000004000000000000f03f000000000000f03f00000000000.
.0f03f
(1 row)
```

- Obtain a WKB string with the big endian byte order specified.

```
SELECT ST_AsBinary(ST_GeomFromText('POLYGON((1 1,1 2,2 2,2 1,1 1))',4326),'XDR');
                          st_asbinary
----------------------------------------------------------------
 \x000000000300000001000000053ff00000000000003ff00000000000003.
.ff00000000000040000000000000004000000000000000040000000000000.
.004000000000000003ff00000000000003ff00000000000003ff00000000.
.00000
(1 row)
```

4.1.2. ST_AsEncodedPolyline

This topic describes the ST_AsEncodedPolyline function. This function returns the encoded polyline that represents a LineString object.

Syntax

```
text  ST_AsEncodedPolyline(geometry  geom, integer  precision);
```

Parameters

Parameter	Description
geom	The LineString object whose encoded polyline representation you want to obtain.
precision	The precision of encoding. Default value: 5. The precision parameter specifies the number of decimal places that you want to retain in the encoded polyline. The precision of encoding and the precision of decoding must be the same. If the precision of encoding and the precision of decoding are different, the coordinates of the encoded polyline are incorrect.

Examples

```
SELECT ST_AsEncodedPolyline(GeomFromEWKT('SRID=4326;LINESTRING(120 36,130 40,126 43)'));
    st_asencodedpolyline
----------------------------
 _gvzE_ol{U_glW_c`|@_}hQ~flW
(1 row)
```

4.1.3. ST_AsEWKB

This topic describes the ST_AsEWKB function. This function returns the Well-Known Binary (WKB) string that represents a geometry object or a geography object. The WKB string that is returned contains spatial reference identifier (SRID) metadata.

Syntax

```
bytea  ST_AsEWKB(geometry  g1);
bytea  ST_AsEWKB(geometry  g1 , text  NDRorXDR);
bytea  ST_AsEWKB(geography  g1);
bytea  ST_AsEWKB(geography  g1 , text  NDRorXDR);
```

Parameters

Parameter	Description
g1	The geometry object or geography object whose WKB representation you want to obtain.
NDRorXDR	The byte order that you want to use. Valid values: • XDR: big endian. • NDR: little endian. This is the default value.

Description

- The WKB specification defines that standard WKB strings do not contain SRIDs. If you want to obtain a WKB string that contains an SRID, use this function.
- This function supports circular strings, curves, polyhedral surfaces, triangles, triangulated irregular network (TIN) surfaces, and 3D objects.

Examples

- Obtain a WKB string with the default little endian byte order retained.

```
SELECT ST_AsEWKB(ST_GeomFromText('POLYGON((1 1,1 2,2 2,2 1,1 1))',4326));
                           st_asewkb
-------------------------------------------------------------
 \x0103000020e6100000010000000500000000000000000f03f000000000.
.000f03f000000000000f03f0000000000000040000000000000000400000000.
.0000000040000000000000000400000000000000f03f000000000000f03f000.
.000000000f03f
(1 row)
```

- Obtain a WKB string with the big endian byte order specified.

```
SELECT ST_AsEWKB(ST_GeomFromText('POLYGON((1 1,1 2,2 2,2 1,1 1))',4326),'XDR');
                           st_asewkb
-------------------------------------------------------------
 \x0020000003000010e6000000010000000053ff00000000000003ff000000.
.00000003ff0000000000000400000000000000040000000000000000400000.
.0000000000400000000000000003ff00000000000003ff00000000000003ff.
.0000000000000
(1 row)
```

4.1.4. ST_AsEWKT

This topic describes the ST_AsEWKT function. This function returns the Well-Known Text (WKT) string that represents a geometry object or a geography object. The WKT string that is returned contains spatial reference identifier (SRID) metadata.

Syntax

```
text  ST_AsEWKT(geometry  g1);
text  ST_AsEWKT(geography  g1);
```

Parameters

Parameter	Description
g1	The geometry object or geography object whose WKT representation you want to obtain.

Description

- If you want to obtain a standard WKT string that does not contain an SRID, use the ST_AsText function.
- The WKT format cannot ensure precision. If you want to prevent floating-point numbers of a specific decision from being truncated, we recommend that you use the ST_AsBinary function or the ST_AsEWKB function.
- This function supports circular strings, curves, polyhedral surfaces, triangles, triangulated irregular network (TIN) surfaces, and 3D objects.

Examples

- Obtain a WKT string that contains an SRID.

```
SELECT ST_AsEWKT(ST_GeomFromText('POLYGON((1 1,1 2,2 2,2 1,1 1))',4326));
                st_asewkt
-----------------------------------------
 SRID=4326;POLYGON((1 1,1 2,2 2,2 1,1 1))
(1 row)
```

- Obtain a WKT string that does not contain an SRID.

```
SELECT ST_AsEWKT(ST_GeomFromText('POLYGON((1 1,1 2,2 2,2 1,1 1))'));
           st_asewkt
--------------------------------
 POLYGON((1 1,1 2,2 2,2 1,1 1))
(1 row)
```

4.1.5. ST_AsGeoJSON

This topic describes the ST_AsGeoJSON function. This function returns the GeoJSON element that represents a geometry object or a geography object.

Syntax

```
text  ST_AsGeoJSON(geometry  geom , integer  maxdecimaldigits , integer  options);
text  ST_AsGeoJSON(geography  geog , integer  maxdecimaldigits , integer  options);
text  ST_AsGeoJSON(integer  gjVersion , geometry  geom , integer  maxdecimaldigits , integer  options);
text  ST_AsGeoJSON(integer  gjVersion , geography  geog , integer  maxdecimaldigits , integer  options);
```

Parameters

Parameter	Description
geom	The geometry object whose GeoJSON representation you want to obtain.
maxdecimaldigits	The maximum number of decimal places that you want to retain. Default value: 15.
options	The additional information that you want to return in the GeoJSON element. Valid values: • 0: specifies not to return additional information. This is the default value. • 1: specifies to return the bbox member of the GeoJSON element. • 2: specifies to return the coordinate reference system (CRS) of the GeoJSON element by using a short expression. An example expression is `EPSG:4326`. • 4: specifies to return the CRS of the GeoJSON element by using a long expression. An example expression is `urn:ogc:def:crs:EPSG::4326`.
geog	The geography object whose GeoJSON representation you want to obtain.
gjVersion	The version of the GeoJSON specification. If you specify this parameter, set the value to 1.

Description

This function supports 3D objects and does not discard the z-index of the geometry object that is constructed.

Examples

Obtain the GeoJSON element that represents a geometry object.

```
SELECT ST_AsGeoJSON('LINESTRING(1 2, 3 4)');
                   st_asgeojson
-----------------------------------------------------
 {
     "type":"LineString",
  "coordinates":[[1,2],[3,4]]
 }
(1 row)
```

4.1.6. ST_AsGML

This topic describes the ST_AsGML function. This function returns the Geography Markup Language (GML) element that represents a geometry object or a geography object.

Syntax

```
text ST_AsGML(geometry geom , integer maxdecimaldigits , integer options);
text ST_AsGML(geography geog , integer maxdecimaldigits , integer options);
text ST_AsGML(integer version , geometry geom , integer maxdecimaldigits , integer opt
ions , text nprefix, text id);
text ST_AsGML(integer version , geography geog , integer maxdecimaldigits , integer op
tions , text nprefix, text id);
```

Parameters

Parameter	Description
geom	The geometry object whose GML representation you want to obtain.
maxdecimaldigits	The maximum number of decimal places that you want to retain. Default value: 15.
options	The type of coordinate reference system (CRS) information that you want to return in the GML element. Valid values: • 0: specifies to return the CRS of the GML element by using a short expression. An example expression is `EPSG:4326`. • 1: specifies to return the CRS of the GML element by using a long expression. An example expression is `urn:ogc:def:crs:EPSG::4326`. • 2: specifies to delete the srsDimension attribute of the GML element if GML version 3 is used. • 4: specifies to use `<LineString>` rather than `<Curve>` to mark line objects if GML version 3 is used. • 16: specifies that the latitude precedes the longitude in the GML element. For example, if the spatial reference identifier (SRID) is 4326, this value is valid. • 32: specifies to return the bounding box of the geometry object.
geog	The geography object whose GML representation you want to obtain.
version	The GML version that you want to use. The default value 2 specifies GML version 2.1.2. You can set this parameter to 3, which specifies GML version 3.1.1.
nprefix	The prefix of the namespace. Default value: NULL. You can specify a custom prefix.
id	The ID of the GML element. Default value: NULL. This parameter is valid only when GML version 3 is used.

Description

- This function supports circular strings, curves, polyhedral surfaces, triangles, triangulated irregular network (TIN) surfaces, and 3D objects.
- This function supports polyhedral surfaces and TIN surfaces only when GML version 3 or a later

version is used.

Examples

Obtain a GML element that contains the CRS in a long expression, the bounding box, and the coordinates in which the latitude precedes the longitude.

```
SELECT ST_AsGML(3, ST_GeomFromText('POINT(116 40)',4326), 5, 32|16|1);
                            st_asgml
-------------------------------------------------------------
 <gml:Envelope srsName="urn:ogc:def:crs:EPSG::4326" srsDimension="2">
     <gml:lowerCorner>40 116</gml:lowerCorner>
  <gml:upperCorner>40 116</gml:upperCorner>
 </gml:Envelope>
(1 row)
```

4.1.7. ST_AsHEXEWKB

This topic describes the ST_AsHEXEWKB function. This function returns the HEXEWKB string that represents a geometry object.

Syntax

```
text  ST_AsHEXEWKB(geometry  g1 , text  NDRorXDR);
text  ST_AsHEXEWKB(geometry  g1);
```

Parameters

Parameter	Description
g1	The geometry object whose HEXEWKB representation you want to obtain.
NDRorXDR	The byte order that you want to use. Valid values: • XDR: big endian. • NDR: little endian. This is the default value.

Description

- This function supports circular strings and curves.
- This function supports 3D objects and does not discard the z-index of the geometry object that is constructed.

Examples

```
SELECT ST_AsHEXEWKB(ST_GeomFromText('POLYGON((1 1,1 2,2 2,2 1,1 1))',4326));
                          st_ashexewkb
----------------------------------------------------------------
 0103000020E6100000010000000500000000000000000F03F00000000000.
.0F03F000000000000F03F0000000000000040000000000000004000000000.
.0000004000000000000000040000000000000F03F000000000000F03F00000.
.0000000F03F
(1 row)
```

4.1.8. ST_AsKML

This topic describes the ST_AsKML function. This function returns the Keyhole Markup Language (KML) element that represents a geometry object or a geography object.

Syntax

```
text ST_AsKML(geometry geom , integer maxdecimaldigits);
text ST_AsKML(geography geog , integer maxdecimaldigits);
text ST_AsKML(integer version , geometry geom , integer maxdecimaldigits , text nprefi
x);
text ST_AsKML(integer version , geography geog , integer maxdecimaldigits , text npref
ix);
```

Parameters

Parameter	Description
geom	The geometry object whose KML representation you want to obtain.
maxdecimaldigits	The maximum number of decimal places that you want to retain. Default value: 15.
geog	The geography object whose KML representation you want to obtain.
version	The KML version that you want to use. The default value 2 specifies GML version 2.1.2. You can set this parameter to 3, which specifies GML version 3.1.1.
nprefix	The prefix of the namespace. Default value: NULL. You can specify a custom prefix.

Description

This function supports 3D objects and does not discard the z-index of the geometry object that is constructed.

Examples

```
SELECT ST_AsKML(ST_GeomFromText('POLYGON((1 1,1 2,2 2,2 1,1 1))',4326));
                            st_askml
-------------------------------------------------------------
 <Polygon><outerBoundaryIs>
     <LinearRing><coordinates>1,1 1,2 2,.2 2,1 1,1</coordinates>
  </LinearRing></outerBoundaryIs>
 </Polygon>
(1 row)
```

4.1.9. ST_AsLatLonText

This topic describes the ST_AsLatLonText function. This function returns the degrees, minutes, and seconds that represent the coordinates of a point object.

Syntax

```
text  ST_AsLatLonText(geometry  pt , text  format);
```

Parameters

Parameter	Description
pt	The point object whose degree, minute, and second representation you want to obtain.
format	The format in which the degrees, minutes, and seconds are returned. Default value: null. Valid tokens are D, M, S, and C. D represents the degrees, M represents the minutes, S represents the seconds, and C represents the cardinal direction. The cardinal direction can be north, south, east, or west.

Description

- This function assumes that the point object is in a projection that is based on latitude and longitude coordinates. The x coordinate (longitude) and the y coordinate (latitude) are normalized in the return result by using the following normal ranges: the longitude ranges from -90 to +90, and the latitude ranges from -180 to +180.
- Tokens M, S, and C are optional.
  - If you do not specify C, the degrees are displayed as a hyphen (-) when the cardinal direction is south or west.
  - If you do not specify S, the minutes are displayed as a decimal string that consists of as many decimal places as you specify.
  - If you do not specify M, the degrees are displayed as a decimal string that consists of as many decimal places as you specify.

Examples

- Retain the default parameter settings.

```
SELECT ST_AsLatLonText(ST_GeomFromText('POINT(116 40)',4326));
      st_aslatlontext
----------------------------
 40°0'0.000"N 116°0'0.000"E
(1 row)
```

- Generalize the latitude and the longitude.

```
SELECT ST_AsLatLonText(ST_GeomFromText('POINT(476 40)',4326));
      st_aslatlontext
----------------------------
 40°0'0.000"N 116°0'0.000"E
(1 row)
```

- Specify the format parameter.

```
SELECT ST_AsLatLonText(ST_GeomFromText('POINT(116 40)',4326),'D degrees M minutes S.SS se
conds (C)');
           st_aslatlontext
--------------------------------------
 40 degrees 0 minutes 0.00 seconds (N) 116 degrees 0 minutes 0.00 seconds (E)
(1 row)
```

4.1.10. ST_AsSVG

This topic describes the ST_AsSVG function. This function returns the Scalar Vector Graphics (SVG) path data that represents a geometry object or a geography object.

Syntax

```
text  ST_AsSVG(geometry  geom , integer  rel , integer  maxdecimaldigits);
text  ST_AsSVG(geography  geog , integer  rel , integer  maxdecimaldigits);
```

Parameters

Parameter	Description
geom	The geometry object whose SVG representation you want to obtain.
rel	Specifies whether to use relative moves. Default value: 0, which specifies to use absolute moves. You can set this parameter to 1, which specifies to use relative moves to implement path data.
maxdecimaldigits	The maximum number of decimal places that you want to retain. Default value: 15.
geog	The geography object whose SVG representation you want to obtain.

Examples

```
SELECT ST_AsSVG(ST_GeomFromText('POINT(116 40)',4326));
     st_assvg
-------------------
 cx="116" cy="-40"
(1 row)
```

4.1.11. ST_AsText

This topic describes the ST_AsText function. This function returns the Well-Known Text (WKT) string that represents a geometry object or a geography object. The WKT string that is returned does not contain spatial reference identifier (SRID) metadata.

Syntax

```
text  ST_AsText(geometry  g1);
text  ST_AsText(geometry  g1 , integer  maxdecimaldigits);
text  ST_AsText(geography  g1);
text  ST_AsText(geography  g1 , integer  maxdecimaldigits);
```

Parameters

Parameter	Description
g1	The geometry object or geography object whose WKT representation you want to obtain.
maxdecimaldigits	The maximum number of decimal places that you want to retain. Default value: 15.

Description

- If you want to obtain a WKT string that contains an SRID, we recommend that you use the ST_AsEWKT function.
- The WKT format cannot ensure precision. If you want to prevent floating-point numbers of a specific decision from being truncated, we recommend that you use the ST_AsBinary function or the ST_AsEWKB function.

Examples

- Obtain a WKT string with the default parameter settings retained.

```
SELECT ST_AsText(ST_GeomFromText('POINT(116 40)',4326));
   st_astext
---------------
 POINT(116 40)
(1 row)
```

- Obtain a WKT string with the number of decimal places specified.

```
SELECT ST_AsText(ST_GeomFromText('POINT(116.112 40.412)',4326),2);
      st_astext
---------------------
 POINT(116.11 40.41)
(1 row)
```

4.1.12. ST_AsTWKB

This topic describes the ST_AsTWKB function. This function returns the Tiny Well-Known Binary (TWKB) string that represents one or more geometry objects.

Syntax

```
bytea ST_AsTWKB(geometry g1 , integer decimaldigitsXY , integer decimaldigitsZ , integer decimaldigitsM , boolean includeSizes , boolean includeBoundingBoxes);
bytea ST_AsTWKB(geometry[] geometries , bigint[] uniqueIds , integer decimaldigitsXy , integer decimaldigitsZ , integer decimaldigitsM , boolean includeSizes , boolean includeBoundingBoxes);
```

Parameters

Parameter	Description
g1	The geometry object whose TWKB representation you want to obtain.
decimaldigitsXy	The number of decimal places that you want to retain in the x and y coordinates. Default value: 0.
decimaldigitsZ	The number of decimal places that you want to retain in the z coordinate. Default value: 0.
decimaldigitsM	The number of decimal places that you want to retain in the m coordinate. Default value: 0.
includeSizes	Specifies whether to return the size of the geometry object. Default value: false.
includeBoundingBoxes	Specifies whether to return the bounding box of the geometry object. Default value: false.
geometries	An array that consists of the geometry objects whose TWKB representation you want to obtain.
uniqueIds	An array that consists of the unique identifiers of the geometry objects whose TWKB representation you want to obtain.

Examples

```
SELECT ST_AsTWKB(ST_GeomFromText('POINT(116 40)',4326));
       st_astwkb
------------------------
 \xa1008082880b80a4e803
(1 row)
```

4.1.13. ST_AsX3D

This topic describes the ST_AsX3D function. This function returns the Extensible 3D (X3D) string that represents a geometry object.

Syntax

```
text  ST_AsX3D(geometry  g1 , integer  maxdecimaldigits , integer  options);
```

Parameters

Parameter	Description
g1	The geometry object whose X3D representation you want to obtain.
maxdecimaldigits	The maximum number of decimal places that you want to retain. Default value: 15.
options	Specifies whether to represent coordinates with the X3D GeoCoordinates Geospatial node and whether to flip the x and y coordinates. Default value: 0 • 0: specifies to return the x and y coordinates in the order that is used in the database. • 1: specifies to flip the x and y coordinates. By default, if you use this option in conjunction with the GeoCoordinate option, the default setting "latitude_first" is used and the x and y coordinates are flipped. • 2: specifies to return the x and y coordinates in the order that is used in GeoSpatial GeoCoordinates. If the geometry object is not in the WGS 84 coordinate system that contains a spatial reference identifier (SRID) of 4326, this option causes errors. This is the only GeoCoordinate type that is supported. Specify a spatial reference system based on the X3D specification. The default output that is returned is GeoCoordinate geoSystem='"GD" "WE" "longitude_first"'. • 3: specifies to return the X3D string based on the following default settings: GeoCoordinate geoSystem='"GD" "WE" "latitude_first"'.

Description

This function supports circular strings, curves, polyhedral surfaces, triangles, triangulated irregular network (TIN) surfaces, and 3D objects.

Examples

```
select ST_AsX3D( ST_GeomFromEWKT('POLYGON((1 1 0,1 2 0,2 2 0,2 1 0,1 1 0))'));
                            st_asx3d
-------------------------------------------------------------
 <IndexedFaceSet  convex='false' coordIndex='0 1 2 3'>
     <Coordinate point='1 1 0 1 2 0 2 2 0 2 1 0 ' />
 </IndexedFaceSet>
(1 row)
```

4.1.14. ST_GeoHash

This topic describes the GeoHash function. This function returns a Geohash string that represents a geometry object.

Syntax

```
text  ST_GeoHash(geometry  geom , integer  maxchars);
```

Parameters

Parameter	Description
geom	The geometry object whose Geohash representation you want to obtain.
maxchars	The precision based on which the Geohash string is returned. If you do not specify the maxchars parameter, this function returns a Geohash string based on the full precision of the geometry object.

Description

- If the geometry object that you specify is not a point object, this function starts the calculation from the center of the bounding box of the geometry object.
- This function does not support geometry objects that are not in longitude- and latitude-based geographic systems.
- This function supports circular strings and curves.

Examples

- Obtain a Geohash string with the default precision retained.

```
SELECT ST_GeoHash(ST_GeomFromText('POINT(116 40)',4326));
      st_geohash
----------------------
 wx47x9u8gumnhzp791zb
(1 row)
```

- Obtain a Geohash string with a custom precision specified.

```
SELECT ST_GeoHash(ST_GeomFromText('POINT(116 40)',4326),3);
 st_geohash
------------
 wx4
(1 row)
```

4.1.15. ST_AsGeobuf

This topic describes the ST_AsGeobuf function. This function returns a Geobuf string that represents a collection of rows.

Syntax

```
bytea  ST_AsGeobuf(anyelement set  row);
bytea  ST_AsGeobuf(anyelement  row , text  geomName);
```

Parameters

Parameter	Description
row	The rows whose Geobuf representation you want to obtain. The rows must include the data of at least one geometry column.
geomName	The name of the geometry column in the row data. If the value is NULL, this function uses the first geometry column that is found.

Description

Geobuf strings cannot be transmitted as streams.

Examples

```
SELECT ST_AsGeobuf(q) FROM ( select ST_GeomFromText('POINT(116 40)',4326)) as q;
           st_asgeobuf
----------------------------------
 \x1800220b0a090a0708001a03e80150
(1 row)
```

4.2. Editor functions

4.2.1. ST_AddPoint

This topic describes the ST_AddPoint function. This function adds a point to a LineString object.

Syntax

```
geometry  ST_AddPoint(geometry  linestring , geometry  point);
geometry  ST_AddPoint(geometry  linestring , geometry  point , integer  position);
```

Parameters

Parameter	Description
linestring	The geometry object that you want to specify.
point	The point that you want to add.
position	The position to which you want to add the point.

Description

- The position indexes start from 0.
- If the value of the position parameter is -1, the point is added to the end of the LineString object.
- This function supports 3D objects and does not delete z coordinates.

Examples

```
SELECT ST_AsEWKT(ST_AddPoint(ST_GeomFromEWKT('LINESTRING(0 0,0 1)'), ST_MakePoint(1,2)));
        st_asewkt
-------------------------
 LINESTRING(0 0,0 1,1 2)
(1 row)
```

4.2.2. ST_Affine

This topic describes the ST_Affine function. This function applies a 3D affine transformation to a geometry object. After the transformation, you can perform operations, such as translation, rotation, and scaling, on the geometry object at the same time.

Syntax

```
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 xo
ff , float yoff);
```

Parameters

Parameter	Description
geomA	The geometry object that you want to specify.
a	The parameter that is used to specify an affine transformation.
b	The parameter that is used to specify an affine transformation.

Parameter	Description
c	The parameter that is used to specify an affine transformation.
d	The parameter that is used to specify an affine transformation.
e	The parameter that is used to specify an affine transformation.
f	The parameter that is used to specify an affine transformation.
g	The parameter that is used to specify an affine transformation.
h	The parameter that is used to specify an affine transformation.
i	The parameter that is used to specify an affine transformation.
xoff	The parameter that is used to specify an affine transformation.
yoff	The parameter that is used to specify an affine transformation.
zoff	The parameter that is used to specify an affine transformation.

Description

- Method 1: If the expression of the ST_Affine function is ST_Affine(geom, a, b, c, d, e, f, g, h, i, xoff, yoff, zoff), the function represents the following transformation matrix:

```
/ a  b  c  xoff \
| d  e  f  yoff |
| g  h  i  zoff |
\ 0  0  0     1 /
```

- The following expressions show how vertices of the geometry object are transformed. All translation and scaling functions are represented by an affine transformation.

```
x' = a*x + b*y + c*z + xoff
y' = d*x + e*y + f*z + yoff
z' = g*x + h*y + i*z + zoff
```

- Method 2: If the expression of the ST_Affine function is ST_Affine(geom, a, b, d, e, xoff, yoff), the function represents the following transformation matrix:

```
/  a  b  0  xoff  \       /  a  b  xoff  \
|  d  e  0  yoff  | rsp.  |  d  e  yoff  |
|  0  0  1     0  |       \  0  0     1  /
\  0  0  0     1  /
```

- The following expressions show how the vertices of the geometry object are transformed:

```
x' = a*x + b*y + xoff
y' = d*x + e*y + yoff
z' = z
```

  This method is a subcase of the preceding 3D methods.

- This function supports circular strings, curves, polyhedral surfaces, triangles, triangulated irregular network (TIN) surfaces, and 3D objects.

Examples

The following examples show how to rotate a geometry object 180 degrees on the x-axis, y-axis, and z-axis at the same time:

```
SELECT ST_AsEWKT(ST_Affine(ST_GeomFromEWKT('POINT(1 2 3)'),
                           cos(pi()), -sin(pi()), 0, sin(pi()), cos(pi()), -sin(pi()), 0, s
in(pi()), cos(pi()), 0, 0, 0));
    st_asewkt
-----------------
 POINT(-1 -2 -3)
(1 row)
```

4.2.3. ST_Force2D

This topic describes the ST_Force2D function. This function forcefully converts a geometry object into a 2D object.

Syntax

```
geometry  ST_Force2D(geometry  geomA);
```

Parameters

Parameter	Description
geomA	The geometry object that you want to specify.

Description

- The output geometry object contains only the x and y coordinates.
- This function supports circular strings, curves, polyhedral surfaces, triangles, triangulated irregular network (TIN) surfaces, and 3D objects.

Examples

```
SELECT ST_AsEWKT(ST_Force2D(ST_GeomFromEWKT('POINT(1 2 3)')));
 st_asewkt
------------
 POINT(1 2)
(1 row)
```

4.2.4. ST_Force3D

This topic describes the ST_Force3D function. This function forcefully converts a geometry object into a 3D object.

Syntax

```
geometry  ST_Force3D(geometry  geomA);
```

Parameters

Parameter	Description
geomA	The geometry object or that you want to specify.

Description

- The ST_Force3D function is an alias for the ST_Force_3DZ function.
- If the geometry object does not contain z coordinates, the values of z coordinates of the output are specified as 0.
- This function supports circular strings, curves, polyhedral surfaces, and 3D objects.

Examples

```
SELECT ST_AsEWKT(ST_Force3D(ST_GeomFromEWKT('POINT(1 2)')));
  st_asewkt
--------------
 POINT(1 2 0)
(1 row)
```

4.2.5. ST_Force3DZ

This topic describes the ST_Force3DZ function. This function forcefully converts a geometry object into a 3D object.

Syntax

```
geometry  ST_Force3DZ(geometry  geomA);
```

Parameters

Parameter	Description
geomA	The geometry object that you want to specify.

Description

- If the geometry object does not contain z coordinates, the values of z coordinates of the output are specified as 0.
- This function supports circular strings, curves, polyhedral surfaces, and 3D objects.

Examples

```
SELECT ST_AsEWKT(ST_Force3DZ(ST_GeomFromEWKT('POINT(1 2)')));
  st_asewkt
--------------
 POINT(1 2 0)
(1 row)
```

4.2.6. ST_Force3DM

This topic describes the ST_Force3DM function. This function forcefully converts a geometry object into a 3D object.

Syntax

```
geometry  ST_Force3DM(geometry  geomA);
```

Parameters

Parameter	Description
geomA	The geometry object that you want to specify.

Description

- If the input geometry object does not contain m coordinates, the values of m coordinates of the output geometry object are specified as 0.
- If the input geometry object contains z coordinates, z coordinates are removed from the output geometry object.
- This function supports circular strings and curves.

Examples

- Obtain a 3D object by using the default parameter settings.

```
SELECT ST_AsEWKT(ST_Force3DM(ST_GeomFromEWKT('POINT(1 2)')));
   st_asewkt
---------------
 POINTM(1 2 0)
(1 row)
```

- Execute the following statement to obtain a 3D object for which the z coordinates are removed from the input geometry object:

```
SELECT ST_AsEWKT(ST_Force3DM(ST_GeomFromEWKT('POINT(1 2 3)')));
   st_asewkt
---------------
 POINTM(1 2 0)
(1 row)
```

4.2.7. ST_Force4D

This topic describes the ST_Force4D function. This function forcefully converts a geometry object into a 4D object.

Syntax

```
geometry  ST_Force4D(geometry  geomA);
```

Parameters

Parameter	Description
geomA	The geometry object that you want to specify.

Description

- If the input geometry object does not contain z and m coordinates, the values of z and m coordinates of the output geometry object are specified as 0.
- This function supports circular strings and curves.
- This function supports 3D objects and does not delete z coordinates.

Examples

```
SELECT ST_AsEWKT(ST_Force4D(ST_GeomFromEWKT('POINT(1 2 3)')));
   st_asewkt
----------------
 POINT(1 2 3 0)
(1 row)
```

4.2.8. ST_ForcePolygonCCW

This topic describes the ST_ForcePolygonCCW function. This function forces the exterior boundaries of a polygon object to be orientated in the anticlockwise direction and forces the interior boundaries of the polygon object to be orientated in the clockwise direction.

Syntax

```
geometry  ST_ForcePolygonCCW (geometry  geom);
```

Parameters

Parameter	Description
geom	The geometry object that you want to specify.

Description

- This function does not change a non-polygon object.
- This function supports 3D objects and does not delete z coordinates.
- This function supports objects that contain m coordinates.

Examples

```
SELECT ST_AsText(ST_ForcePolygonCCW(ST_GeomFromText('POLYGON((1 1,1 2,2 2,2 1,1 1))')));
           st_astext
-------------------------------
 POLYGON((1 1,2 1,2 2,1 2,1 1))
(1 row)
```

4.2.9. ST_ForceCollection

This topic describes the ST_ForceCollection function. This function converts a geometry object into a GeometryCollection object.

Syntax

```
geometry  ST_ForceCollection(geometry  geomA);
```

Parameters

Parameter	Description
geomA	The geometry object that you want to specify.

Description

- The ST_ForceCollection function is useful for simplifying the Well-Known Binary (WKB) representation.
- This function supports circular strings, curves, and polyhedral surfaces.
- This function supports 3D objects and does not delete z coordinates.

Examples

```
SELECT ST_AsText(ST_ForceCollection(ST_GeomFromText('POLYGON((1 1,2 1,2 2,1 2,1 1))')));
                     st_astext
----------------------------------------------------
 GEOMETRYCOLLECTION(POLYGON((1 1,2 1,2 2,1 2,1 1)))
(1 row)
```

4.2.10. ST_ForcePolygonCW

This topic describes the ST_ForcePolygonCW function. This function forces the exterior boundaries of a MultiPolygon object to be orientated in the clockwise direction and forces the interior boundaries of the MultiPolygon object to be orientated in the anticlockwise direction.

Syntax

```
geometry  ST_ForcePolygonCW (geometry  geom);
```

Parameters

Parameter	Description
geom	The geometry object that you want to specify.

Description

- This function does not change a non-polygon object.
- This function supports 3D objects and does not delete z coordinates.
- This function supports objects that contain m coordinates.

Examples

```
SELECT ST_AsText(ST_ForcePolygonCW(ST_GeomFromText('POLYGON((1 1,2 1,2 2,1 2,1 1))')));
           st_astext
--------------------------------
 POLYGON((1 1,1 2,2 2,2 1,1 1))
(1 row)
```

4.2.11. ST_ForceSFS

This topic describes the ST_ForceSFS function. This function forcefully converts a geometry object to a geometry object with data types defined in the Simple Feature for SQL (SFS) 1.1.

Syntax

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

Parameters

Parameter	Description
geomA	The geometry object that you want to specify.
version	The SFS version.

Description

This function supports circular strings, curves, polyhedral surfaces, triangles, triangulated irregular network (TIN) surfaces, and 3D objects.

Examples

```
SELECT ST_ForceSFS(ST_GeomFromText('POLYGON((1 1,2 1,2 2,1 2,1 1))'));
                          st_forcesfs
----------------------------------------------------------------
 01030000000100000005000000000000000000F03F000000000000F03F000.
.00000000000400000000000000F03F000000000000000400000000000000040.
.000000000000F03F000000000000000400000000000000F03F000000000000F.
.03F
(1 row)
```

4.2.12. ST_ForceRHR

This topic describes the ST_ForceRHR function. This function forces the orientation of vertices of a polygon object to follow the Right-Hand-Rule and the area that is bounded by the polygon object is to the right of boundaries.

Syntax

```
geometry  ST_ForceRHR(geometry  g);
```

Parameters

Parameter	Description
g	The geometry object that you want to specify.

Description

- The exterior rings of the polygon object are orientated in the clockwise direction, and the interior rings of the polygon object are orientated in the anticlockwise direction.
- The ST_ForceRHR function is a synonym for the ST_ForcePolygonCW function.

- This function supports polyhedral surfaces.
- This function supports 3D objects and does not delete z coordinates.

Examples

```
SELECT ST_AsText(ST_ForceRHR(ST_GeomFromText('POLYGON((1 1,2 1,2 2,1 2,1 1))')));
           st_astext
--------------------------------
 POLYGON((1 1,1 2,2 2,2 1,1 1))
(1 row)
```

4.2.13. ST_ForceCurve

This topic describes the ST_ForceCurve function. This function converts a geometry object into its curved representation.

Syntax

```
geometry  ST_ForceCurve(geometry  g);
```

Parameters

Parameter	Description
g	The geometry object that you want to specify.

Description

- The following list describes the conversions of different geometry objects:
  - A line object is converted into a CompoundCurve object.
  - A MultiLines object is converted into a MultiCurve object.
  - A polygon object is converted into a CurvePolygon object.
  - A MultiPolygons is converted into a MultiSurface object.
- If the geometry object that you specified is in curved representation, the ST_ForceCurve function returns the original geometry object.
- This function supports 3D objects and does not delete z coordinates.
- This function supports circular strings and curves.

Examples

```
SELECT ST_AsText(ST_ForceCurve(ST_GeomFromText('POLYGON((1 1,2 1,2 2,1 2,1 1))')));
              st_astext
-------------------------------------
 CURVEPOLYGON((1 1,2 1,2 2,1 2,1 1))
(1 row)
```

4.2.14. ST_LineMerge

This topic describes the ST_LineMerge function. This function returns one or more LineString objects concatenated by a MultiLineString object.

Syntax

```
geometry  ST_LineMerge(geometry  amultilinestring);
```

Parameters

Parameter	Description
amultilinestring	The geometry object that you want to specify.

Description

- The ST_LineMerge function supports MultiLineString objects and LineString objects. If you specify an object of another type, the ST_LineMerge function returns an empty GeometryCollection object.
- This function deletes the m coordinates of MultiLineString objects.

Examples

```
SELECT ST_AsText(ST_LineMerge(ST_GeomFromText('MULTILINESTRING((1 1,2 1),(2 1,2 2))')));
        st_astext
-------------------------
 LINESTRING(1 1,2 1,2 2)
(1 row)
```

4.2.15. ST_CollectionExtract

This topic describes the ST_CollectionExtract function. This function returns one or multiple geometry objects of specified types from an input GeometryCollection object.

Syntax

```
geometry  ST_CollectionExtract(geometry  collection , integer  type);
```

Parameters

Parameter	Description
collection	The geometry object that you want to specify.

Parameter	Description
type	This function supports objects of the following types: • Point • LineString • Polygon

Description

- This function supports point objects, LineString objects, and polygon objects.
- If geometry objects of the specified types are not found in the input Geometrycollection object, an empty geometry object is returned.
- If you specify the type parameter to Polygon, the ST_CollectionExtract function returns a MultiPolygon object even when the edges of objects in the input GeometryCollection object are shared.

> **Note** In most cases, the ST_CollectionExtract function returns an invalid MultiPolygon object. For example, when you apply this function to a collection that is returned by the ST_Split function, the ST_CollectionExtract function returns an invalid MultiPolygon object.

Examples

```
SELECT ST_AsText(ST_CollectionExtract(ST_GeomFromText('GEOMETRYCOLLECTION(LINESTRING(1 1,2 1),LINESTRING(2 1,2 2))'),2));
              st_astext
-------------------------------------
 MULTILINESTRING((1 1,2 1),(2 1,2 2))
(1 row)
```

4.2.16. ST_CollectionHomogenize

This topic describes the ST_CollectionHomogenize function. This function returns the simplest representation of the content of a GeometryCollection object.

Syntax

```
geometry  ST_CollectionHomogenize(geometry  collection);
```

Parameters

Parameter	Description
collection	The geometry object that you want to specify.

Description

- If you specify a collection of singleton objects, the ST_CollectionHomogenize function returns a singleton object.
- If you specify a collection of homogeneous objects, the ST_CollectionHomogenize function returns a MULTI object.

Examples

```
SELECT ST_AsText(ST_CollectionHomogenize(ST_GeomFromText('GEOMETRYCOLLECTION(LINESTRING(1 1
,2 1),LINESTRING(2 1,2 2))')));
              st_astext
-------------------------------------
 MULTILINESTRING((1 1,2 1),(2 1,2 2))
(1 row)
```

4.2.17. ST_Multi

This topic describes the ST_Multi function. This function returns a MULTI geometry object that corresponds to the input geometry object.

Syntax

```
geometry  ST_Multi(geometry  g1);
```

Parameters

Parameter	Description
g1	The geometry object that you want to specify.

Description

If the input geometry object is a MULTI geometry object, the ST_Multi function returns the original geometry object.

Examples

```
SELECT ST_AsText(ST_Multi(ST_GeomFromText('LINESTRING(1 1,2 1)')));
         st_astext
----------------------------
 MULTILINESTRING((1 1,2 1))
(1 row)
```

4.2.18. ST_Normalize

This topic describes the ST_Normalize function. This function returns the normalized form of the input geometry object.

Syntax

```
geometry ST_Normalize(geometry geom);
```

Parameters

Parameter	Description
geom	The geometry object that you want to specify.

Description

The ST_Normalize function may reorder the vertices of rings of a polygon object, rings in a polygon object, and elements in a MULTI geometry object.

Examples

```
SELECT ST_AsText(ST_Normalize(ST_GeomFromText('LINESTRING(2 1,1 1)')));
      st_astext
---------------------
 LINESTRING(1 1,2 1)
(1 row)
```

4.2.19. ST_QuantizeCoordinates

This topic describes the ST_QuantizeCoordinates function. This function returns a geometry object whose coordinate values are displayed in a specific number of significant digits.

Syntax

```
geometry ST_QuantizeCoordinates (geometry g , int precX , int precY , int precZ , int
 precM);
```

Parameters

Parameter	Description
g	The geometry object that you want to specify.
precX	The parameter that is used to specify the decimal places of the values of the x coordinates.
precY	The parameter that is used to specify the decimal places of the values of the y coordinates.
precZ	The parameter that is used to specify the decimal places of the values of the z coordinates.

Parameter	Description
precM	The parameter that is used to specify the decimal places of the values of the m coordinates.

Description

- The ST_QuantizeCoordinates function sets all digits in a coordinate value to 0 except the significant digits.
  - The obtained coordinate value is rounded to the original coordinate value of the geometry object. However, the compressibility of coordinate values is improved. If the columns of the geometry object use a compressible storage type, you can compress the data to reduce the disk usage.
  - The ST_QuantizeCoordinates function does not affect the sizes of the geometry objects that are stored in memory.
- If you do not specify a precision for values of a coordinate, the precision of values of the x coordinates is assumed to be the precision of values of the coordinates.
- If you specify a negative value for a parameter, the significant digits to display in a coordinate value are digits to the left of the decimal point.

  Note If you set the precX parameter to -2, the significant digits to display in a coordinate value are two digits to the left of the decimal point. The ST_QuantizeCoordinates preserves the coordinate value to the nearest 100.
- The ST_QuantizeCoordinates function does not affect existing topological relationships between geometry objects.
- If the number of significant digits lower than the intrinsic precision of a geometry object is called, the ST_QuantizeCoordinates function may return an invalid geometry object

Examples

```
SELECT ST_AsEWKT(ST_QuantizeCoordinates(ST_GeomFromText('POINT(1.123456 1.123456)'),4));
                st_asewkt
-----------------------------------------
 POINT(1.12345123291016 1.12345123291016)
(1 row)
```

4.2.20. ST_RemovePoint

This topic describes the ST_RemovePoint function. This function removes a point with the specified index number from a LineString object.

Syntax

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

Parameters

Parameter	Description
linestring	The geometry object that you want to specify.
offset	The index number.

Description

- The index number starts from 0.
- The ST_RemovePoint function is useful for turning a closed ring into a nonclosed LineString object.
- This function supports 3D objects and does not delete z coordinates.

Examples

```
SELECT ST_AsText(ST_RemovePoint(ST_GeomFromText('LINESTRING(2 1,1 1,1 0)'),0));
      st_astext
---------------------
 LINESTRING(1 1,1 0)
(1 row)
```

4.2.21. ST_Reverse

This topic describes the ST_Reverse function. This function returns a geometry object of the reversed order of the vertices of the input geometry object.

Syntax

```
geometry  ST_Reverse(geometry  g1);
```

Parameters

Parameter	Description
g1	The geometry object that you want to specify.

Description

- The ST_Reverse function can reverse the order of the vertices of all geometry objects.
- This function supports polyhedral surfaces.
- This function supports 3D objects and does not delete z coordinates.

Examples

```
SELECT ST_AsText(ST_Reverse(ST_GeomFromText('LINESTRING(2 1,1 1)')));
      st_astext
---------------------
 LINESTRING(1 1,2 1)
(1 row)
```

4.2.22. ST_Rotate

This topic describes the ST_Rotate function. This function rotates a geometry object by specific radians in the anticlockwise direction.

Syntax

```
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);
```

Parameters

Parameter	Description
geomA	The geometry object that you want to specify.
rotRadians	The radian in the rotation.
x0	The x coordinate of the rotation origin.
y0	The y coordinate of the rotation origin.
pointOrigin	The point object that represents the rotation origin.

Description

- If you do not specify the rotation origin, the point (0 0) point is used as the rotation origin.
- This function supports circular strings, curves, polyhedral surfaces, triangles, triangulated irregular network (TIN) surfaces, and 3D objects.

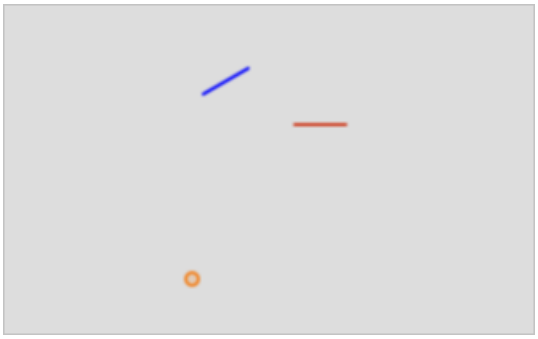
Examples

- Execute the following statement to rotate a geometry object 180 degrees:

```
SELECT ST_AsEWKT(ST_Rotate('LINESTRING (1 2,2 2)'::geometry, pi()));
        st_asewkt
-------------------------
 LINESTRING(-1 -2,-2 -2)
(1 row)
```

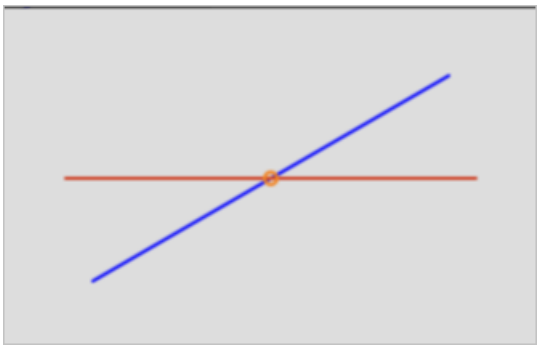
- Execute the following statement to rotate the geometry object 30 degrees in the anticlockwise direction with the (-1,-1) point as the rotation origin:

```
SELECT ST_AsEWKT(ST_Rotate('LINESTRING (1 2,2 2)'::geometry, pi()/6,-1,-1));
                           st_asewkt
-------------------------------------------------------------
 LINESTRING(-0.767949192431122 2.59807621135332,0.098076211353.
.316 3.09807621135332)
(1 row)
```



- Execute the following statement to rotate the geometry object 90 degrees in the clockwise direction with the centroid as the rotation origin.

```
SELECT ST_AsEWKT(ST_Rotate('LINESTRING (1 2,2 2)'::geometry, pi()/6,ST_Centroid('LINESTRI
NG (1 2,2 2)'::geometry)));
                        st_asewkt
-------------------------------------------------------
 LINESTRING(1.06698729810778 1.75,1.93301270189222 2.25)
(1 row)
```



4.2.23. ST_RotateX

This topic describes the ST_RotateX function. This function rotates a geometry object by specific radians about the x-axis.

Syntax

```
geometry  ST_RotateX(geometry  geomA , float  rotRadians);
```

Parameters

Parameter	Description
geomA	The geometry object that you want to specify.
rotRadians	The radian in the rotation.

Description

- The ST_RotateX(geomA,rotRadians) function is shorthand for the ST_Affine(geomA,1,0,0,0,cos(rotRadians),-sin(rotRadians), 0,sin(rotRadians), or cos(rotRadians),0,0,0) function.
- This function supports circular strings, curves, polyhedral surfaces, triangles, triangulated irregular network (TIN) surfaces, and 3D objects.

Examples

```
SELECT ST_AsEWKT(ST_RotateX('LINESTRING (1 2,2 2)'::geometry, pi()));
       st_asewkt
-----------------------
 LINESTRING(1 -2,2 -2)
(1 row)
```

4.2.24. ST_RotateY

This topic describes the ST_RotateY function. This function rotates a geometry object by specific radians about the y-axis.

Syntax

```
geometry  ST_RotateY(geometry  geomA , float  rotRadians);
```

Parameters

Parameter	Description
geomA	The geometry object that you want to specify.

Parameter	Description
rotRadians	The radian in the rotation.

Description

- The ST_RotateY(geomA,rotRadians) function is shorthand for the ST_Affine(geomA,1,0,0,0,cos(rotRadians),-sin(rotRadians), 0,sin(rotRadians), or cos(rotRadians),0,0,0) function.
- This function supports circular strings, curves, polyhedral surfaces, triangles, triangulated irregular network (TIN) surfaces, and 3D objects.

Examples

```
SELECT ST_AsEWKT(ST_RotateY('LINESTRING (1 2,2 2)'::geometry, pi()));
       st_asewkt
-----------------------
 LINESTRING(-1 2,-2 2)
(1 row)
```

4.2.25. ST_RotateZ

This topic describes the ST_RotateZ function. This function rotates a geometry object by specific radians about the z-axis.

Syntax

```
geometry  ST_RotateZ(geometry  geomA , float  rotRadians);
```

Parameters

Parameter	Description
geomA	The geometry object that you want to specify.
rotRadians	The radian in the rotation.

Description

- The ST_RotateZ(geomA,rotRadians) function is shorthand for the ST_Affine(geomA,1,0,0,0,cos(rotRadians),-sin(rotRadians), 0,sin(rotRadians), or cos(rotRadians),0,0,0) function.
- This function supports circular strings, curves, polyhedral surfaces, triangles, triangulated irregular network (TIN) surfaces, and 3D objects.

Examples

```
SELECT ST_AsEWKT(ST_RotateZ('LINESTRING (1 2 1,2 2 1)'::geometry, pi()));
          st_asewkt
-----------------------------
 LINESTRING(-1 -2 1,-2 -2 1)
(1 row)
```

4.2.26. ST_Scale

This topic describes the ST_Scale function. This function scales a geometry object to a new size by multiplying its coordinate values with the corresponding factors.

Syntax

```
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);
```

Parameters

Parameter	Description
geomA/geom	The geometry object that you want to specify.
xFactor	The factor for the values of the x coordinate.
yFactor	The factor for the values of the y coordinate.
zFactor	The factor for the values of the y coordinate.
factor	The scaling factor.
origin	The origin of the scaling.

Description

- You can specify the factor parameter to a 2D, 3DM, 3DZ, or 4D point object. This configuration ensures the scaling factor for all supported dimensions. If the xFactor, yFactor, or zFactor parameter is not specified, the scaling is not performed on the corresponding dimension of the geometry object.
- This function supports circular strings, curves, polyhedral surfaces, triangles, triangulated irregular network (TIN) surfaces, and 3D objects.
- This function supports geometry objects that contain m coordinates.
- If you specify the origin parameter, the ST_Scale function immediately scales the geometry object in place. For example, you can specify the centroid of the geometry object as the scaling origin. If you do not specify the origin parameter, the ST_Scale function scales the geometry object relative to the actual origin of the geometry object. In this case, this function scales the geometry object by multiplying coordinate values by the values of the corresponding factor parameters.

Examples

- Scales a geometry object by using the default parameter settings.

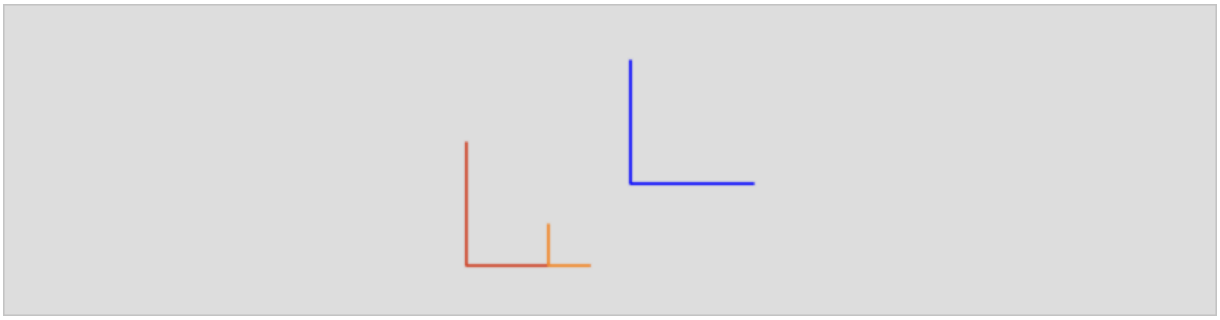
```
SELECT ST_AsText(ST_Scale('LINESTRING(2 1,1 1)'::geometry,2,2));
      st_astext
---------------------
 LINESTRING(4 2,2 2)
(1 row)
```

- Scales a geometry object by specifying the factor parameter.

```
SELECT ST_AsText(ST_Scale('LINESTRING(2 1,1 1)'::geometry,'POINT(2 2)'::geometry));
      st_astext
---------------------
 LINESTRING(4 2,2 2)
(1 row)
```

- Scales a geometry object by specifying the origin parameter.

```
SELECT
    ST_Scale(g,factor),
  ST_Scale(g,factor,'POINT(2 1)'::geometry),g
  from (select 'LINESTRING(2 1,1 1,1 2)'::geometry as g,'POINT(3 3)'::geometry as factor)
as t;
```



4.2.27. ST_Segmentize

This topic describes the ST_Segmentize function. This function returns a geometry or geography object for which the largest segment is no longer than the specified length.

Syntax

```
geometry  ST_Segmentize(geometry  geom , float  maxSegmentLength);
geography  ST_Segmentize(geography  geog , float  maxSegmentLength);
```

Parameters

Parameter	Description
geom	The geometry object that you want to specify.

Parameter	Description
maxSegmentLength	The specified length.
geog	The geography object that you want to specify.

Description

- The ST_Segmentize function calculates lengths only in 2D.
- If you specify a geometry object, the unit of the specified length is in spatial reference.
- If you specify a geography object, the unit of the specified length is meters.
- The ST_Segmentize function does not stretch segments whose length is shorter than the specified length.

Examples

```
SELECT ST_AsText(ST_Segmentize(ST_GeomFromText('LINESTRING(2 1,1 1)'),0.5));
         st_astext
---------------------------
 LINESTRING(2 1,1.5 1,1 1)
(1 row)
```

4.2.28. ST_SetPoint

This topic describes the ST_SetPoint function. This function replaces the points of a LineString object with the specified points. The coordinates of points are specified by indexes.

Syntax

```
geometry  ST_SetPoint(geometry  linestring , integer  zerobasedposition , geometry  point);
```

Parameters

Parameter	Description
linestring	The geometry object that you want to specify.
zerobasedposition	The position of the replacement. The position indexes start from 0.
point	The point that is used for replacement.

Description

- Negative indexes are counted backward, and -1 is the last point.
- The ST_SetPoint function is useful in triggers when you trying to maintain the relationship of joints when one vertex moves.
- This function supports 3D objects and does not delete z coordinates.

Examples

```
SELECT ST_AsText(ST_SetPoint(ST_GeomFromText('LINESTRING(2 1,1 1)'),0,ST_MakePoint(0,1)));
      st_astext
---------------------
 LINESTRING(0 1,1 1)
(1 row)
```

4.2.29. ST_SetSRID

This topic describes the ST_SetSRID function. This function sets a spatial reference identifier (SRID) for a geometry object.

Syntax

```
geometry  ST_SetSRID(geometry  geom , integer  srid);
```

Parameters

Parameter	Description
geom	The geometry object that you want to specify.
srid	The SRID of the geometry object.

Description

- The ST_SetSRID function does not transform the coordinates of the geometry object that you specify. This function sets the metadata that defines the spatial reference system in which the geometry object resides.

  > **Note** If you want to transform the geometry object into a new projection, use the ST_Transform function.

- The ST_SetSRID function supports circular strings and curves.

Examples

```
SELECT ST_AsEWKT(ST_SetSRID(ST_GeomFromText('LINESTRING(2 1,1 1)'),4326));
           st_asewkt
------------------------------
 SRID=4326;LINESTRING(2 1,1 1)
(1 row)
```

4.2.30. ST_Transform

This topic describes the ST_Transform function. This function transforms the original spatial reference system in which a geometry object resides into a new spatial reference system.

Syntax

```
geometry  ST_Transform(geometry  g1 , integer  srid);
geometry  ST_Transform(geometry  geom , text  toProj);
geometry  ST_Transform(geometry  geom , text  fromProj , text  toProj);
geometry  ST_Transform(geometry  geom , text  fromProj , integer  toSrid);
```

Parameters

Parameter	Description
g1/geom	The geometry object that you want to specify.
srid/toSrid	The SRID of the new spatial reference system. The SRID that you specify must be stored in the space_ref_sys table.
toProj	The PROJ string that defines the new spatial reference system.
fromProj	The PROJ string that defines the original spatial reference system.

Description

The ST_Transform function supports circular strings, curves, and polyhedral surfaces.

Examples

```
SELECT ST_AsEWKT(ST_Transform(ST_GeomFromText('LINESTRING(2 1,1 1)',4326),3857));
                           st_asewkt
---------------------------------------------------------------
 SRID=3857;LINESTRING(222638.981586547 111325.142866385,111319.
..490793274 111325.142866385)
(1 row)
```

4.2.31. ST_Translate

This topic describes the ST_Translate function. This function returns a new geometry object whose coordinates are offset by the specified units.

Syntax

```
geometry ST_Translate(geometry  g1 , float  deltax , float  deltay);
geometry  ST_Translate(geometry  g1 , float  deltax , float  deltay , float  deltaz);
```

Parameters

Parameter	Description
g1	The geometry object that you want to specify.
deltax	The offset units of the delta x.
deltay	The offset units of the delta y.
deltaz	The offset units of the delta z.

Description

- The specified units are based on the units that are defined in spatial reference identifier (SRID) of the geometry object.
- The ST_Translate function supports 3D objects and does not delete z coordinates.
- The ST_Translate function supports circular strings and curves.

Examples

```
SELECT ST_AsEWKT(ST_Translate(ST_GeomFromText('LINESTRING(2 1,1 1)',4326),1,1));
           st_asewkt
-------------------------------
 SRID=4326;LINESTRING(3 2,2 2)
(1 row)
```

4.2.32. ST_TransScale

This topic describes the ST_TransScale function. This function returns a new object whose coordinates values are offset by specified units and scaled by specified factors.

Syntax

```
geometry  ST_TransScale(geometry  geomA , float  deltaX , float  deltaY , float  xFactor ,
float  yFactor);
```

Parameters

Parameter	Description
geomA	The geometry object that you want to specify.
deltaX	The offset units of the delta x.
deltaY	The offset units of the delta y.
xFactor	The scale factor of the delta x.
yFactor	The scale factor of the delta y.

Description

- The ST_TransScale function returns a new object whose coordinates values are offset by specified units and scaled by specified factors.
- The ST_TransScale function is available only for 2D objects.
- The ST_TransScale function supports 3D objects and does not delete z coordinates.
- The ST_TransScale function supports circular strings and curves.

Examples

```
SELECT ST_AsEWKT(ST_TransScale(ST_GeomFromText('LINESTRING(2 1,1 1)',4326),1,1,2,2));
           st_asewkt
------------------------------
 SRID=4326;LINESTRING(6 4,4 4)
(1 row)
```

4.3. Geometry constructors

4.3.1. ST_BdPolyFromText

This topic describes the ST_BdPolyFromText function. This function constructs a polygon object by using one or more closed linestrings that are represented by a Well-Known Text (WKT) string.

Syntax

```
geometry  ST_BdPolyFromText(text  WKT , integer  srid);
```

Parameters

Parameter	Description
WKT	The WKT string that you want to specify.
srid	The spatial reference identifier (SRID) of the polygon object.

Description

- If the WKT string that you specify does not represent a MultiLineString object, this function returns an error.
- If this function obtains a MultiPolygon object by using the WKT string that you specify, this function returns an error. In this case, use the method that is used in the ST_BdMPolyFromText function or the ST_BuildArea function to specify the WKT string.

Examples

Construct a WKT string that represents a MultiLineString object and convert the WKT string into a polygon object.

```
 SELECT st_AsText(ST_BdPolyFromText(ST_AsText(ST_GeomFromText('MultiLineString((1 1,1 2),(2
2,1 1),(1 2,2 2))')),4326));
         st_astext
----------------------------
 POLYGON((1 1,1 2,2 2,1 1))
(1 row)
```

4.3.2. ST_BdMPolyFromText

This topic describes the ST_BdMPolyFromText function. This function constructs a MultiPolygon object by using one or more closed linestrings, polygons, or MultiLineStrings that are represented by a Well-Known Text (WKT) string.

Syntax

```
geometry  ST_BdMPolyFromText(text  WKT , integer  srid);
```

Parameters

Parameter	Description
WKT	The WKT string that you want to specify.
srid	The spatial reference identifier (SRID) of the MultiPolygon object.

Description

- If the WKT string that you specify does not represent a MultiLineString object, this function returns an error.
- This function returns a MultiPolygon object even if you specify a single polygon.

> **Note** If you want to construct a MultiPolygon object by using a single polygon, we recommend that you use the ST_BdPolyFromText function.

Examples

Construct a WKT string that represents a MultiLineString object and convert the WKT string into a MultiPolygon object.

```
SELECT st_AsText(ST_BdMPolyFromText('MultiLineString((1 1,1 2),(2 2,1 1),(1 2,2 2),(-1 -1,-
1 -2),(-2 -2,-1 -1),(-1 -2,-2 -2))',4326));
                            st_astext
-----------------------------------------------------------------
 MULTIPOLYGON(((-1 -1,-1 -2,-2 -2,-1 -1)),((1 1,1 2,2 2,1 1)))
(1 row)
```

4.3.3. ST_Box2dFromGeoHash

This function describes the ST_Box2dFromGeoHash function. This function constructs a Box2D object by using a Geohash string.

Syntax

```
box2d  ST_Box2dFromGeoHash(text  geohash , integer  precision);
```

Parameters

Parameter	Description
geohash	The Geohash string that you want to specify.
precision	The precision based on which you want to construct the Box2D object. If you do not specify this parameter, this function constructs the Box2D object by using the Geohash string in the full precision.

Examples

- Construct a Box2D object by using a Geohash string in the full precision.

```
SELECT ST_Box2dFromGeoHash('wx47x9u8gumnhzp791zb');
       st_box2dfromgeohash
----------------------------------
 BOX(116 39.9999999999999,116 40)
(1 row)
```

- Construct a Box2D object by using a Geohash string in a custom precision.

```
SELECT ST_Box2dFromGeoHash('wx47x9u8gumnhzp791zb',5);
                    st_box2dfromgeohash
-------------------------------------------------------------
 BOX(115.9716796875 39.990234375,116.015625 40.0341796875)
(1 row)
```

4.3.4. ST_GeogFromText

This topic describes the ST_GeogFromText function. This function constructs a geography object by using a standard Well-Known Text (WKT) string or an Extended Well-Known Text (EWKT) string.

Syntax

```
geography  ST_GeogFromText(text  eWKT);
```

Parameters

Parameter	Description
eWKT	The standard WKT string or EWKT string that you want to specify.

Description

- If you do not specify a spatial reference identifier (SRID), the default SRID 4326 is used.
- This function is equivalent to the ST_GeographyFromText function.
- The coordinates of a point follow the order in which the longitude precedes the latitude.

Examples

- Construct a geography object by using the default SRID 4326.

```
SELECT ST_SRID(ST_GeogFromText('POINT(116 40)'));
 st_srid
---------
    4326
(1 row)
```

- Construct a geography object by using a specified SRID.

```
SELECT ST_SRID(ST_GeogFromText('SRID=4256;POINT(116 40)'));
 st_srid
---------
    4256
(1 row)
```

4.3.5. ST_GeographyFromText

This topic describes the ST_GeographyFromText function. This function constructs a geography object by using a standard Well-Known Text (WKT) string or an Extended Well-Known Text (EWKT) string.

Syntax

```
geography  ST_GeographyFromText(text  eWKT);
```

Parameters

Parameter	Description
eWKT	The standard WKT string or EWKT string that you want to specify.

Description

- If you do not specify a spatial reference identifier (SRID), the default SRID 4326 is used.
- The coordinates of a point follow the order in which the longitude precedes the latitude.

Examples

- Construct a geography object by using the default SRID 4326.

```
SELECT ST_SRID(ST_GeographyFromText('POINT(116 40)'));
 st_srid
---------
    4326
(1 row)
```

- Construct a geography object by using a specified SRID.

```
SELECT ST_SRID(ST_GeographyFromText('SRID=4256;POINT(116 40)'));
 st_srid
---------
    4256
(1 row)
```

4.3.6. ST_GeogFromWKB

This topic describes the ST_GeogFromWKB function. This function constructs a geography object by using a standard Well-Known Binary (WKB) string or an Extended Well-Known Binary (EWKB) string.

Syntax

```
geography  ST_GeogFromWKB(bytea  wkb);
```

Parameters

Parameter	Description
wkb	The standard WKB string or EWKB string that you want to specify.

Description

- If you do not specify a spatial reference identifier (SRID), the default SRID 4326 is used.
- This function supports circular strings and curves.

Examples

```
SELECT ST_AsText(ST_GeogFromWKB(E'\\x0101000000000000000000005d400000000000004440'));
   st_astext
---------------
 POINT(116 40)
(1 row)
```

4.3.7. ST_GeomFromTWKB

This topic describes the ST_GeomFromTWKB function. This function constructs a geometry object by using a Tiny Well-Known Binary (TWKB) string.

Syntax

```
geometry  ST_GeomFromTWKB(bytea  twkb);
```

Parameters

Parameter	Description
twkb	The TWKB string that you want to specify.

Examples

```
SELECT ST_AsText(ST_GeomFromTWKB(E'\\x0100e80150'));
   st_astext
---------------
 POINT(116 40)
(1 row)
```

4.3.8. ST_GeomCollFromText

This topic describes the ST_GeomCollFromText function. This function constructs a GeometryCollection object by using a Well-Known Text (WKT) string and a spatial reference identifier (SRID).

Syntax

```
geometry  ST_GeomCollFromText(text  WKT , integer  srid);
geometry  ST_GeomCollFromText(text  WKT);
```

Parameters

Parameter	Description
WKT	The WKT string that you want to specify.
srid	The SRID of the GeometryCollection object.

Description

- If you do not specify an SRID, the default SRID 0 is used.
- If the WKT string that you specify does not represent a GeometryCollection object, this function returns an error.
- If you cannot make sure that the WKT string that you specify represents a GeometryCollection object,

do not use this function. This function requires additional verification steps. As a result, this function constructs geometry objects at lower speeds than the ST_GeomFromText function.

Examples

```
SELECT ST_AsText(ST_GeomCollFromText('GEOMETRYCOLLECTION(POINT(1 2),LINESTRING(3 4, 5 6))')
);
                     st_astext
----------------------------------------------------
 GEOMETRYCOLLECTION(POINT(1 2),LINESTRING(3 4,5 6))
(1 row)
```

4.3.9. ST_GeomFromEWKB

This topic describes the ST_GeomFromEWKB function. This function constructs a geometry object by using an Extended Well-Known Binary (EWKB) string.

Syntax

```
geometry  ST_GeomFromEWKB(bytea  eWKB);
```

Parameters

Parameter	Description
eWKB	The EWKB string that you want to specify.

Description

This function supports circular strings, curves, polyhedral surfaces, triangles, triangulated irregular network (TIN) surfaces, and 3D objects.

Examples

Construct a geometry object by using the default parameter settings and specify that the spatial reference identifier (SRID) must be returned.

```
SELECT ST_AsText(ST_GeomFromEWKB(E'\\x0101000020e6100000000000000000005d400000000000004440'))
;
   st_astext
---------------
 POINT(116 40)
(1 row)
SELECT ST_SRID(ST_GeomFromEWKB(E'\\x0101000020e6100000000000000000005d400000000000004440'));
 st_srid
---------
    4326
(1 row)
```

4.3.10. ST_GeomFromEWKT

This topic describes the ST_GeomFromEWKT function. This function constructs a geometry object by using an Extended Well-Known Text (EWKT) string.

Syntax

```
geometry  ST_GeomFromEWKT(text  eWKT);
```

Parameters

Parameter	Description
eWKT	The EWKT string that you want to specify.

Description

This function supports circular strings, curves, polyhedral surfaces, triangles, triangulated irregular network (TIN) surfaces, and 3D objects.

Examples

Construct a CircularString object.

```
SELECT ST_GeomFromEWKT('CIRCULARSTRING(200000 150000 0,21000 160000 0,220000 170000 1)');
                        st_geomfromewkt
----------------------------------------------------------------
 0108000080030000000000000000006A08410000000804F024100000000000.
.0000000000000082D4400000000000880341000000000000000000000000.
.00DB0A410000000080C00441000000000000F03F
(1 row)
```

4.3.11. ST_GeometryFromText

This topic describes the ST_GeometryFromText function. This function constructs a geometry object by using a Well-Known Text (WKT) string.

Syntax

```
geometry  ST_GeometryFromText(text  WKT);
geometry  ST_GeometryFromText(text  WKT , integer  srid);
```

Parameters

Parameter	Description
WKT	The WKT string that you want to specify.

Parameter	Description
srid	The spatial reference identifier (SRID) of the geometry object.

Description

- This function supports circular strings and curves.
- This function is equivalent to the ST_GeomFromText function.

Examples

- Construct a geometry object by using the default parameter settings.

```
SELECT ST_GeomFromText('POINT(116 40)');
              st_geomfromtext
--------------------------------------------
 0101000000000000000005D400000000000004440
(1 row)
```

- Construct a geometry object by using a specified SRID.

```
SELECT ST_GeomFromText('POINT(116 40)',4326);
                  st_geomfromtext
----------------------------------------------------
 0101000020E6100000000000000005D400000000000004440
(1 row)
```

4.3.12. ST_GeomFromText

This topic describes the ST_GeomFromText function. This function constructs a geometry object by using a Well-Known Text (WKT) string.

Syntax

```
geometry  ST_GeomFromText(text  WKT);
geometry  ST_GeomFromText(text  WKT , integer  srid);
```

Parameters

Parameter	Description
WKT	The WKT string that you want to specify.
srid	The spatial reference identifier (SRID) of the geometry object.

Description

This function supports circular strings and curves.

Examples

- Construct a geometry object by using the default parameter settings.

```
SELECT ST_GeomFromText('POINT(116 40)');
              st_geomfromtext
--------------------------------------------
 0101000000000000000000005D400000000000004440
(1 row)
```

- Construct a geometry object by using a specified SRID.

```
SELECT ST_GeomFromText('POINT(116 40)',4326);
                  st_geomfromtext
----------------------------------------------------
 0101000020E6100000000000000000005D400000000000004440
(1 row)
```

4.3.13. ST_GeomFromGeoHash

This topic describes the ST_GeomFromGeoHash function. This function constructs a geometry object by using a Geohash string.

Syntax

```
geometry  ST_GeomFromGeoHash(text  geohash , integer  precision);
```

Parameters

Parameter	Description
geohash	The Geohash string that you want to specify.
precision	The precision based on which you want to construct the geometry object. If you do not specify a precision, this function returns a polygon by using the Geohash string in the full precision.

Examples

- Construct a geometry object by using the full precision.

```
SELECT ST_AsText(ST_GeomFromGeoHash('wx47x9u8gumnhzp791zb'));
                           st_astext
-------------------------------------------------------------
 POLYGON((116 39.9999999999999,116 40,116 40,116 39.9999999999.
.999,116 39.9999999999999))
(1 row)
```

- Construct a geometry object by using a custom precision.

```
SELECT ST_AsText(ST_GeomFromGeoHash('wx47x9u8gumnhzp791zb',2));
                            st_astext
-------------------------------------------------------------
 POLYGON((112.5 39.375,112.5 45,123.75 45,123.75 39.375,112.5 .
.39.375))
(1 row)
```

4.3.14. ST_GeomFromGML

This topic describes the ST_GeomFromGML function. This function constructs a geometry object by using a Geography Markup Language (GML) string.

Syntax

```
geometry  ST_GeomFromGML(text  geomgml);
geometry  ST_GeomFromGML(text  geomgml , integer  srid);
```

Parameters

Parameter	Description
geomgml	The GML string that you want to specify.
srid	The spatial reference identifier (SRID) of the geometry object.

Description

- This function supports only GML geometry fragments. If you use this function on an entire GML document, this function returns an error.
- GML supports mixed dimensions. This function does not support mixed dimensions. If no z coordinate is found, this function converts the geometry object that is constructed into a 2D object.
- GML supports mixed spatial reference systems (SRSs). This function does not support mixed SRSs. This function reprojects all sub-geometries to the SRS root node. If no srsName attribute is available for the GML root node, this function returns an error.
- This function supports polyhedral surfaces, triangles, triangulated irregular network (TIN) surfaces, and 3D objects.

Examples

```
SELECT ST_AsText(ST_GeomFromGML('<gml:Point srsName="EPSG:4326"><gml:coordinates>116,40</gm
l:coordinates></gml:Point>'));
   st_astext
---------------
 POINT(116 40)
(1 row)
```

4.3.15. ST_GeomFromGeoJSON

This topic describes the ST_GeomFromGeoJSON function. This function constructs a geometry object by using a GeoJSON string.

Syntax

```
geometry  ST_GeomFromGeoJSON(text  geomjson);
geometry  ST_GeomFromGeoJSON(json  geomjson);
geometry  ST_GeomFromGeoJSON(jsonb  geomjson);
```

Parameters

Parameter	Description
geomjson	The GeoJSON string that you want to specify.

Description

- This function supports only GeoJSON geometry fragments. If you use this function on an entire GeoJSON document, this function returns an error.
- This function supports 3D objects and does not discard the z-index of the geometry object that is constructed.

Examples

```
SELECT ST_AsText(ST_GeomFromGeoJSON('{"type":"Point","coordinates":[116,40]}'));
   st_astext
---------------
 POINT(116 40)
(1 row)
```

4.3.16. ST_GeomFromKML

This topic describes the ST_GeomFromKML function. This function constructs a geometry object by using an Open Geospatial Consortium (OGC) Keyhole Markup Language (KML) string.

Syntax

```
geometry  ST_GeomFromKML(text  geomkml);
```

Parameters

Parameter	Description
geomkml	The KML string that you want to specify.

Description

- This function supports only KML geometry fragments. If you use this function on an entire KML document, this function returns an error.
- This function supports 3D objects and does not discard the z-index of the geometry object that is constructed.

Examples

```
SELECT ST_AsText(ST_GeomFromKML('<Point><coordinates>116,40</coordinates></Point>'));
   st_astext
---------------
 POINT(116 40)
(1 row)
```

4.3.17. ST_GMLToSQL

This topic describes the ST_GMLToSQL function. This function constructs a geometry object by using an Open Geospatial Consortium (OGC) Geography Markup Language (GML) string.

Syntax

```
geometry  ST_GMLToSQL(text  geomgml);
geometry  ST_GMLToSQL(text  geomgml , integer  srid);
```

Parameters

Parameter	Description
geomkml	The Keyhole Markup Language (KML) string that you want to specify.
srid	The spatial reference identifier (SRID) of the geometry object.

Description

- This function supports only GML geometry fragments. If you use this function on an entire GML document, this function returns an error.
- GML supports mixed dimensions. This function does not support mixed dimensions. If no z coordinate is found, this function converts the geometry object that is constructed into a 2D object.
- GML supports mixed spatial reference systems (SRSs). This function does not support mixed SRSs. This function reprojects all sub-geometries to the SRS root node. If no srsName attribute is available for the GML root node, this function returns an error.
- This function supports polyhedral surfaces, triangles, triangulated irregular network (TIN) surfaces, and 3D objects.

Examples

```
SELECT ST_AsText(ST_GMLToSQL('<Point><coordinates>116,40</coordinates></Point>'));
   st_astext
---------------
 POINT(116 40)
(1 row)
```

4.3.18. ST_GeomFromWKB

This topic describes the ST_GeomFromWKB function. This function constructs a geometry object by using a Well-Known Binary (WKB) string and optionally a spatial reference identifier (SRID).

Syntax

```
geometry  ST_GeomFromWKB(bytea  geom);
geometry  ST_GeomFromWKB(bytea  geom , integer  srid);
```

Parameters

Parameter	Description
geom	The WKB string that you want to specify.
srid	The spatial reference identifier (SRID) of the geometry object.

Description

- If you do not specify an SRID, the default SRID 0 is used.
- This function supports circular strings and curves.

Examples

```
SELECT ST_AsText(ST_GeomFromWKB(E'\\x0101000000000000000000005d400000000000004440'));
   st_astext
---------------
 POINT(116 40)
(1 row)
```

4.3.19. ST_LineFromEncodedPolyline

This topic describes the ST_LineFromEncodedPolyline function. This function constructs a LineString object by using an encoded string that represents a polyline.

Syntax

```
geometry  ST_LineFromEncodedPolyline(text  polyline , integer  precision);
```

Parameters

Parameter	Description
polyline	The encoded string that represents the polyline.
precision	The precision that specifies how many decimal places in the polyline are retained. The precision of encoding and the precision of decoding must be the same. Otherwise, coordinates are inaccurate.

Description

For more information about the encoding format that is used by this function, see Encoded Polyline Algorithm Format.

Examples

```
SELECT ST_AsText(ST_LineFromEncodedPolyline('_gvzE_ol{U_glW_c`|@_}hQ~flW'));
            st_astext
----------------------------------
 LINESTRING(120 36,130 40,126 43)
(1 row)
```

4.3.20. ST_LineFromMultiPoint

This topic describes the ST_LineFromMultiPoint function. This function constructs a LineString object by using a MultiPoint object.

Syntax

```
geometry  ST_LineFromMultiPoint(geometry  aMultiPoint);
```

Parameters

Parameter	Description
aMultiPoint	The MultiPoint object that you want to specify.

Examples

```
SELECT ST_AsText(ST_LineFromMultiPoint(ST_GeomFromEWKT('MULTIPOINT(1 2,3 4,5 6)')));
        st_astext
-------------------------
 LINESTRING(1 2,3 4,5 6)
(1 row)
```

4.3.21. ST_LineFromText

This topic describes the ST_LineFromText function. This function constructs a LineString object by using a Well-Known Text (WKT) string and a spatial reference identifier (SRID).

Syntax

```
geometry  ST_LineFromText(text  wKT);
geometry  ST_LineFromText(text  wKT , integer  srid);
```

Parameters

Parameter	Description
WKT	The WKT string that you want to specify.
srid	The SRID of the LineString object.

Description

- If you do not specify an SRID, the default SRID 0 is used.
- If the WKT string that you specify does not represent a LineString object, this function returns NULL.

Examples

```
SELECT ST_AsText(ST_LineFromText('LINESTRING(1 2, 3 4)'));
      st_astext
---------------------
 LINESTRING(1 2,3 4)
(1 row)
```

4.3.22. ST_LineFromWKB

This topic describes the ST_LineFromWKB function. This function constructs a geometry object by using a Well-Known Binary (WKB) string and a spatial reference identifier (SRID).

Syntax

```
geometry  ST_LineFromWKB(bytea  WKB);
geometry  ST_LineFromWKB(bytea  WKB , integer  srid);
```

Parameters

Parameter	Description
WKB	The WKB string that you want to specify.
srid	The SRID of the geometry object.

Description

- If you do not specify an SRID, the default SRID 0 is used.
- If the WKB string that you specify does not represent a LineString object, this function returns NULL.
- If you can make sure that the WKB string you specify represents a LineString object, we recommend that you use the ST_GeomFromWKB function, which constructs a geometry object in a more efficient manner than the ST_LineFromWKB function.

Examples

```
SELECT ST_AsText(ST_LineFromWKB(E'\\x0102000000020000000000000000000f03f0000000000000040000000000000084000000000000001040'));
      st_astext
---------------------
 LINESTRING(1 2,3 4)
(1 row)
```

4.3.23. ST_LinestringFromWKB

This topic describes the ST_LinestringFromWKB function. This function constructs a LineString object by using a Well-Known Binary (WKB) string and a spatial reference identifier (SRID).

Syntax

```
geometry  ST_LinestringFromWKB(bytea  WKB);
geometry  ST_LinestringFromWKB(bytea  WKB , integer  srid);
```

Parameters

Parameter	Description
WKB	The WKB string that you want to specify.
srid	The SRID of the LineString object.

Description

- If you do not specify an SRID, the default SRID 0 is used.
- If the WKB string that you specify does not represent a LineString object, this function returns NULL.
- If you can make sure that the WKB string you specify represents a LineString object, we recommend that you use the ST_GeomFromWKB function, which constructs a geometry object in a more efficient manner than the ST_LinestringFromWKB function.
- This function is equivalent to the ST_LineFromWKB function.

Examples

```
SELECT ST_AsText(ST_LineFromWKB(E'\\x0102000000020000000000000000000f03f0000000000000040000000000000084000000000000001040'));
      st_astext
--------------------
 LINESTRING(1 2,3 4)
(1 row)
```

4.3.24. ST_MakeBox2D

This topic describes the ST_MakeBox2D function. This function constructs a Box2D object by using a point in the lower-left corner and a point in the upper-right corner.

Syntax

```
box2d  ST_MakeBox2D(geometry  pointLowLeft , geometry  pointUpRight);
```

Parameters

Parameter	Description
pointLowLeft	The point in the lower-left corner.
pointUpRight	The point in the upper-right corner.

Description

This function is suitable for range queries.

Examples

```
SELECT ST_MakeBox2D(ST_Point(115, 39), ST_Point(116, 40));
    st_makebox2d
--------------------
 BOX(115 39,116 40)
(1 row)
```

4.3.25. ST_3DMakeBox

This topic describes the ST_3DMakeBox function. This function constructs a Box3D object by using a point in the lower-left corner and a point in the upper-right corner.

Syntax

```
box3d  ST_3DMakeBox(geometry  point3DLowLeftBottom , geometry  point3DUpRightTop);
```

Parameters

Parameter	Description
point3DLowLeftBottom	The point in the lower-left corner.
point3DUpRightTop	The point in the upper-right corner.

Description

This function supports 3D objects and does not discard the z-index of the geometry object that is constructed.

Examples

```
SELECT ST_3DMakeBox(ST_MakePoint(115, 39, 10), ST_MakePoint(116, 40, 10));
        st_3dmakebox
----------------------------
 BOX3D(115 39 10,116 40 10)
(1 row)
```

4.3.26. ST_MakeLine

This topic describes the ST_MakeLine function. This function constructs a LineString object by using geometry objects, such as points, MultiPoints, or lines.

Syntax

```
geometry  ST_MakeLine(geometry  set geoms);
geometry  ST_MakeLine(geometry  geom1 , geometry  geom2);
geometry  ST_MakeLine(geometry[]  geomsArray);
```

Parameters

Parameter	Description
geoms	The geometry objects that you want to specify.
geom1	The first geometry object that you want to specify.
geom2	The second geometry object that you want to specify.
geomsArray	An array that consists of the geometry objects you want to specify.

Description

- If you specify geometry objects that are not points, MultiPoints, or lines, this function ignores the specified geometry objects.
- When a LineString object is added, this function deletes the common nodes at the beginning of lines.

If you specify points or MultiPoints, this function does not delete the common nodes at the beginning of points or MultiPoints.

- This function supports 3D objects and does not discard the z-index of the geometry object that is constructed.

Examples

- Construct a LineString object by using the default parameter settings.

```
SELECT ST_AsText(ST_MakeLine(ST_MakePoint(1,2), ST_MakePoint(3,4)));
      st_astext
--------------------
 LINESTRING(1 2,3 4)
(1 row)
```

- Construct a 3D LineString object.

```
SELECT ST_AsText(ST_MakeLine(ST_MakePoint(1,2,3), ST_MakePoint(4,5,6)));
         st_astext
----------------------------
 LINESTRING Z (1 2 3,4 5 6)
(1 row)
```

- Construct a LineString object by using an array.

```
SELECT ST_AsText(ST_MakeLine(ARRAY[ST_MakePoint(1,2),ST_MakePoint(3,4), ST_MakePoint(5,6)
]));
        st_astext
-------------------------
 LINESTRING(1 2,3 4,5 6)
(1 row)
```

4.3.27. ST_MakeEnvelope

This topic describes the ST_MakeEnvelope function. This function constructs a polygon object by using the minimum values and maximum values of coordinates.

Syntax

```
geometry  ST_MakeEnvelope(double precision  xmin , double precision  ymin , double precisio
n  xmax , double precision  ymax , integer  srid);
```

Parameters

Parameter	Description
xmin	The minimum value of the x coordinate.
ymin	The minimum value of the y coordinate.

Parameter	Description
xmax	The maximum value of the x coordinate.
ymax	The maximum value of the y coordinate.
srid	The spatial reference identifier (SRID) of the polygon object. Default value: 0, which indicates that the SRID is unknown.

Description

The maximum values and minimum values that you specify must be within the value ranges that are supported by the specified SRID.

Examples

```
SELECT ST_AsText(ST_MakeEnvelope(1, 2, 3, 4, 4326));
           st_astext
--------------------------------
 POLYGON((1 2,1 4,3 4,3 2,1 2))
(1 row)
```

4.3.28. ST_MakePolygon

This topic describes the ST_MakePolygon function. This function constructs a polygon object by using shells.

Syntax

```
geometry  ST_MakePolygon(geometry  linestring);
geometry  ST_MakePolygon(geometry  outerlinestring , geometry[]  interiorlinestrings);
```

Parameters

Parameter	Description
linestring	The LineString object that represents the shell.
outerlinestring	The LineString object that represents the outer shell.
interiorlinestrings	The LineString object that represents the inner shell.

Description

You can specify only a closed LineString.

Examples

- Construct a polygon object by using the default parameter settings.

```
SELECT ST_AsText(ST_MakePolygon(ST_GeomFromText('LINESTRING(1 2,3 4,5 6,1 2)')));
         st_astext
----------------------------
 POLYGON((1 2,3 4,5 6,1 2))
(1 row)
```

- Construct a polygon object that has an inner shell.

```
SELECT ST_AsText(ST_MakePolygon(ST_GeomFromText('LINESTRING(0 0,0 1,1 1,0 0)'),ARRAY[ST_G
eomFromText('LINESTRING(-1 -1,-1 2,2 2,-1 -1)')]));
                     st_astext
---------------------------------------------------
 POLYGON((0 0,0 1,1 1,0 0),(-1 -1,-1 2,2 2,-1 -1))
(1 row)
```

4.3.29. ST_MakePoint

This topic describes the ST_MakePoint function. This function constructs a 2D, 3DZ, or 4D point object.

Syntax

```
geometry  ST_MakePoint(double precision  x , double precision  y);
geometry  ST_MakePoint(double precision  x , double precision  y , double precision  z);
geometry  ST_MakePoint(double precision  x , double precision  y , double precision  z , do
uble precision  m);
```

Parameters

Parameter	Description
x	The x coordinate, which represents the longitude of the point object.
y	The y coordinate, which represents the latitude of the point object.
z	The z coordinate of the point object.
m	The m coordinate of the point object.

Description

- The ST_MakePoint function constructs a geometry object in a faster and more accurate manner than the ST_GeomFromText function and the ST_PointFromText function.
- If you want to construct a 3DM point object, use the ST_MakePointM function.
- The ST_MakePoint function supports 3D objects and does not discard the z-index of the geometry object that is constructed.

Examples

```
SELECT ST_AsText(ST_MakePoint(1,2));
 st_astext
------------
 POINT(1 2)
(1 row)
```

- Construct a 3DZ point object.

```
SELECT ST_AsText(ST_MakePoint(1,2,3));
    st_astext
-----------------
 POINT Z (1 2 3)
(1 row)
```

- Create a 4D object.

```
SELECT ST_AsText(ST_MakePoint(1,2,3,4));
     st_astext
--------------------
 POINT ZM (1 2 3 4)
(1 row)
```

4.3.30. ST_MakePointM

This topic describes the ST_MakePointM function. This function constructs a PointM object by using x, y, and m coordinates.

Syntax

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

Parameters

Parameter	Description
x	The x coordinate, which specifies the longitude of the PointM object.
y	The y coordinate, which specifies the latitude of the PointM object.
m	The m coordinate of the PointM object.

Examples

```
SELECT ST_AsEWKT(ST_MakePointM(1,2,3));
   st_asewkt
---------------
 POINTM(1 2 3)
(1 row)
```

4.3.31. ST_MPointFromText

This topic describes the ST_MPointFromText function. This function constructs a MultiPoint object by using a Well-Known Text (WKT) string and a spatial reference identifier (SRID).

Syntax

```
geometry  ST_MPointFromText(text  wKT , integer  srid);
geometry  ST_MPointFromText(text  wKT);
```

Parameters

Parameter	Description
wKT	The WKT string that you want to specify.
srid	The SRID of the MultiPoint object.

Description

- If you can make sure that the information that you specify can be used to construct a MultiPoint object, we recommend that you use the ST_GeomFromText function, which constructs a geometry object at a higher speed because no unnecessary checks are required.
- If you do not specify an SRID, the default SRID 0 is used.
- If the WKT string that you specify does not represent a MultiPoint object, this function returns null.

Examples

```
SELECT ST_AsText(ST_MPointFromText('MULTIPOINT(1 2,3 4)'));
      st_astext
---------------------
 MULTIPOINT(1 2,3 4)
(1 row)
```

4.3.32. ST_MLineFromText

This topic describes the ST_MLineFromText function. This function constructs a MultiLine object by using a Well-Known Text (WKT) string and a spatial reference identifier (SRID).

Syntax

```
geometry  ST_MLineFromText(text  wKT , integer  srid);
geometry  ST_MLineFromText(text  wKT);
```

Parameters

Parameter	Description
wKT	The WKT string that you want to specify.
srid	The SRID of the MultiLine object.

Description

- If you can make sure that the information that you specify can be used to construct a MultiLine object, we recommend that you use the ST_GeomFromText function, which constructs a geometry object at a higher speed because no unnecessary checks are required.
- If you do not specify an SRID, the default SRID 0 is used.
- If the WKT string that you specify does not represent a MultiLine object, this function returns null.

Examples

```
SELECT ST_AsText(ST_MLineFromText('MULTILINESTRING((1 2,3 4), (5 6,7 8))'));
              st_astext
-------------------------------------
 MULTILINESTRING((1 2,3 4),(5 6,7 8))
(1 row)
```

4.3.33. ST_MPolyFromText

This topic describes the ST_MPolyFromText function. This function constructs a MultiPolygon object by using a Well-Known Text (WKT) string and a spatial reference identifier (SRID).

Syntax

```
geometry  ST_MPolyFromText(text  wKT , integer  srid);
geometry  ST_MPolyFromText(text  wKT);
```

Parameters

Parameter	Description
wKT	The WKT string that you want to specify.
srid	The SRID of the MultiPolygon object.

Description

- If you can make sure that the information that you specify can be used to construct a MultiPolygon

object, we recommend that you use the ST_GeomFromText function, which constructs a geometry object at a higher speed because no unnecessary checks are required.

- If you do not specify an SRID, the default SRID 0 is used.
- If the WKT string that you specify does not represent a MultiPolygon object, this function returns NULL.

Examples

```
SELECT ST_AsText(ST_MPolyFromText('MULTIPOLYGON(((0 0,1 0,1 1,0 0),(1 1,2 1,2 2,1 1)))'));
                      st_astext
----------------------------------------------------
 MULTIPOLYGON(((0 0,1 0,1 1,0 0),(1 1,2 1,2 2,1 1)))
(1 row)
```

4.3.34. ST_Point

This topic describes the ST_Point function. This function construct a point object by using coordinates.

Syntax

```
geometry  ST_Point(float  xLon , float  yLat);
```

Parameters

Parameter	Description
xLon	The x coordinate, which specifies the longitude of the point object.
yLat	The y coordinate, which specifies the latitude of the point object.

Description

This function contains only the x and y coordinates. This function is equivalent to the ST_MakePoint function that is defined in the Open Geospatial Consortium (OGC) specification.

Examples

```
SELECT ST_AsText(ST_Point(116,40));
   st_astext
---------------
 POINT(116 40)
(1 row)
```

4.3.35. ST_PointFromGeoHash

This topic describes the ST_PointFromGeoHash function. This function constructs a point object by using a Geohash string.

Syntax

```
point  ST_PointFromGeoHash(text  geohash , integer  precision);
```

Parameters

Parameter	Description
geohash	The Geohash string that you want to specify.
precision	The precision based on which you want to construct a point object. If you do not specify a precision, this function returns a point object by using the Geohash string in the full precision.

Description

The point object that is constructed by this function is the center point of the region that is represented by the Geohash string.

Examples

- Construct a point object by using the default parameter settings.

```
SELECT ST_AsText(ST_PointFromGeoHash('wx47x9u8gumnhzp791zb'));
          st_astext
----------------------------
 POINT(116 39.9999999999999)
(1 row)
```

- Construct a point object by using a custom precision.

```
SELECT ST_AsText(ST_PointFromGeoHash('wx47x9u8gumnhzp791zb',2));
       st_astext
------------------------
 POINT(118.125 42.1875)
(1 row)
```

4.3.36. ST_PointFromText

This topic describes the ST_PointFromText function. This function constructs a point object by using a Well-Known Text (WKT) string and a spatial reference identifier (SRID).

Syntax

```
geometry  ST_PointFromText(text  wKT);
geometry  ST_PointFromText(text  wKT , integer  srid);
```

Parameters

Parameter	Description
wKT	The WKT string that you want to specify.
srid	The SRID of the point object.

Description

- If you do not specify an SRID, the default SRID 0 is used.
- If the WKT string that you specify does not represent a point, this function returns NULL.
- If the WKT string that you specify is invalid, this function returns an error.
- If you can make sure that the information that you specify can be used to construct a point object, we recommend that you use the ST_GeomFromText function, which constructs a geometry object at a higher speed because no unnecessary checks are required.
- If you want to construct a point object by using latitude and longitude coordinates and you are concerned more about performance and accuracy than about compatibility with the Open Geospatial Consortium (OGC) specification, we recommend that you use the ST_MakePoint function or the ST_Point function.

Examples

Construct a point object by using a specified SRID.

```
SELECT ST_AsEWKT(ST_PointFromText('POINT(119 40)', 4326));
        st_asewkt
-------------------------
 SRID=4326;POINT(119 40)
(1 row)
```

4.3.37. ST_PointFromWKB

This topic describes the ST_PointFromWKB function. This function constructs a point object by using a Well-Known Binary (WKB) string.

Syntax

```
geometry  ST_GeomFromWKB(bytea  geom);
geometry  ST_GeomFromWKB(bytea  geom , integer  srid);
```

Parameters

Parameter	Description
geom	The WKB string that you want to specify.
srid	The spatial reference identifier (SRID) of the point object.

Description

- If you do not specify an SRID, the SRID defaults to 0.
- If the WKB string that you specify does not represent a point object, this function returns NULL.
- This function supports 3D objects and does not discard the z-index of the geometry object that is constructed.
- This function supports circular strings and curves.

Examples

```
SELECT ST_AsEWKT(ST_GeomFromWKB(E'\\x0101000000000000000c05d400000000000004440'));
   st_asewkt
---------------
 POINT(119 40)
(1 row)
```

4.3.38. ST_Polygon

This topic describes the ST_Polygon function. This function constructs a polygon object by using a LineString object and a spatial reference identifier (SRID).

Syntax

```
geometry  ST_Polygon(geometry  aLineString , integer  srid);
```

Parameters

Parameter	Description
aLineString	The LineString object that you want to specify.
srid	The spatial reference identifier (SRID) of the polygon object.

Description

- This function does not support MultiLineString objects. If you want to construct a polygon object by using a MultiLineString object, you can use the ST_LineMerge function to merge the LineString objects that comprise the MultiLineString object.
- This function cannot construct a polygon object that contains holes. If you want to construct a polygon object that contains holes, use the ST_MakePolygon function.
- This function supports 3D objects and does not discard the z-index of the geometry object that is constructed.

Examples

```
SELECT ST_AsEWKT(ST_Polygon(ST_GeomFromText('LINESTRING(1 2,3 4,5 6,1 2)'),4326));
              st_asewkt
--------------------------------------
 SRID=4326;POLYGON((1 2,3 4,5 6,1 2))
(1 row)
```

4.3.39. ST_PolygonFromText

This topic describes the ST_PolygonFromText function. This function constructs a polygon object by using a Well-Known Text (WKT) string and a spatial reference identifier (SRID).

Syntax

```
geometry  ST_PolygonFromText(text  wKT);
geometry  ST_PolygonFromText(text  wKT , integer  srid);
```

Parameters

Parameter	Description
wKT	The WKT string that you want to specify.
srid	The spatial reference identifier (SRID) of the polygon object.

Description

- If you do not specify an SRID, the SRID defaults to 0.
- If the WKT string that you specify does not represent a polygon object, this function returns NULL.
- If you are sure that the WKT string that you want to specify represents a polygon object, we recommend that you use the ST_GeomFromText function, which can construct polygon objects at higher speeds because no unnecessary checks are required.

Examples

```
SELECT ST_AsEWKT(ST_PolygonFromText('POLYGON((1 2,3 4,5 6,1 2))',4326));
              st_asewkt
--------------------------------------
 SRID=4326;POLYGON((1 2,3 4,5 6,1 2))
(1 row)
```

4.3.40. ST_WKBToSQL

This topic describes the ST_WKBToSQL function. This function constructs a geometry object by using a Well-Known Binary (WKB) string.

Syntax

```
geometry  ST_WKBToSQL(bytea  WKB);
```

Parameters

Parameter	Description
WKB	The WKB string that you want to specify.

Description

This function is equivalent to the ST_GeomFromWKB function, except that this function does not include the spatial reference identifier (SRID) parameter.

Examples

```
SELECT ST_AsText(ST_WKBToSQL(E'\\x0101000000000000000000005d400000000000004440'));
   st_astext
---------------
 POINT(116 40)
(1 row)
```

4.3.41. ST_WKTToSQL

This topic describes the ST_WKTToSQL function. This function constructs a geometry object by using a Well-Known Text (WKT) string.

Syntax

```
geometry  ST_WKTToSQL(text  WKT);
```

Parameters

Parameter	Description
WKT	The WKT string that you want to specify.

Description

- This function is equivalent to the ST_GeomFromText function.
- This function supports circular strings and curves.

Examples

```
SELECT ST_WKTToSQL('POINT(116 40)');
              st_geomfromtext
--------------------------------------------
 0101000000000000000000005D400000000000004440
(1 row)
```

4.4. Access functions

4.4.1. GeometryType

This topic describes the GeometryType function. This function returns the type of the input geometry object, such as *LineString*, *polygon*, or *MultiPoint*.

Syntax

```
text  GeometryType(geometry  geomA);
```

Parameters

Parameter	Description
geomA	The geometry object that you want to specify.

Description

- If the input geometry object contains m coordinates, the function returns a string in the PointM format.
- This function supports circular strings, curves, polyhedral surfaces, triangles, triangulated irregular network (TIN) surfaces, and 3D objects.

Examples

- Results returned by using the default parameter settings:

```
SELECT GeometryType('LINESTRING(1 0, 2 0)'::geometry);
 geometrytype
--------------
 LINESTRING
(1 row)
```

- Results returned by specifying a geometry object that contains m coordinates:

```
SELECT GeometryType('LINESTRINGM(1 0 1, 2 0 2)'::geometry);
 geometrytype
--------------
 LINESTRINGM
(1 row)
```

4.4.2. ST_Boundary

This topic describes the ST_Boundary function. This function returns the boundary of the input geometry object.

Syntax

```
geometry  ST_Boundary(geometry  geomA);
```

Parameters

Parameter	Description
geomA	The geometry object that you want to specify.

Description

This function supports 3D objects and does not delete z coordinates.

Examples

- If you specify a point object, the function returns an empty GeometryCollection object.

```
SELECT ST_AsText(ST_Boundary('POINT(1 0)'::geometry));
        st_astext
--------------------------
 GEOMETRYCOLLECTION EMPTY
(1 row)
```

- If you specify a LineString object, the function returns a MultiPoint object.

```
SELECT ST_AsText(ST_Boundary('LINESTRINGM(1 0 1, 2 0 2)'::geometry));
      st_astext
---------------------
 MULTIPOINT(1 0,2 0)
(1 row)
```

- If you specify a polygon object, the function returns a LineString object.

```
SELECT ST_AsText(ST_Boundary('POLYGON((1 0, 2 0,0 2,1 0))'::geometry));
          st_astext
----------------------------
 LINESTRING(1 0,2 0,0 2,1 0)
(1 row)
```

- If you specify a polygon object containing holes, the function returns a MultiLineString object.

```
SELECT ST_AsText(ST_Boundary('POLYGON((1 0,3 0,0 3,1 0),(1 0 ,2 0, 0 2 ,1 0))'::geometry)
);
                       st_astext
-----------------------------------------------------
 MULTILINESTRING((1 0,3 0,0 3,1 0),(1 0,2 0,0 2,1 0))
(1 row)
```

4.4.3. ST_CoordDim

This topic describes the ST_CoordDim function. This function returns the coordinate dimension of a geometry object.

Syntax

```
integer  ST_CoordDim(geometry  geomA);
```

Parameters

Parameter	Description
geomA	The geometry object that you want to specify.

Description

- The ST_CoordDim function is the alias of the ST_NDims function.
- This function supports circular strings, curves, polyhedral surfaces, triangles, triangulated irregular network (TIN) surfaces, and 3D objects.

Examples

- Execute the following statement to return a 2D object:

```
SELECT ST_CoordDim('POINT(1 0)');
 st_coorddim
-------------
           2
(1 row)
```

- Execute the following statement to return a 3DM or 3DZ object:

```
SELECT ST_CoordDim('POINT(1 0 1)');
 st_coorddim
-------------
           3
(1 row)
SELECT ST_CoordDim('POINTM(1 0 1)');
 st_coorddim
-------------
           3
(1 row)
```

- Execute the following statement to return a 4D object:

```
SELECT ST_CoordDim('POINT(1 0 1 1)');
 st_coorddim
-------------
           4
(1 row)
```

4.4.4. ST_Dimension

This topic describes the ST_Dimension function. This function returns the inherent dimension of a geometry object.

Syntax

```
integer  ST_Dimension(geometry  g);
```

Parameters

Parameter	Description
g	The geometry object that you want to specify.

Description

- This function returns different inherent dimensions when you specify different types of geometry objects.
  - If you specify a point object, this function returns 0.
  - If you specify a LineString object, this function returns 1.
  - If you specify a polygon object, this function returns 2.
- If you specify a GeometryCollection object, this function returns the largest dimension of components of the object.
- If you specify an object whose inherent dimension is unknown, such as an empty GeometryCollection object, this function returns 0.
- This function supports polyhedral surfaces, triangles, triangulated irregular network (TIN) surfaces, and 3D objects.

Examples

```
SELECT ST_Dimension('POINT(1 0)');
 st_dimension
--------------
            0
(1 row)
```

- Results returned by specifying a GeometryCollection object:

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

4.4.5. ST_EndPoint

This topic describes the ST_EndPoint function. This function returns the last point of a LineString object.

Syntax

```
geometry  ST_EndPoint(geometry  g);
```

Parameters

Parameter	Description
g	The geometry object that you want to specify.

Description

- If you do not specify a LineString object, this function returns NULL.
- This function supports 3D objects and does not delete z coordinates.
- This function supports circular strings and curves.

Examples

```
SELECT ST_AsText(ST_EndPoint('LINESTRING(0 0,2 2)'::geometry ));
 st_astext
------------
 POINT(2 2)
(1 row)
```

4.4.6. ST_Envelope

This topic describes the ST_Envelope function. This function returns the bounding box of the input geometry object.

Syntax

```
geometry  ST_Envelope(geometry  g1);
```

Parameters

Parameter	Description
g1	The geometry object that you want to specify.

Description

- The value of the bounding box is in the FLOAT8 data type.
- A bounding box is represented as a polygon object and the bounding box is defined by the corner points of the polygon object. Format of corner points: (MINX,MINY),(MINX,MAXY),(MAXX,MAXY),(MAXX,MINY),(MINX,MINY).
- If you specify a vertical line or a point, this function returns a geometry object of a lower dimension than a polygon object, such as a point object or a LineString object. This is a degenerate case.

Examples

- Results returned by using the default parameter settings:

```
SELECT ST_AsText(ST_Envelope('LINESTRING(0 0,2 2)'::geometry ));
           st_astext
-------------------------------
 POLYGON((0 0,0 2,2 2,2 0,0 0))
(1 row)
```

- Results returned in a degenerate case:

```
SELECT ST_AsText(ST_Envelope('LINESTRING(0 0,0 2)'::geometry ));
      st_astext
--------------------
 LINESTRING(0 0,0 2)
(1 row)
```

4.4.7. ST_BoundingDiagonal

This topic describes the ST_BoundingDiagonal function. This function returns a LineString object that represents the diagonal of the bounding box of the input geometry object.

Syntax

```
geometry  ST_BoundingDiagonal(geometry  geom , boolean  fits);
```

Parameters

Parameter	Description
geom	The geometry object that you want to specify.
fits	Specifies whether the best fit is required. Default value: false.

Description

- If the input geometry object is empty, the diagonal is also empty. If the input geometry object is not empty, this function returns a two-point LineString object. The minimum value of each dimension at the start point of the LineString object, and the maximum value of each dimension at the end point of the LineString object.
- The returned LineString object retains the spatial reference identifier (SRID) and the dimension of the input geometry object.
- If the value of the fits parameter is false, this function can process the diagonal of a larger bounding box. In this case, this function returns the diagonal of the bounding box of a geometry object that contains many vertices in a short period of time.

  In summary, the bounding box that corresponds to the returned diagonal always covers the input geometry object.
- This function supports 3D objects and does not delete z coordinates.
- This function supports m coordinates.

Examples

```
SELECT ST_AsText(ST_BoundingDiagonal('LINESTRING(2 2,0 0)'::geometry ));
      st_astext
---------------------
 LINESTRING(0 0,2 2)
(1 row)
```

4.4.8. ST_ExteriorRing

This topic describes the ST_ExteriorRing function. This function returns a LineString object that represents the exterior ring of the input polygon object.

Syntax

```
geometry  ST_ExteriorRing(geometry  aPolygon);
```

Parameters

Parameter	Description
aPolygon	The polygon object that you want to specify.

Description

- If you do not specify a polygon object, this function returns NULL.
- This function supports 3D objects and does not delete z coordinates.

Examples

```
SELECT ST_AsText(ST_ExteriorRing('POLYGON((1 0,3 0,0 3,1 0),(1 0,2 0, 0 2,1 0))'::geometry
));
          st_astext
----------------------------
 LINESTRING(1 0,3 0,0 3,1 0)
(1 row)
```

4.4.9. ST_GeometryN

This topic describes the ST_GeometryN function. This function returns the Nth object of the input geometry object.

Syntax

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

Parameters

Parameter	Description
geomA	The geometry object that you want to specify.
n	The index number of the object that you want to obtain.

Description

- The index number starts from 1.
- This function supports GeometryCollection objects, MultiPoint objects, MultiLineString objects, MultiCurve objects, MultiPolygon objects, and polyhedral surfaces. If you specify another type of geometry object, this function returns NULL.
- If you want to extract all the geometries of the input geometry object, you can use the ST_Dump function, which is more efficient and supportsvarious geometry objects.
- This function supports circular strings, curves, polyhedral surfaces, triangles, triangulated irregular network (TIN) surfaces, and 3D objects.

Examples

- Execute the following statement to return a MULTI object:

```
SELECT ST_AsText(ST_GeometryN('MULTIPOLYGON(((1 0,3 0,0 3,1 0)),((1 0,2 0, 0 2,1 0)))'::g
eometry,2));
         st_astext
----------------------------
 POLYGON((1 0,2 0,0 2,1 0))
(1 row)
```

- Execute the following statement to return a GeometryCollection objects:

```
SELECT ST_AsText(ST_GeometryN('GeometryCollection(POINT(1 0),POLYGON((1 0,2 0, 0 2,1 0)))
'::geometry,1));
 st_astext
------------
 POINT(1 0)
(1 row)
```

4.4.10. ST_GeometryType

This topic describes the ST_GeometryType function. This function returns the type of the input geometry object.

Syntax

```
text  ST_GeometryType(geometry  g1);
```

Parameters

Parameter	Description
g1	The geometry object that you want to specify.

Description

- This function returns the type of the input geometry object as a string, such as ST_Linestring, ST_Polygon, or ST_MultiPolygon.
- This function is different from the GeometryType function. The ST_GeometryType function does not indicate whether the input geometry object contains m coordinates.
- This function supports 3D objects and does not delete z coordinates.
- This function supports polyhedral surfaces.

Examples

- Results returned by using the default parameter settings:

```
SELECT ST_GeometryType('POINT(1 0)'::geometry);
 st_geometrytype
-----------------
 ST_Point
(1 row)
```

- Results returned by specifying a geometry object that contains m coordinates:

```
SELECT ST_GeometryType('POINTM(1 0 1)'::geometry);
 st_geometrytype
-----------------
 ST_Point
(1 row)
```

4.4.11. ST_InteriorRingN

This topic describes the ST_InteriorRingN function. This function returns the Nth interior ring of the input polygon object as a LineString object. The interior rings are identified by the specified indexes. The indexes start from 1.

Syntax

```
geometry  ST_InteriorRingN(geometry  aPolygon , integer  n);
```

Parameters

Parameter	Description
aPolygon	The polygon object that you want to specify.
n	The index number of the interior ring that you want to obtain.

Description

- If the input geometry object is not a polygon object or the value of the n parameter is invalid, this function returns NULL.
- This function does not support MultiPolygon objects. You need to use the ST_Dump function to convert a MultiPolygon object into a polygon object.
- This function supports 3D objects and does not delete z coordinates.

Examples

```
SELECT ST_AsText(ST_InteriorRingN('POLYGON((1 0,3 0,0 3,1 0),(1 0,2 0, 0 2,1 0))'::geometry
, 1));
          st_astext
----------------------------
 LINESTRING(1 0,2 0,0 2,1 0)
(1 row)
```

4.4.12. ST_IsPolygonCCW

This topic describes the ST_IsPolygonCCW function. This function returns true if the exterior rings of the input polygon object are orientated in the anticlockwise direction and the interior rings of the input polygon object are orientated in the clockwise direction.

Syntax

```
boolean  ST_IsPolygonCCW (geometry  geom);
```

Parameters

Parameter	Description
geom	The geometry object that you want to specify.

Description

- If you do not specify a polygon object, this function returns true.

  A closed LineString object is not considered a polygon object.
- If one or more interior rings and the exterior rings of the input polygon object are orientated in the same direction, both the ST_IsPolygonCW function and the ST_IsPolygonCCW function return false.
- This function supports 3D objects and does not delete z coordinates.
- This function supports m coordinates.

Examples

Execute the following statements to check whether the input polygon object is orientated in the anticlockwise direction:

```
SELECT ST_IsPolygonCCW('POLYGON((1 0,3 0,0 3,1 0),(1 0,0 2,2 0,1 0))'::geometry);
 st_ispolygonccw
-----------------
 t
(1 row)
```

4.4.13. ST_IsPolygonCW

This topic describes the ST_IsPolygonCW function. This function returns true if the exterior rings of the input polygon object are orientated in the clockwise direction and the interior rings of the input polygon object are orientated in the anticlockwise direction.

Syntax

```
boolean  ST_IsPolygonCW (geometry  geom);
```

Parameters

Parameter	Description
geom	The geometry object that you want to specify.

Description

- If you do not specify a polygon object, this function returns true.

  A closed LineString object is not considered a polygon object.
- If one or more interior rings and the exterior rings of the input polygon object are orientated in the same direction, both the ST_IsPolygonCW function and the ST_IsPolygonCCW function return false.
- This function supports 3D objects and does not delete z coordinates.
- This function supports m coordinates.

Examples

Execute the following statements to check whether the input polygon object is orientated in the clockwise direction:

```
SELECT ST_IsPolygonCW('POLYGON((1 0,0 3,3 0,1 0),(1 0,2 0,0 2,1 0))'::geometry);
 st_ispolygoncw
----------------
 t
(1 row)
```

4.4.14. ST_IsClosed

This topic describes the ST_IsClosed function. This function returns true if the input geometry object is closed.

Syntax

```
boolean  ST_IsClosed(geometry  g);
```

Parameters

Parameter	Description

Parameter	Description
g	The geometry object that you want to specify.

Description

- If you specify a polyhedral surfaces, the result of this function indicates whether the object is an open plane or a closed geometry.
- This function supports 3D objects and does not delete z coordinates.
- This function supports polyhedral surfaces.
- This function supports circular strings and curves.

Examples

- Results returned by specifying a nonclosed geometry object:

```
SELECT ST_IsClosed('LINESTRING(0 0,0 1)'::geometry);
 st_isclosed
-------------
 f
(1 row)
```

- Results returned by specifying a closed geometry object:

```
SELECT ST_IsClosed('LINESTRING(0 0,0 1,1 0,0 0)'::geometry);
 st_isclosed
-------------
 t
(1 row)
```

4.4.15. ST_IsCollection

This topic describes the ST_IsCollection function. This function returns true if the type of the input geometry object is GeometryCollection, MultiPoint, Polygon, MultiLineString, MultiCurve, MultiSurface, or MultiCompoundCurve.

Syntax

```
boolean  ST_IsCollection(geometry  g);
```

Parameters

Parameter	Description
g	The geometry object that you want to specify.

Description

- If you specify an empty GeometryCollection object, this function returns true.

- This function supports 3D objects and does not delete z coordinates.
- This function supports circular strings and curves.

Examples

- Results returned by specifying a MULTI object:

```
SELECT ST_IsCollection('MULTIPOINT((1 0),(2 0))'::geometry);
 st_iscollection
-----------------
 t
(1 row)
```

- Results returned by specifying a GeometryCollection object:

```
SELECT ST_IsCollection('GeometryCollection(POINT(1 0),POLYGON((1 0,2 0, 0 2,1 0)))'::geom
etry);
 st_iscollection
-----------------
 t
(1 row)
```

4.4.16. ST_IsEmpty

This topic describes the ST_IsEmpty function. This function returns true if the input geometry object is an empty geometry object, such as a GeometryCollection object, polygon object, or point object.

Syntax

```
boolean  ST_IsEmpty(geometry  geomA);
```

Parameters

Parameter	Description
geomA	The geometry object that you want to specify.

Description

This function supports circular strings and curves.

Examples

- Results returned by specifying an empty GeometryCollection object:

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

- Results returned by specifying an empty polygon object:

```
SELECT ST_IsEmpty('POLYGON EMPTY'::geometry);
 st_isempty
------------
 t
(1 row)
```

4.4.17. ST_IsRing

This topic describes the ST_IsRing function. This function returns true if the results of both the ST_IsClosed function and the ST_IsSimple function for the input geometry object are true.

Syntax

```
boolean  ST_IsRing(geometry  g);
```

Parameters

Parameter	Description
g	The geometry object that you want to specify.

Examples

- If you execute the following statement, the function returns true:

```
SELECT ST_IsRing(geom), ST_IsClosed(geom), ST_IsSimple(geom) FROM (SELECT 'LINESTRING(0 0
,0 2,2 0,0 0)'::geometry AS geom) AS test;
 st_isring | st_isclosed | st_issimple
-----------+-------------+-------------
 t         | t           | t
(1 row)
```

- If you execute the following statement, the function returns false:

```
SELECT ST_IsRing(geom), ST_IsClosed(geom), ST_IsSimple(geom) FROM (SELECT 'LINESTRING(0 0
,0 2,0 0)'::geometry AS geom) AS test;
 st_isring | st_isclosed | st_issimple
-----------+-------------+-------------
 f         | t           | f
(1 row)
```

4.4.18. ST_IsSimple

This topic describes the ST_IsSimple function. This function returns true if the input geometry object has no self-intersection points.

Syntax

```
boolean  ST_IsSimple(geometry  geomA);
```

Parameters

Parameter	Description
geomA	The geometry object that you want to specify.

Description

This function supports 3D objects and does not delete z coordinates.

Examples

- If you execute the following statement, the function returns true:

```
SELECT ST_IsSimple('LINESTRING(0 0,0 2,2 0,0 0)'::geometry);
 st_issimple
-------------
 t
(1 row)
```

- If you execute the following statement, the function returns false:

```
SELECT ST_IsSimple('LINESTRING(0 0,0 2,0 0)'::geometry);
 st_issimple
-------------
 f
(1 row)
```

4.4.19. ST_IsValid

This topic describes the ST_IsValid function. This function checks whether a geometry object returned by the ST_Geometry function is in a valid format.

Syntax

```
boolean  ST_IsValid(geometry  g);
boolean  ST_IsValid(geometry  g , integer  flags);
```

Parameters

Parameter	Description
g	The geometry object that you want to specify.

Parameter	Description
flags	The bitfield that you want to specify. If you specify this parameter to 1, the self-intersection ring forming a hole is considered to be valid. This is the Environmental Systems Research Institute (ESRI) flag.

Description

If the input geometry object is invalid, this function provides details about the invalid object.

Examples

```
SELECT ST_IsValid('POLYGON((0 0,0 1,1 0,1 1,0 0))'::geometry);
NOTICE:  Self-intersection at or near point 0.5 0.5
 st_isvalid
------------
 f
(1 row)
```

4.4.20. ST_IsValidReason

This topic describes the ST_IsValidReason function. This function returns the text that states whether the input geometry object is in a valid format. If not, the text states the reasons.

Syntax

```
boolean  ST_IsValidReason(geometry  geomA);
boolean  ST_IsValidReason(geometry  geomA , integer  flags);
```

Parameters

Parameter	Description
geomA	The geometry object that you want to specify.
flags	The bitfield that you want to specify. If you specify this parameter to 1, the self-intersection ring forming a hole is considered to be valid. This is the Environmental Systems Research Institute (ESRI) flag.

Description

This function can be used with the ST_IsValid function to generate a detailed report of invalid geometry object and reasons.

Examples

```
SELECT ST_IsValidReason('POLYGON((0 0,0 1,1 0,1 1,0 0))'::geometry);
      st_isvalidreason
----------------------------
 Self-intersection[0.5 0.5]
(1 row)
```

4.4.21. ST_IsValidDetail

This topic describes the ST_IsValidDetail function. This function returns details about whether the input geometry object is in a valid format. If not, this function returns the reasons and the invalid locations.

Syntax

```
validDetail  ST_IsValidDetail(geometry  geom);
validDetail  ST_IsValidDetail(geometry  geom , integer  flags);
```

Parameters

Parameter	Description
geom	The geometry object you want to specify.
flags	The bitfield that you want to specify. If you specify this parameter to 1, the self-intersection ring forming a hole is considered to be valid. This is the Environmental Systems Research Institute (ESRI) flag.

Description

- This function returns a valid_detail row. If the input geometry object is in an invalid format, the row contains the valid field stating the validity of the input geometry object, the reason field stating the reason, and a field pointing out the location of the invalid object. The value of the valid field is a Boolean value, the value of the reason field is a Varchar value, and the value of the field pointing out locations is a Geometry value.
- This function is used to replace and improve the combination of the ST_IsValid function and the ST_IsValidReason function to generate a detailed report of the invalid geometry object.

Examples

```
SELECT (g).valid,(g).reason,st_astext((g).location) from (select ST_IsValidDetail('POLYGON(
(0 0,0 1,1 0,1 1,0 0))'::geometry) as g) as t;
 valid |       reason        |   st_astext
-------+---------------------+----------------
 f     | Self-intersection | POINT(0.5 0.5)
(1 row)
```

4.4.22. ST_M

This topic describes the ST_M function. This function returns the m coordinate of the input geometry object. The input geometry object must be a point object.

Syntax

```
float  ST_M(geometry  aPoint);
```

Parameters

Parameter	Description
aPoint	The point object that you want to specify.

Description

This function supports 3D objects and does not delete z coordinates.

Examples

```
SELECT ST_M('POINTM(0 1 2)'::geometry);
 st_m
------
    2
(1 row)
```

4.4.23. ST_NDims

This topic describes the ST_NDims function. This function returns the coordinate dimension of the input geometry object.

Syntax

```
integer  ST_NDims(geometry  g1);
```

Parameters

Parameter	Description
g1	The geometry object that you want to specify.

Description

- This function returns different coordinate dimensions when you specify different types of geometry objects.
  - If you specify a 2D object, this function returns the coordinate dimension in the (X,Y) format.
  - If you specify a 3D object, this function returns the coordinate dimension in the (X,Y,Z) or (X,Y,M) format.
  - If you specify a 4D object, this function returns the coordinate dimension in the (X,Y,Z,M) format.

- This function supports 3D objects and does not delete z coordinates.

Examples

- Results returned by specifying a 2D object:

```
SELECT ST_NDims('POINT(0 1)'::geometry);
 st_ndims
----------
        2
(1 row)
```

- Results returned by specifying a 3DM object and a 3DZ object:

```
SELECT ST_NDims('POINT(0 1 2)'::geometry) as _3DZ,ST_NDims('POINTM(0 1 2)'::geometry) as
_3DM;
 _3dz | _3dm
------+------
    3 |    3
(1 row)
```

- Results returned by specifying a 4D object:

```
SELECT ST_NDims('POINT(0 1 2 3)'::geometry);
 st_ndims
----------
        4
(1 row)
```

4.4.24. ST_NPoints

This topic describes the ST_NPoints function. This function returns the number of vertexes in a geometry object. This function works for all geometry objects.

Syntax

```
integer  ST_NPoints(geometry  g1);
```

Parameters

Parameter	Description
g1	The geometry object that you want to specify.

Description

This function supports circular strings, curves, polyhedral surfaces, and 3D objects.

Examples

```
SELECT ST_NPoints('POLYGON((0 0,0 1,1 1,0 0))'::geometry);
 st_npoints
------------
          4
(1 row)
```

4.4.25. ST_NRings

This topic describes the ST_NRings function. If the input geometry object is a polygon or MultiPolygon object, this function returns the number of rings of the input geometry object.

Syntax

```
integer  ST_NRings(geometry  geomA);
```

Parameters

Parameter	Description
geomA	The geometry object that you want to specify.

Description

- This function is different from the ST_NumInteriorRings function. This function also returns the number of outer rings.
- This function supports 3D objects and does not delete z coordinates.
- This function supports circular strings and curves.

Examples

- Results returned by specifying a polygon object that contains interior rings:

```
SELECT ST_NRings('POLYGON((1 0,0 3,3 0,1 0),(1 0,2 0,0 2,1 0))'::geometry);
 st_nrings
-----------
         2
(1 row)
```

- Results returned by specifying a polygon object that does not contain interior rings:

```
SELECT ST_NRings('POLYGON((1 0,0 3,3 0,1 0))'::geometry);
 st_nrings
-----------
         1
(1 row)
```

4.4.26. ST_NumGeometries

This topic describes the ST_NumGeometries function. If the input geometry object is a collection of geometries, such as a GeometryCollection or MULTI object, this function returns the number of geometry objects of the input geometry object. If the input geometry object is a single geometry object, this function returns 1. In other cases, this function returns NULL.

Syntax

```
integer  ST_NumGeometries(geometry  geom);
```

Parameters

Parameter	Description
geom	The geometry object that you want to specify.

Description

This function supports polyhedral surfaces, triangles, triangulated irregular network (TIN) surfaces, and 3D objects.

Examples

- Results returned by specifying a MULTI object:

```
SELECT ST_NumGeometries('MULTIPOLYGON(((1 0,0 3,3 0,1 0)),((1 0,2 0,0 2,1 0)))'::geometry
);
 st_numgeometries
------------------
                 2
(1 row)
```

- Results returned by specifying a GeometryCollection object:

```
SELECT ST_NumGeometries('GeometryCollection(POINT(1 0),POLYGON((1 0,2 0, 0 2,1 0)))'::geo
metry);
 st_numgeometries
------------------
                 2
(1 row)
```

4.4.27. ST_NumInteriorRings

This topic describes the ST_NumInteriorRings function. This function returns the number of interior rings of the input geometry object.

Syntax

```
integer  ST_NumInteriorRings(geometry  aPolygon);
```

Parameters

Parameter	Description
aPolygon	The geometry object that you want to specify.

Description

If the input geometry object is not a polygon object, this function returns NULL.

Examples

Comparison between the results of the ST_NumInteriorRings function and the ST_NRings function:

```
SELECT ST_NRings(geom),ST_NumInteriorRings(geom) FROM (SELECT 'POLYGON((1 0,0 3,3 0,1 0),(1
0,2 0,0 2,1 0))'::geometry as geom) as test;
 st_nrings | st_numinteriorrings
-----------+---------------------
         2 |                   1
(1 row)
```

4.4.28. ST_NumInteriorRing

This topic describes the ST_NumInteriorRing function. This function returns the number of interior rings of the input geometry object.

Syntax

```
integer  ST_NumInteriorRing(geometry  aPolygon);
```

Parameters

Parameter	Description
aPolygon	The geometry object that you want to specify.

Examples

```
SELECT ST_NumInteriorRing('POLYGON((1 0,0 3,3 0,1 0),(1 0,2 0,0 2,1 0))'::geometry);
 st_numinteriorring
--------------------
                  1
(1 row)
```

4.4.29. ST_NumPatches

This topic describes the ST_NumPatches function. This function returns the number of planes of the input geometry object.

Syntax

```
integer  ST_NumPatches(geometry  g1);
```

Parameters

Parameter	Description
g1	The geometry object that you want to specify.

Description

- If the input geometry object is not a polyhedral object, this function returns NULL.
- This function is the alias of the ST_NumGeometries function. This function implements the SQL/MM specifications.
- This function supports polyhedral surfaces.
- This function supports 3D objects and does not delete z coordinates.

Examples

```
SELECT ST_NumPatches('POLYHEDRALSURFACE( ((0 0 0, 0 0 1, 0 1 1, 0 1 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);
 st_numpatches
---------------
             5
(1 row)
```

4.4.30. ST_NumPoints

This topic describes the ST_NumPoints function. If the input geometry object is a LineString or CircularString object, this function returns the number of points of the input geometry object.

Syntax

```
integer  ST_NumPoints(geometry  g1);
```

Parameters

Parameter	Description
g1	The geometry object that you want to specify.

Description

- This function is the alias of the ST_NPoints function.
- We recommend that you use the ST_NPoints function, which can be used in multiple scenarios and

can work with many types of geometry objects.

Examples

```
SELECT ST_NumPoints('LineString(0 0,0 1,1 1)'::geometry);
 st_numpoints
--------------
            3
(1 row)
```

4.4.31. ST_PatchN

This topic describes the ST_PatchN function. If the input geometry object is a polyhedral surface or polyhedral surfacem object, this function returns the Nth plane of the input object. The index number starts from 1. In other cases, this function returns NULL.

Syntax

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

Parameters

Parameter	Description
geomA	The geometry object that you want to specify.
n	The index number.

Description

- If you want to extract all geometry objects of the input geometry object, we recommend that you use the ST_Dump function, which is more efficient.
- This function supports polyhedral surfaces.
- This function supports 3D objects and does not delete z coordinates.

Examples

```
SELECT ST_AsText(ST_PatchN('POLYHEDRALSURFACE( ((0 0 0, 0 0 1, 0 1 1, 0 1 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,1));
                  st_astext
---------------------------------------------
 POLYGON Z ((0 0 0,0 0 1,0 1 1,0 1 0,0 0 0))
(1 row)
```

4.4.32. ST_PointN

This topic describes the ST_PointN function. This function returns the Nth point of a LineString object or circular LineString object.

Syntax

```
geometry  ST_PointN(geometry  aLinestring , integer  n);
```

Parameters

Parameter	Description
aLinestring	The LineString object or circular LineString object that you want to specify.
n	The number of the index.

Description

- If you specify an object that is not a LineString object, the ST_PointN function returns NULL.
- If the index that you specify is a negative number, this function counts the points forward from the end of the LineString object. Negative values are counted from the end of the LineString object and -1 is the last point.
- If you want to obtain the Nth point of each LineString in a MultiLineStirng object, you must use the ST_PointN function with the ST_Dump function.
- The ST_PointN function supports circular strings and curves.
- The ST_PointN function supports 3D objects and does not delete z coordinates.

Examples

- Obtain the Nth point of a LineString object by using the default parameter settings.

```
SELECT ST_AsText(ST_PointN('LINESTRING(0 0,2 2)'::geometry,1));
 st_astext
------------
 POINT(0 0)
(1 row)
```

- Obtain the Nth point of a LineString object by using the backward counting order.

```
SELECT ST_AsText(ST_PointN('LINESTRING(0 0,2 2)'::geometry,-1));
 st_astext
------------
 POINT(2 2)
(1 row)
```

4.4.33. ST_Points

This topic describes the ST_Points function. This function constructs a MultiPoint object that contains all vertexes of a geometry object.

Syntax

```
geometry ST_Points(geometry geom);
```

Parameters

Parameter	Description
geom	The geometry object that you want to specify.

Description

- The ST_Points function does not remove the duplicate points in the geometry object. For example, the ST_Points function does not delete the start point and end point of a circular geometry object.

  You can use the ST_RemoveRepeatedPoints function to remove the duplicate points.
- If the geometry object has m and z coordinates, the ST_Points function preserves the m and z coordinates.
- The ST_Points function supports circular strings and curves.

Examples

Construct a MultiPoint object by using the default parameter settings.

```
SELECT ST_AsText(ST_Points('LINESTRING(0 0,2 2)'::geometry));
      st_astext
---------------------
 MULTIPOINT(0 0,2 2)
(1 row)
```

4.4.34. ST_SRID

This topic describes the ST_SRID function. This function returns the spatial reference identifier (SRID) of a geometry object. The SRID must be defined in the spatial_ref_sys table.

Syntax

```
integer ST_SRID(geometry g1);
```

Parameters

Parameter	Description
g1	The geometry object that you want to specify.

Description

The ST_SRID function supports circular strings and curves.

Examples

Obtain the SRID of a geometry object by using the default parameter settings.

```
SELECT ST_SRID('SRID=4326;POINT(0 0)'::geometry);
 st_srid
---------
    4326
(1 row)
```

4.4.35. ST_StartPoint

This topic describes the ST_StartPoint function. This function returns the start point of a LineString object or circular LineString object. If the geometry object that you specify is not a LineString object or circular LineString object, this function returns NULL.

Syntax

```
geometry  ST_StartPoint(geometry  geomA);
```

Parameters

Parameter	Description
geomA	The geometry object that you want to specify.

Description

- The ST_StartPoint function supports 3D objects and does not delete z coordinates.
- The ST_StartPoint function supports circular strings and curves.

Examples

Obtain the start point of a LineString object or circular LineString object by using the default parameter settings.

```
SELECT ST_AsText(ST_StartPoint('LINESTRING(0 0,2 2)'::geometry));
 st_astext
------------
 POINT(0 0)
(1 row)
```

4.4.36. ST_Summary

This topic describes the ST_Summary function. This function returns a text summary of a geometry object or geography object.

Syntax

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

Parameters

Parameter	Description
g	The geometry object or geography object that you want to specify.

Description

- The parameters that are enclosed in a pair of square brackets ([]) following the geometry object or geography object have the following meanings:
  - M: The returned text summary contains m coordinates.
  - Z: The returned text summary contains z coordinates.
  - B: The returned text summary contains cached bounding boxes.
  - G: The object that you specify is a geography object.
  - S: The returned text summary contains the spatial reference system in which the geometry object or geography object resides.
- This function supports circular strings, curves, polyhedral surfaces, triangles, triangulated irregular network (TIN) surfaces, and 3D objects.

Examples

Obtain a text summary by using the default parameter settings.

```
SELECT ST_Summary('MULTIPOLYGON(((1 0,0 3,3 0,1 0)),((1 0,2 0,0 2,1 0)))'::geometry);
           st_summary
--------------------------------
 MultiPolygon[B] with 2 elements+
   Polygon[] with 1 rings       +
    ring 0 has 4 points         +
   Polygon[] with 1 rings       +
    ring 0 has 4 points
(1 row)
```

4.4.37. ST_X

This topic describes the ST_X function. This function returns the x coordinate of a point object.

Syntax

```
float  ST_X(geometry  aPoint);
```

Parameters

Parameter	Description
aPoint	The point object that you want to specify.

Description

- The object that you specify must be a point object.
- If the object that you specify is unavailable or invalid, the ST_X function returns NULL.
- The ST_X function supports 3D objects and does not delete z coordinates.

Examples

```
SELECT ST_X('POINT(0 1 2 3)'::geometry);
 st_x
------
    0
(1 row)
```

4.4.38. ST_XMax

This topic describes the ST_XMax function. This function returns the maximum x coordinate on the bounding box of a Box2D object, Box3D object, or geometry object.

Syntax

```
float  ST_XMax(box3d  aGeomorBox2DorBox3D);
```

Parameters

Parameter	Description
aGeomorBox2DorBox3D	The Box2D object, Box3D object, or geometry object that you want to specify.

Description

- The parameters of the ST_XMax function are defined to support only Box3D objects. However, the function supports auto-casting, which allows the function to process Box2D objects and geometry objects. However, if Box2D objects and geometry objects are represented by strings, the function does not support auto-casting and therefore cannot process Box2D objects or geometry objects.
- The ST_XMax function supports 3D objects and does not delete z coordinates.
- The ST_XMax function supports circular strings and curves.

Examples

Obtain the maximum x coordinate on the bounding box of a LineString object by using the default parameter settings.

```
SELECT ST_XMax('LINESTRING(0 1,2 3)'::geometry);
 st_xmax
---------
       2
(1 row)
```

4.4.39. ST_XMin

This topic describes the ST_XMin function. This function returns the minimum x coordinate on the bounding box of a Box2D object, Box3D object, or geometry object.

Syntax

```
float  ST_XMin(box3d  aGeomorBox2DorBox3D);
```

Parameters

Parameter	Description
aGeomorBox2DorBox3D	The Box2D object, Box3D object, or geometry object that you want to specify.

Description

- The parameters of the ST_XMin function are defined to support only Box3D objects. However, the function supports auto-casting, which allows the function to process Box2D objects and geometry objects. However, if Box2D objects and geometry objects are represented by strings, the function does not support auto-casting and therefore cannot process Box2D objects or geometry objects.
- The ST_XMin function supports 3D objects and does not delete z coordinates.
- The ST_XMin function supports circular strings and curves.

Examples

Obtain the minimum x coordinate on the bounding box of a LineString object by using the default parameter settings.

```
SELECT ST_XMin('LINESTRING(0 1,2 3)'::geometry);
 st_xmin
---------
       0
(1 row)
```

4.4.40. ST_Y

This topic describes the ST_Y function. This function returns the y coordinate of a point object.

Syntax

```
float  ST_Y(geometry  aPoint);
```

Parameters

Parameter	Description
aPoint	The point object that you want to specify.

Description

- The object that you specify must be a point object.
- If the object that you specify is unavailable or invalid, the ST_Y function returns NULL.
- The ST_Y function supports 3D objects and does not delete z coordinates.

Examples

```
SELECT ST_Y('POINT(0 1 2 3)'::geometry);
 st_y
------
    1
(1 row)
```

4.4.41. ST_YMax

This topic describes the ST_YMax function. This function returns the maximum y coordinate on the bounding box of a Box2D object, Box3D object, or geometry object.

Syntax

```
float  ST_YMax(box3d  aGeomorBox2DorBox3D);
```

Parameters

Parameter	Description
aGeomorBox2DorBox3D	The Box2D object, Box3D object, or geometry object that you want to specify.

Description

- The parameters of the ST_YMax function are defined to support only Box3D objects. However, the function supports auto-casting, which allows the function to process Box2D objects and geometry objects. However, if Box2D objects and geometry objects are represented by strings, the function does not support auto-casting and therefore cannot process Box2D objects or geometry objects.
- The ST_YMax function supports 3D objects and does not delete z coordinates.
- The ST_YMax function supports circular strings and curves.

Examples

Obtain the maximum y coordinate on the bounding box of a LineString object by using the default parameter settings.

```
SELECT ST_YMax('LINESTRING(0 1,2 3)'::geometry);
 st_ymax
---------
       3
(1 row)
```

4.4.42. ST_YMin

This topic describes the ST_YMin function. This function returns the minimum y coordinate on the bounding box of a Box2D object, Box3D object, or geometry object.

Syntax

```
float  ST_YMin(box3d  aGeomorBox2DorBox3D);
```

Parameters

Parameter	Description
aGeomorBox2DorBox3D	The Box2D object, Box3D object, or geometry object that you want to specify.

Description

- The parameters of the ST_YMin function are defined to support only Box3D objects. However, the function supports auto-casting, which allows the function to process Box2D objects and geometry objects. However, if Box2D objects and geometry objects are represented by strings, the function does not support auto-casting and therefore cannot process Box2D objects or geometry objects.
- The ST_YMin function supports 3D objects and does not delete z coordinates.
- The ST_YMin function supports circular strings and curves.

Examples

Obtain the minimum y coordinate on the bounding box of a LineString object by using the default parameter settings.

```
SELECT ST_YMin('LINESTRING(0 1,2 3)'::geometry);
 st_ymin
---------
       1
(1 row)
```

4.4.43. ST_Z

This topic describes the ST_Z function. This function returns the z coordinate of a point object.

Syntax

```
float  ST_Z(geometry  aPoint);
```

Parameters

Parameter	Description
aPoint	The point object that you want to specify.

Description

- The object that you specify must be a point object.
- If the object that you specify is unavailable or invalid, the ST_Z function returns NULL.
- The ST_Z function supports 3D objects and does not delete z coordinates.

Examples

```
SELECT ST_Z('POINT(0 1 2 3)'::geometry);
 st_z
------
    2
(1 row)
```

4.4.44. ST_ZMax

This topic describes the ST_ZMax function. This function returns the maximum z coordinate on the bounding box of a Box2D object, Box3D object, or geometry object.

Syntax

```
float  ST_ZMax(box3d  aGeomorBox2DorBox3D);
```

Parameters

Parameter	Description
aGeomorBox2DorBox3D	The Box2D object, Box3D object, or geometry object that you want to specify.

Description

- The parameters of the ST_ZMax function are defined to support only Box3D objects. However, the function supports auto-casting, which allows the function to process Box2D objects and geometry

objects. However, if Box2D objects and geometry objects are represented by strings, the function does not support auto-casting and therefore cannot process Box2D objects or geometry objects.

- The ST_ZMax function supports 3D objects and does not delete z coordinates.
- The ST_ZMax function supports circular strings and curves.

Examples

Obtain the maximum z coordinate on the bounding box of a LineString object by using the default parameter settings.

```
SELECT ST_ZMax('LINESTRING(0 1 2,3 4 5)'::geometry);
 st_zmax
---------
       5
(1 row)
```

4.4.45. ST_Zmflag

This topic describes the ST_Zmflag function. This function returns an integer that represents the ZM flag of a geometry object.

Syntax

```
smallint  ST_Zmflag(geometry  geomA);
```

Parameters

Parameter	Description
geomA	The geometry object that you want to specify.

Description

- The meaning of the return values:
  - 0: The ZM flag of the geometry object is a 2D point object.
  - 1: The ZM flag of the geometry object is a 3DM point object.
  - 2: The ZM flag of the geometry object is a 3DZ point object.
  - 3: The ZM flag of the geometry object is a 4D point object.
- The ST_Zmflag function supports 3D objects and does not delete z coordinates.
- The ST_Zmflag function supports circular strings and curves.

Examples

The following examples show the return values for different types of geometry objects:

```
SELECT ST_Zmflag('POINT(0 1)'::geometry) as _2D,
       ST_Zmflag('POINTM(0 1 2)'::geometry) as _3DM,
             ST_Zmflag('POINT(0 1 2)'::geometry) as _3DZ,
       ST_Zmflag('POINT(0 1 2 3)'::geometry) as _4D;
 _2d | _3dm | _3dz | _4d
-----+------+------+-----
   0 |    1 |    2 |   3
(1 row)
```

4.4.46. ST_ZMin

This topic describes the ST_ZMin function. This function returns the minimum z coordinate on the bounding box of a Box2D object, Box3D object, or geometry object.

Syntax

```
float  ST_ZMin(box3d  aGeomorBox2DorBox3D);
```

Parameters

Parameter	Description
aGeomorBox2DorBox3D	The Box2D object, Box3D object, or geometry object that you want to specify.

Description

- The parameters of the ST_ZMin function are defined to support only Box3D objects. However, the function supports auto-casting, which allows the function to process Box2D objects and geometry objects. However, if Box2D objects and geometry objects are represented by strings, the function does not support auto-casting and therefore cannot process Box2D objects or geometry objects.
- The ST_ZMin function supports 3D objects and does not delete z coordinates.
- The ST_ZMin function supports circular strings and curves.

Examples

Obtain the minimum z coordinate on the bounding box of a LineString object by using the default parameter settings.

```
SELECT ST_ZMin('LINESTRING(0 1 2,3 4 5)'::geometry);
 st_zmin
---------
       2
(1 row)
```

4.5. Functions for spatial relationships and measurements

4.5.1. ST_3DClosestPoint

This topic describes the ST_3DClosestPoint function. This function returns the 3D point that is the closest from one geometry object to another geometry object. This point is also the first point on the line that represents the shortest straight distance between the geometry objects.

Syntax

```
geometry  ST_3DClosestPoint(geometry  g1 , geometry  g2);
```

Parameters

Parameter	Description
g1	The first geometry object that you want to specify.
g2	The second geometry object that you want to specify.

Description

- This function supports 3D objects and does not delete z coordinates.

  If you specify 3D objects, this function calculates the shortest 3D distance between the objects.
- This function supports polyhedral surfaces.
- If you specify 2D objects, this function returns a 2D point that does not have a z coordinate.

Examples

Find the 3D point that is the closest from one geometry object to another geometry object by using the default parameter settings.

```
SELECT ST_AsText(ST_3DClosestPoint('LINESTRING(0 0 0,1 1 1)'::geometry,'LINESTRING(0 0 1,1
1 0)'::geometry));
       st_astext
-----------------------
 POINT Z (0.5 0.5 0.5)
(1 row)
```

4.5.2. ST_3DDistance

This topic describes the ST_3DDistance function. This function calculates the Euclidean distance between two 3D geometry objects in a projected coordinate system.

Syntax

```
float  ST_3DDistance(geometry  g1 , geometry  g2);
```

Parameters

Parameter	Description
g1	The first geometry object that you want to specify.
g2	The second geometry object that you want to specify.

Description

- This function supports 3D objects and does not delete z coordinates.
- This function supports polyhedral surfaces.

Examples

```
SELECT ST_3DDistance('POINT(0 0 0)'::geometry,'POINT(1 1 1)'::geometry);
  st_3ddistance
------------------
 1.73205080756888
(1 row)
```

4.5.3. ST_3DDWithin

This topic describes the ST_3DDWithin function. This function checks whether the 3D distance between two geometry objects in a projected coordinate system does not exceed the value of the distanceOfSrid parameter. If the 3D distance does not exceed the value of the distanceOfSrid parameter, this function returns True. If the 3D distance exceeds the value of the distanceOfSrid parameter, this function returns False.

Syntax

```
boolean  ST_3DDWithin(geometry  g1 , geometry  g2 , double precision  distanceOfSrid);
```

Parameters

Parameter	Description
g1	The first geometry object that you want to specify.
g2	The second geometry object that you want to specify.

Parameter	Description
distanceOfSrid	The distance limit that you want to specify. The distance is measured in the unit that is specified by the spatial reference system of the geometry objects.

Description

- This function supports 3D objects and does not delete z coordinates.
- This function supports polyhedral surfaces.

Examples

```
SELECT ST_3DDWithin('POINT(0 0 0)'::geometry,'POINT(1 1 1)'::geometry,2);
 st_3ddwithin
--------------
 t
(1 row)
```

4.5.4. ST_3DDFullyWithin

This topic describes the ST_3DDFullyWithin function. This function checks whether the distance between any two points on two 3D geometry objects is less than a specified distance. If the distance is less than the specified distance, this function returns True. Otherwise, this function returns False.

Syntax

```
boolean  ST_3DDFullyWithin(geometry  g1 , geometry  g2 , double  precision distance);
```

Parameters

Parameter	Description
g1	The first geometry object that you want to specify.
g2	The second geometry object that you want to specify.
distance	The distance limit that you want to specify. The distance is measured in the unit that is specified by the spatial reference system of the geometry objects.

Description

- The two geometry objects that you specify must use the same projection method and have the same spatial reference identifier (SRID).
- This function automatically compares the bounding boxes of the geometry objects that you specify

by using all available indexes on the geometry objects.

- This function supports 3D objects and does not delete z coordinates.
- This function supports polyhedral surfaces.

Examples

The following example shows the difference between this function and the ST_3DDWithin function:

```
SELECT ST_3DDWithin(g1,g2,2), ST_3DDFullyWithin(g1,g2,2) from (SELECT 'LINESTRING(0 1 0,1 1
0)'::geometry as g1,
                                                                                    'LINESTRING(
0 0 0,0 -1 0)'::geometry as g2) as test;
 st_3ddwithin | st_3ddfullywithin
--------------+-------------------
 t            | f
(1 row)
```

4.5.5. ST_3DIntersects

This topic describes the ST_3DIntersects function. This function checks whether two geometry objects intersect in the 3D space. If the two geometry objects intersect in the 3D space, this function returns True. Otherwise, this function returns False.

Syntax

```
boolean  ST_3DIntersects(geometry  geomA , geometry  geomB);
```

Parameters

Parameter	Description
geomA	The first geometry object that you want to specify.
geomB	The second geometry object that you want to specify.

Description

- If any of the Overlaps, Touches, or Within conditions evaluates to True for the geometry objects that you specify, this function considers that the geometry objects spatially intersect. If the Disjoint condition evaluates to True for the geometry objects that you specify, this function considers that the geometry objects do not spatially intersect.
- This function automatically compares the bounding boxes of the geometry objects that you specify by using all available indexes on the geometry objects.
- This function supports 3D objects and does not delete z coordinates.
- This function supports polyhedral surfaces, triangles, and triangulated irregular network (TIN) surfaces.

Examples

```
SELECT ST_3DIntersects('LINESTRING(0 0 0,1 1 1)'::geometry,'LINESTRING(1 0 0,0 1 1)'::geometry);
 st_3dintersects
-----------------
 t
(1 row)
```

4.5.6. ST_3DLongestLine

This topic describes the ST_3DLongestLine function. This function returns the longest line that is formed by the points of two geometry objects in the 3D space.

Syntax

```
geometry  ST_3DLongestLine(geometry  g1 , geometry  g2);
```

Parameters

Parameter	Description
g1	The first geometry object that you want to specify.
g2	The second geometry object that you want to specify.

Description

- If this function finds more than one longest line between the geometry objects you specify, this function returns only the first longest line that is found.
- The line that is returned by this function always starts from the first geometry object that you specify and points to the second geometry object that you specify.
- When you specify the same first and second geometry objects for this function and the ST_3DMaxDistance function, these functions always return the same result.
- This function supports 3D objects and does not delete z coordinates.
- This function supports polyhedral surfaces.

Examples

```
SELECT ST_AsText(ST_3DLongestLine('POINT(0 0 0)'::geometry,'LINESTRING(0 0 1,1 1 0)'::geometry));
         st_astext
----------------------------
 LINESTRING Z (0 0 0,1 1 0)
(1 row)
```

4.5.7. ST_3DMaxDistance

This topic describes the ST_3DMaxDistance function. This function calculates the farthest distance between two geometry objects in the 3D space.

Syntax

```
float  ST_3DMaxDistance(geometry  g1 , geometry  g2);
```

Parameters

Parameter	Description
g1	The first geometry object that you want to specify.
g2	The second geometry object that you want to specify.

Description

- This function returns the maximum Euclidean distance between the geometry objects that you specify. The distance is measured in the unit that is specified by the projected coordinate system of the geometry objects.
- This function supports polyhedral surfaces.
- This function supports 3D objects and does not delete z coordinates.

Examples

```
SELECT ST_3DMaxDistance('POINT(0 0 0)'::geometry,'LINESTRING(0 0 1,1 1 0)'::geometry);
 st_3dmaxdistance
------------------
  1.4142135623731
(1 row)
```

4.5.8. ST_3DShortestLine

This topic describes the ST_3DShortestLine function. This function returns the shortest line that can be formed by the points of two geometry objects in the 3D space.

Syntax

```
geometry  ST_3DShortestLine(geometry  g1 , geometry  g2);
```

Parameters

Parameter	Description
g1	The first geometry object that you want to specify.

Parameter	Description
g2	The second geometry object that you want to specify.

Description

- If this function finds more than one shortest line between the geometry objects you specify, this function returns only the first shortest line that is found.
- If the geometry objects that you specify intersect at only one point, this function returns a line whose start point and end point are both the intersection point.
- If the geometry objects that you specify intersect at more than one point, this function returns a line whose start point and end point are both one of the intersection points.
- The line that is returned by this function always starts from the first geometry object that you specify and points to the second geometry object that you specify.
- When you specify the same first and second geometry objects for this function and the ST_3DMaxDistance function, these functions always return the same result.
- This function supports 3D objects and does not delete z coordinates.
- This function supports polyhedral surfaces.

Examples

The following example shows the difference between the ST_3DLongestLine function and the ST_3DShortestLine function.

```
SELECT ST_AsText(ST_3DLongestLine(g1,g2)) as lonest,
             ST_AsText(ST_3DShortestLine(g1,g2)) as shortest
       from (SELECT 'POINT(0 0 0)'::geometry as g1,
             'LINESTRING(0 0 1,1 1 0)'::geometry as g2) as test;
          lonest          |              shortest
--------------------------+-----------------------------------
 LINESTRING Z (0 0 0,1 1 .| LINESTRING Z (0 0 0,0.333333333333.
.0)                       |.333 0.333333333333333 0.6666666666.
                          |.66667)
(1 row)
```

4.5.9. ST_Area

This topic describes the ST_Area function. This function calculates the area of a polygon object.

Syntax

```
float  ST_Area(geometry  g1);
float  ST_Area(geography  geog , boolean  useSpheroid);
```

Parameters

Parameter	Description
g1	The geometry object that you want to specify.
geog	The geography object that you want to specify.
useSpheroid	Specifies whether to use an ellipsoid. Default value: True.

Description

- If you specify a geometry object, this function returns the 2D Cartesian area of the geometry object in the unit that is specified by the spatial reference system of the geometry object.
- If you specify a geography object, this function returns the area of the geometry object on the ellipsoid that you define. The area is measured in square meters.

  You can choose not to use an ellipsoid. This way, this function can calculate the area at a higher speed but with a lower accuracy.
- This function supports polyhedral surfaces.

  This function supports 2D polyhedral surfaces. This function does not support 2.5D polyhedral surfaces. If you specify a 2.5D polyhedral surface that completely resides on a plane with only x and y coordinates, this function may return a non-zero result.

Examples

```
SELECT ST_Area(ST_Buffer(ST_GeomFromText('POINT(0 0)',3857),1000));
     st_area
------------------
 3121445.15225805
(1 row)
```

4.5.10. ST_Azimuth

This topic describes the ST_Azimuth function. This function calculates the azimuth from one point to another. The azimuth is an angle that rotates in the clockwise direction starting from the north.

Syntax

```
float  ST_Azimuth(geometry  pointA , geometry  pointB);
float  ST_Azimuth(geography  pointA , geography  pointB);
```

Parameters

Parameter	Description
pointA	The first point that you want to specify.
pointB	The second point that you want to specify.

Description

- If the points that you specify are coincident, this function returns NULL.
- The azimuth is an angle that is counted starting from the north and is positive in the clockwise direction. The azimuth is 0 on the due north , π/2 on the due east, π on the due south, and 3π/2 on the due west.

Examples

Calculate the azimuth from one point to another by using the default parameter settings.

```
SELECT degrees(ST_Azimuth('POINT(0 0)'::geometry,'POINT(0 -1)'::geometry));
 degrees
---------
     180
(1 row)
```

4.5.11. ST_Angle

This topic describes the ST_Angle function. This function calculates the angle between three points or between two vectors that are depicted by four points or by two lines.

Syntax

```
float  ST_Angle(geometry  point1 , geometry  point2 , geometry  point3 , geometry  point4);
float  ST_Angle(geometry  line1 , geometry  line2);
```

Parameters

Parameter	Description
point1	The first point that you want to specify.
point2	The second point that you want to specify.
point3	The third point that you want to specify.
point4	The fourth point that you want to specify.
line1	The first line that you want to specify.
line2	The second line that you want to specify.

Description

- If you specify three points, which are named P1, P2, and P3 in sequence, this function calculates the angle clockwise from P1 to P2 and then to P3.

  ```
  ST_Angle(P1,P2,P3) = ST_Angle(P2,P1,P2,P3)
  ```

- If you specify four points, which are named P1, P2, P3, and P4 in sequence, this function calculates the angle clockwise from P1 to P2 and from P3 to P4.

- The angle that is returned is always positive, ranging between 0 and 2π radians. You can use the degrees() function to convert the angle into a degree.
- If you specify two lines, this function considers the closing points of the lines as four points.

Examples

Calculate the angle between two lines.

```
SELECT degrees(ST_Angle('LINESTRING(0 0,0 1)'::geometry,'LINESTRING(0 0,1 1)'::geometry));
 degrees
---------
      45
(1 row)
```

4.5.12. ST_Centroid

This topic describes the ST_Centroid function. This function calculates the center of a geometry object or geography object.

Syntax

```
geometry  ST_Centroid(geometry  g1);
geography  ST_Centroid(geography  g1 , boolean  useSpheroid);
```

Parameters

Parameter	Description
g1	The geometry object or geography object that you want to specify.
useSpheroid	Specifies whether to use an ellipsoid. This parameter is valid if you specify a geography object. Default value: True.

Examples

- Calculate the center of a MultiPoint object.

```
SELECT ST_AsText(ST_Centroid('MULTIPOINT(1 1,-1 -1)'::geometry));
 st_astext
------------
 POINT(0 0)
(1 row)
```

- Calculate the center of a LineString object.

```
SELECT ST_AsText(ST_Centroid('LINESTRING(0 0,0 1,1 2)'::geometry));
                 st_astext
-------------------------------------------
 POINT(0.292893218813453 1.08578643762691)
(1 row)
```

4.5.13. ST_ClosestPoint

This topic describes the ST_ClosestPoint function. This function compares two geometry objects and returns the 2D point that belongs to one geometry object and is the closest to the other geometry object.

Syntax

```
geometry  ST_ClosestPoint(geometry  g1 , geometry  g2);
```

Parameters

Parameter	Description
g1	The first geometry object that you want to specify.
g2	The second geometry object that you want to specify.

Description

The point that is returned is the end point of the shortest line between the geometry objects that you specify.

Examples

```
SELECT ST_AsText(ST_ClosestPoint('LINESTRING(0 -1,0 1)'::geometry,'POINT(1 0)'::geometry));
 st_astext
------------
 POINT(0 0)
(1 row)
```

4.5.14. ST_Contains

This topic describes the ST_Contains function. This function checks whether the first geometry object that you specify contains the second geometry object that you specify. If the first geometry object contains the second geometry object, this function returns True. Otherwise, this function returns False.

Syntax

```
boolean  ST_Contains(geometry  geomA , geometry  geomB);
```

Parameters

Parameter	Description
geomA	The first geometry object that you want to specify.

Parameter	Description
geomB	The second geometry object that you want to specify.

Description

- The first geometry object that you specify contains the second geometry object that you specify only when no points of the second geometry object lie outside the first geometry object and at least one point of the second geometry object lies inside the first geometry object.
  - The ST_Contains function does not count in the boundary of the first geometry object. The ST_Contains function counts in only the first geometry object itself.
  - The ST_ContainsProperly function does not count in the first geometry object itself.
- The two geometry objects that you specify must use the same projection method and have the same spatial reference identifier (SRID).
- The ST_Contains function is inverse to the ST_Within function.

  The ST_Contains(A,B) function and the ST_Within(B,A) function return the same result unless you specify invalid geometry objects in one of the functions.
- The ST_Contains function does not support GeometryCollection objects.
- The ST_Contains function generates a bounding box, which allows you to use the indexes that are available on the geometry objects that you specify. If you do not want to use the indexes, we recommend that you use the _ST_Contains function.

Examples

```
SELECT ST_Contains('POLYGON((0 0,0 3,3 0,0 0))'::geometry,'POLYGON((0 0,0 1,1 0,0 0))'::geo
metry);
 st_contains
-------------
 t
(1 row)
```

4.5.15. ST_ContainsProperly

This topic describes the ST_ContainsProperly function. This function checks whether the second geometry object that you specify completely lies in the first geometry object that you specify. If the second geometry object completely lies in the first geometry object, this function returns True. Otherwise, this function returns False.

Syntax

```
boolean  ST_ContainsProperly(geometry  geomA , geometry  geomB);
```

Parameters

Parameter	Description
geomA	The first geometry object that you want to specify.
geomB	The second geometry object that you want to specify.

Description

- If the second geometry object that you specify intersects with the interior of the first geometry object that you specify but does not touch the boundary or exterior of the first geometry object, this function returns True. All points of the second geometry object lie inside the first geometry object.
- The DE-9IM matrix patterns of the geometry objects that you specify conform to [T**FF*FF*].
- This function does not support GeometryCollection objects.
- Do not specify invalid geometry objects. If you specify invalid geometry objects, this function returns an unexpected result.
- This function generates a bounding box, which allows you to use the indexes that are available on the geometry objects that you specify. If you do not want to use the indexes, we recommend that you use the _ST_ContainsProperly function.

Examples

The following example shows the difference between the ST_ContainsProperly function and the ST_Contains function.

```
SELECT ST_Contains(g1,g2), ST_ContainsProperly(g1,g2),ST_ContainsProperly(g1,g3)
    FROM (SELECT 'POLYGON((0 0,0 3,3 3,3 0,0 0))'::geometry as g1,
                     'POLYGON((0 0,0 1,1 1,1 0,0 0))'::geometry as g2,
                     'POLYGON((1 1,1 2,2 2,2 1,1 1))'::geometry as g3 ) as test;
 st_contains | st_containsproperly | st_containsproperly
-------------+---------------------+---------------------
 t           | f                   | t
(1 row)
```

4.5.16. ST_Covers

This topic describes the ST_Covers function. This function checks whether no point of the second geometry object or geography object that you specify lies outside the first geometry object or geography object that you specify. If no point of the second object lies outside the first object, this function returns True. Otherwise, this function returns False.

Syntax

```
boolean  ST_Covers(geometry  geomA , geometry  geomB);
boolean  ST_Covers(geography  geogpolyA , geography  geogpointB);
```

Parameters

Parameter	Description
geomA	The first geometry object that you want to specify.
geomB	The second geometry object that you want to specify.
geogpolyA	The first geography object that you want to specify.
geogpointB	The second geography object that you want to specify.

Description

- This function does not support GeometryCollection objects.
- Do not specify invalid geometry objects. If you specify invalid geometry objects, this function returns an unexpected result.
- This function generates a bounding box, which allows you to use the indexes that are available on the geometry objects that you specify. If you do not want to use indexes, we recommend that you use the _ST_Covers function.

Examples

The following example shows the difference between the ST_Covers function and the ST_Contains function.

```
SELECT ST_Covers(g1,g2), ST_Contains(g1,g2) FROM (SELECT 'POLYGON((0 0,1 1,1 0,0 0))'::geom
etry as g1,
                                                                    'LINESTRING(0 0,1 1)'::geome
try as g2) as test;
 st_covers | st_contains
-----------+-------------
 t         | f
(1 row)
```

4.5.17. ST_CoveredBy

This topic describes the ST_CoveredBy function. This function checks whether no point of the first geometry object or geography object that you specify lies outside the second geometry object or geography object that you specify. If no point of the first object lies outside the second object, this function returns True. Otherwise, this function returns False.

Syntax

```
boolean  ST_CoveredBy(geometry  geomA , geometry  geomB);
boolean  ST_CoveredBy(geography  geogA , geography  geogB);
```

Parameters

Parameter	Description
geomA	The first geometry object that you want to specify.
geomB	The second geometry object that you want to specify.
geogA	The first geography object that you want to specify.
geogB	The second geography object that you want to specify.

Description

- This function does not support GeometryCollection objects.
- Do not specify invalid geometry objects. If you specify invalid geometry objects, this function returns an unexpected result.
- This function generates a bounding box, which allows you to use the indexes that are available on the geometry objects that you specify. If you do not want to use indexes, we recommend that you use the _ST_CoveredBy function.

Examples

Compare two geometry objects by using the default parameter settings.

```
SELECT ST_Covers(g1,g2), ST_CoveredBy(g2,g1) FROM (SELECT 'POLYGON((0 0,1 1,1 0,0 0))'::geo
metry as g1,
                                                        'LINESTRING(0 0,1 1)'::geometry as
g2) as test;
 st_covers | st_coveredby
-----------+--------------
 t         | t
(1 row)
```

4.5.18. ST_Crosses

This topic describes the ST_Crosses function. This function checks whether two geometry objects partially intersect but one does not lie inside the other. If the two geometry objects partially intersect but one does not lie inside the other, this function returns True. Otherwise, this function returns False.

Syntax

```
boolean  ST_Crosses(geometry  g1 , geometry  g2);
```

Parameters

Parameter	Description
g1	The first geometry object that you want to specify.

Parameter	Description
g2	The second geometry object that you want to specify.

Description

- This function returns True only when the intersection of the geometry objects that you specify is not an empty set and has a dimension that is smaller than the maximum dimension among the geometry objects.
- This function returns True only when the intersection of two geometry objects that you specify is not equal to one of the geometry objects.
- The geometry objects that you specify must use one of the following DE-9IM matrix patterns:
  - T*T******: This pattern is used when you specify a point and a line, a point and an area, or a line and an area.
  - T*****T**: This pattern is used when you specify a line and a point, an area and a point, or an area and a line.
  - 0********: This pattern is used when you specify two lines.
- This function does not support GeometryCollection objects.
- This function generates a bounding box, which allows you to use the indexes that are available on the geometry objects that you specify. If you do not want to use indexes, we recommend that you use the _ST_Crosses function.

Examples

- Compare two geometry objects by using the default parameter settings.

```
SELECT ST_Crosses('LINESTRING(0 0,3 3)'::geometry,'LINESTRING(0 1,1 0)'::geometry);
 st_crosses
------------
 t
(1 row)
```

- Compare two geometry objects that overlap.

```
 SELECT ST_Crosses('LINESTRING(0 0,3 3)'::geometry,'LINESTRING(0 0,1 1)'::geometry);
 st_crosses
------------
 f
(1 row)
```

- Compare two geometry objects whose intersection has a dimension that is the same as the maximum dimension among the geometry objects.

```
SELECT ST_Crosses('LINESTRING(0 0,3 3)'::geometry,'LINESTRING(1 1,4 4)'::geometry);
 st_crosses
------------
 f
(1 row)
```

4.5.19. ST_LineCrossingDirection

This topic describes the ST_LineCrossingDirection function. This function returns an integer that ranges from -3 to 3 to represent how two LineString objects are related to each other.

Syntax

```
integer  ST_LineCrossingDirection(geometry  linestringA , geometry  linestringB);
```

Parameters

Parameter	Description
linestringA	The first LineString object that you want to specify.
linestringB	The second LineString object that you want to specify.

Description

- This topic supports only LineString objects.
- This function can return the following values:
  - 0: The LineString objects do not intersect.
  - -1: The LineString objects intersect on their left parts.
  - 1: The LineString objects intersect on their right parts.
  - -2: The LineString objects intersect mostly on their left parts.
  - 2: The LineString objects intersect mostly on their right parts.
  - -3: The LineString objects intersect mostly on the left part of the first LineString object.
  - 3: The LineString objects intersect mostly on the right part of the first LineString object.

Examples

```
SELECT ST_LineCrossingDirection('LINESTRING(0 0,3 3)'::geometry,'LINESTRING(0 1,1 0)'::geom
etry);
 st_linecrossingdirection
--------------------------
                        1
(1 row)
```

4.5.20. ST_Disjoint

This topic describes the ST_Disjoint function. This function checks whether two geometry objects do not touch each other. If the geometry objects do not touch each other, this function returns True. Otherwise, this function returns False.

Syntax

```
boolean  ST_Disjoint(geometry  a , geometry  b);
```

Parameters

Parameter	Description
a	The first geometry object that you want to specify.
b	The second geometry object that you want to specify.

Description

- If one of the Overlap, Touch, and Within conditions evaluates to True for the geometry objects that you specify, this function returns False. Otherwise, this function returns True.
- This function does not use indexes.

Examples

Compare two geometry objects by using the default parameter settings.

```
SELECT ST_Disjoint('LINESTRING(0 0,-3 -3)'::geometry,'LINESTRING(0 1,1 0)'::geometry);
 st_disjoint
-------------
 t
(1 row)
```

4.5.21. ST_Distance

This topic describes the ST_Distance function. This function calculates the 2D Euclidean distance between two geometry objects in the reference coordinate system of the geometry objects. If you specify two geography objects, this function returns the minimum spherical distance in meters between the geography objects.

Syntax

```
float  ST_Distance(geometry  g1 , geometry  g2);
float  ST_Distance(geography  gg1 , geography  gg2);
float  ST_Distance(geography  gg1 , geography  gg2 , boolean  useSpheroid);
```

Parameters

Parameter	Description
g1	The first geometry object that you want to specify.
g2	The second geometry object that you want to specify.
gg1	The first geography object that you want to specify.
gg2	The second geography object that you want to specify.
useSpheroid	Specifies whether to use an ellipsoid. If you set this parameter to False, a specific precision is applied to improve the speed.

Description

This function supports circular strings and curves.

Examples

The following example shows the difference between the ST_Distance function and the ST_3DDistance function.

```
SELECT ST_Distance(g1,g2),ST_3DDistance(g1,g2) FROM(SELECT 'POINT(0 0 0)'::geometry as g1,
                                                                    'POINT(1 1 1)'::geometry a
s g2) as test;
   st_distance    |  st_3ddistance
-----------------+------------------
 1.4142135623731 | 1.73205080756888
(1 row)
```

4.5.22. ST_MinimumClearance

This topic describes the ST_MinimumClearance function. This function returns the minimum clearance of a geometry object.

Syntax

```
float  ST_MinimumClearance(geometry  g);
```

Parameters

Parameter	Description
g	The geometry object that you want to specify.

Description

- The minimum clearance of a geometry is the smallest distance by which a vertex of the geometry can be moved to produce an invalid geometry. The minimum clearance of a geometry can be considered as a quantitative measure of the robustness of the geometry. A larger minimum clearance indicates a higher robustness.
- If a geometry has a minimum clearance of e, the following conclusions can be made:
  - The distance between any two different vertices of the geometry is greater than e.
  - The distance between any vertex of the geometry and a line that is not an end point is greater than or equal to e.
- If a geometry does not have a minimum clearance, this function returns Infinity. For example, this function returns Infinity for a single point or for a MultiPoint whose points are identical.

Examples

Query the minimum clearance of a geometry object by using the default parameter settings.

```
SELECT ST_MinimumClearance('POLYGON((0 0, 1 0,1 1e-4, 0 0))');
 st_minimumclearance
---------------------
      9.99999995e-05
(1 row)
```

4.5.23. ST_MinimumClearanceLine

This topic describes the ST_MinimumClearanceLine function. This function returns a LineString object that consists of two points, which represent the minimum clearance of a geometry object.

Syntax

```
Geometry  ST_MinimumClearanceLine(geometry  g);
```

Parameters

Parameter	Description
g	The geometry object that you want to specify.

Description

If the geometry object that you specify does not have a minimum clearance, this function returns LineString Empty.

Examples

Query the minimum clearance of a geometry object by using the default parameter settings.

```
SELECT ST_AsText(ST_MinimumClearanceLine('POLYGON((0 0, 1 0,1 1e-4, 0 0))'));
                 st_astext
-------------------------------------------
 LINESTRING(1 0,0.99999999 0.000099999999)
(1 row)
```

4.5.24. ST_HausdorffDistance

This topic describes the ST_HausdorffDistance function. This function calculates the Hausdorff distance between two geometry objects.

Syntax

```
float  ST_HausdorffDistance(geometry  g1 , geometry  g2);
float  ST_HausdorffDistance(geometry  g1 , geometry  g2 , float  densifyFrac);
```

Parameters

Parameter	Description
g1	The first geometry object that you want to specify.
g2	The second geometry object that you want to specify.
densifyFrac	The fraction based on which this function densifies each segment. Each segment is split into a number of equal-length sub-segments, whose fraction of the total length is the closest to the specified fraction.

Description

- This function supports only vertices as discrete locations and does not limit the number of points.
- The Hausdorff distance that is returned by this function is not a standard Hausdorff distance. This function returns an approximate distance for large GeometryCollection objects.
- This function can compute the Hausdorff distance between two LineString objects, which must basically be parallel and have the same length. This function can be used to measure the similarity between LineString objects.

Examples

Calculate the Hausdorff distance between two geometry objects by using the default parameter settings.

```
SELECT st_hausdorffdistance('LINESTRING (1 0,2 0)'::geometry, 'LINESTRING (-1 0,0 0)'::geom
etry);
 st_hausdorffdistance
----------------------
                    2
(1 row)
```

4.5.25. ST_FrechetDistance

This topic describes the ST_FrechetDistance function. This function calculates the Frechet distance between two geometry objects.

Syntax

```
float  ST_FrechetDistance(geometry  g1 , geometry  g2 , float  densifyFrac);
```

Parameters

Parameter	Description
g1	The first geometry object that you want to specify.
g2	The second geometry object that you want to specify.
densifyFrac	The fraction based on which this function densifies each segment. Each segment is split into a number of equal-length sub-segments, whose fraction of the total length is the closest to the specified fraction.

Description

- The Frechet distance is a measure of the similarity between curves. When this function calculates the Frechet distance, this function considers the locations and order of the points along the curves. Therefore, the Frechet distance is more accurate than the Hausdorff distance.
- If you specify the densifyFrac parameter, this function performs a segment densification before it calculates the Frechet distance. Each segment is split into a number of equal-length sub-segments, whose fraction of the total length is the closest to the specified fraction.
- This function supports only vertices as discrete locations and does not limit the number of points.
- A smaller value of the densifyFrac parameter indicates a more accurate Frechet distance. However, the computation time and memory usage are proportional to the square of the number of sub-segments.

Examples

Calculate the Frechet distance between two geometry objects by using the default parameter settings.

```
SELECT ST_FrechetDistance('LINESTRING (1 0,2 0)'::geometry, 'LINESTRING (-1 0,0 0)'::geomet
ry);
 st_frechetdistance
--------------------
                  2
(1 row)
```

4.5.26. ST_MaxDistance

This topic describes the ST_MaxDistance function. This function calculates the farthest distance between two geometry objects. The farthest distance is measured in the unit that is specified in the spatial reference system of the geometry objects.

Syntax

```
float  ST_MaxDistance(geometry  g1 , geometry  g2);
```

Parameters

Parameter	Description
g1	The first geometry object that you want to specify.
g2	The second geometry object that you want to specify.

Description

If the first geometry object and second geometry object that you specify are the same, the function returns the distance between the farthest two vertices of the geometry object.

Examples

Calculate the farthest distance between two geometry objects by using the default parameter settings.

```
SELECT ST_MaxDistance('LINESTRING (0 0,-1 1)'::geometry, 'LINESTRING (0 0,1 1)'::geometry);
 st_maxdistance
----------------
              2
(1 row)
```

4.5.27. ST_DistanceSphere

This topic describes the ST_DistanceSphere function. This function calculates the minimum distance between two geometry objects. The minimum distance is represented by a latitude and a longitude and is measured in meters.

Syntax

```
float  ST_DistanceSphere(geometry  geomlonlatA , geometry  geomlonlatB);
```

Parameters

Parameter	Description
geomlonlatA	The first geometry object that you want to specify.

Parameter	Description
geomlonlatB	The second geometry object that you want to specify.

Description

- This function uses the information about the ellipsoid that is defined by the spatial reference identifier (SRID) in the source data.
- This function calculates the minimum distance between two geometry objects at a faster speed but a lower accuracy than the ST_DistanceSpheroid function.

Examples

```
SELECT ST_DistanceSphere(ST_GeomFromText('POINT (120 36)',4326), ST_GeomFromText('POINT (116 40)',4326));
 st_distancesphere
-------------------
   566139.31447502
(1 row)
```

4.5.28. ST_DistanceSpheroid

This topic describes the ST_DistanceSpheroid function. This function calculates the minimum distance between two geometry objects that are represented by latitudes and longitudes on an ellipsoid. The minimum distance is measured in meters.

Syntax

```
float ST_DistanceSpheroid(geometry geomlonlatA , geometry geomlonlatB , spheroid measurementSpheroid);
```

Parameters

Parameter	Description
geomlonlatA	The first geometry object that you want to specify.
geomlonlatB	The second geometry object that you want to specify.
measurementSpheroid	The ellipsoid that you want to define.

Description

This function does not check the spatial reference identifier (SRID) of a geometry object. This function always assumes that a geometry object is represented in the coordinates of the ellipsoid that you define.

Examples

Calculate the minimum distance between two geometry objects by using the default parameter settings.

```
SELECT ST_DistanceSpheroid(ST_GeomFromText('POINT (120 36)',4326), ST_GeomFromText('POINT (
116 40)',4326), 'SPHEROID["WGS 84",6378137,298.257223563]');
 st_distancespheroid
---------------------
    566034.793071762
(1 row)
```

4.5.29. ST_DFullyWithin

This topic describes the ST_DFullyWithin function. This function checks whether the distance between any two points on two geometry objects is less than a specified distance. If the distance between any two points on the two geometry objects is less than the specified distance, this function returns True. Otherwise, this function returns False.

Syntax

```
boolean  ST_DFullyWithin(geometry  g1 , geometry  g2 , double precision  distance);
```

Parameters

Parameter	Description
g1	The first geometry object that you want to specify.
g2	The second geometry object that you want to specify.
distance	The distance that you want to specify. The distance is measured in the unit that is specified by the spatial reference system of the geometry objects.

Description

- The two geometry objects that you specify must use the same projection method and have the same spatial reference identifier (SRID).
- This function automatically compares the bounding boxes of the geometry objects that you specify by using all available indexes on the geometry objects.

Examples

The following example shows the difference between the ST_DFullyWithin function and the ST_DWithin function:

```
SELECT ST_DFullyWithin(g1,g2,2), ST_DWithin(g1,g2,2) from (SELECT 'LINESTRING(0 1,1 1)'::ge
ometry as g1,
                                                                                 'LINESTRING(0 0,
0 -1)'::geometry as g2) as test;
 st_dfullywithin | st_dwithin
-----------------+------------
 f               | t
(1 row)
```

4.5.30. ST_DWithin

This topic describes the ST_DWithin function. This function checks whether two objects are within a specified distance to each other. If the two objects are within a specified distance to each other, this function returns True. Otherwise, this function returns False.

Syntax

```
boolean  ST_DWithin(geometry  g1 , geometry  g2 , double precision  distanceOfSrid);
boolean  ST_DWithin(geography  gg1 , geography  gg2 , double precision  distanceMeters);
boolean  ST_DWithin(geography  gg1 , geography  gg2 , double precision  distanceMeters , bo
olean  useSpheroid);
```

Parameters

Parameter	Description
g1	The first geometry object that you want to specify.
g2	The second geometry object that you want to specify.
distanceOfSrid	The distance that you want to specify. The distance is measured in the unit that is specified in the spatial reference system of the geometry objects.
gg1	The first geography object that you want to specify.
gg2	The second geography object that you want to specify.
distanceMeters	The distance that you want to specify. Unit: meters.
useSpheroid	Specifies whether to use an ellipsoid reference system. If you use an ellipsoid reference system, this function returns a more accurate result but at a slightly lower speed.

Description

- If you specify two geometry objects, the geometry objects must have the same spatial reference

identifier (SRID).

- By default, if you specify two geography objects, the distance is measured in meters.
- This function automatically compares the bounding boxes of the geometry objects that you specify by using all available indexes on the geometry objects.
- If you want to check whether two 3D objects are within a specified distance to each other, use the ST_3DDWithin function.

Examples

The following example shows the difference between the ST_DWithin function and the ST_DFullyWithin function:

```
SELECT ST_DFullyWithin(g1,g2,2), ST_DWithin(g1,g2,2) from (SELECT 'LINESTRING(0 1,1 1)'::ge
ometry as g1,
                                                                                 'LINESTRING(0 0,
0 -1)'::geometry as g2) as test;
 st_dfullywithin | st_dwithin
-----------------+------------
 f               | t
(1 row)
```

4.5.31. ST_Equals

This topic describes the ST_Equals function. This function checks whether two geometry objects are spatially equal. If the two geometry objects are spatially equal, this function returns True. Otherwise, this function returns False.

Syntax

```
boolean  ST_Equals(geometry  a , geometry  b);
```

Parameters

Parameter	Description
a	The first geometry object that you want to specify.
b	The second geometry object that you want to specify.

Description

- If both ST_Within(A,B) and ST_Within(B,A) return True, Geometry Object A and Geometry Object B are spatially equal. Geometry Object A and Geometry Object B may have their points arranged in different orders, but they have the same spatial structure.

  You can use the ST_OrderingEquals function to check whether Geometry Object A and Geometry Object B have their points arranged in the same order.
- If one of the geometry objects that you specify is invalid, this function returns False unless the

geometry objects are binarily equal.

- This function does not support GeometryCollection objects.

Examples

- The following example shows how two geometry objects are spatially equal even if their points arranged in different orders:

```
SELECT ST_Equals('LINESTRING(0 1,2 3)'::geometry,'LINESTRING(2 3,0 1)'::geometry);
 st_equals
-----------
 t
(1 row)
```

- The following example shows how two geometry objects are identical if they are spatially equal:

```
SELECT ST_Equals('LINESTRING(0 1,0 3)'::geometry,'LINESTRING(0 1,0 2,0 3)'::geometry);
 st_equals
-----------
 t
(1 row)
```

4.5.32. ST_GeometricMedian

This topic describes the ST_GeometricMedian function. This function returns the geometric median of a geometry object.

Syntax

```
geometry  ST_GeometricMedian (geometry  g , float8  tolerance , int  maxIter , boolean  fai
lIfNotConverged);
```

Parameters

Parameter	Description
g	The geometry object that you want to specify.
tolerance	The tolerance that you want to specify.
maxIter	The maximum number of iterations.
failIfNotConverged	Specifies whether to report an error if this function cannot find the geometric median of the geometry object after the maximum number of iterations is reached.

Description

- This function performs iterations until the distance change between successive iterations is less than the value of the tolerance parameter. If the distance change between successive iterations remains

greater than or equal to the value of the tolerance parameter after the maximum number of iterations is reached, this function returns an error and exits unless you set the failIfNotConverged parameter to False.

- If you do not specify the tolerance parameter, this function calculates the default tolerance based on the extent of the geometry object that you specify.
- If the m coordinates of the points of the geometry object that you specify are available, the m coordinates are interpreted as the relative weights of the points.

Examples

The following example shows the difference between the ST_GeometricMedian function and the ST_Centroid function:

```
SELECT ST_AsText(ST_GeometricMedian(geom)) as GeometricMedian, ST_AsText(ST_Centroid(geom))
as Centroid
    from (SELECT 'MULTIPOINT((0 0), (0 1), (1 1), (2 0))'::geometry as geom) as test;
              geometricmedian               |     centroid
---------------------------------------------+-----------------
 POINT(0.665913838138866 0.666097415551148) | POINT(0.75 0.5)
(1 row)
```

4.5.33. ST_HasArc

This topic describes the ST_HasArc function. This function checks whether a geometry object or a GeometryCollection object contains a ring. If the object contains a ring, this function returns True. Otherwise, this function returns False.

Syntax

```
boolean  ST_HasArc(geometry  geomA);
```

Parameters

Parameter	Description
geomA	The geometry object or GeometryCollection object that you want to specify.

Description

- This function supports 3D objects and does not delete z coordinates.
- This function supports circular strings and curves.

Examples

Check whether an object contains a ring by using the default parameter settings.

```
SELECT ST_HasArc('GeometryCollection(POINT(1 0),CIRCULARSTRING(1 2,3 4,5 6))'::geometry);
 st_hasarc
-----------
 t
(1 row)
```

4.5.34. ST_Intersects

This topic describes the ST_Intersects function. This function checks whether two geometry objects or two geography objects intersect. If the geometry objects or the geography objects share any part of space, the objects intersect.

Syntax

```
boolean  ST_Intersects(geometry  geomA , geometry  geomB);
boolean  ST_Intersects(geography  geogA , geography  geogB);
```

Parameters

Parameter	Description
geomA	The first geometry object that you want to specify.
geomB	The second geometry object that you want to specify.
geogA	The first geography object that you want to specify.

Parameter	Description
geogB	The second geography object that you want to specify.

Description

- If you specify two geography objects, this function uses a tolerance of 0.00001 meter. Any points that are within 0.00001 meter to each other are considered as intersected. In addition, this function uses a sphere model rather than an ellipsoid model.
- If the relationship between the objects that you specify is Overlap, Touch, or Within, the objects intersect.
- This function automatically compares the bounding boxes of the geometry objects that you specify by using all available indexes on the geometry objects.

Examples

Check whether two objects intersect by using the default parameter settings.

```
SELECT ST_Intersects('LINESTRING(0 0,1 1)'::geometry,'LINESTRING(0 1,1 0)'::geometry);
 st_intersects
---------------
 t
(1 row)
```

4.5.35. ST_Length

This topic describes the ST_Length function. This function calculates the length of a LineString object or MultiLineString object within the 2D space of the object.

Syntax

```
float  ST_Length(geometry  a2dlinestring);
float  ST_Length(geography  geog , boolean  useSpheroid);
```

Parameters

Parameter	Description
a2dlinestring	The 2D linear geometry object that you want to specify.
geog	The geography object that you want to specify.
useSpheroid	Specifies whether to use an ellipsoid model.

Description

- If you specify a geometry object, the following rules apply:

- If the geometry object is a LineString, MultiLineString, ST_Curve, or ST_MultiCurve, this function returns a 2D Cartesian length.
- If the geometry object is a surface, this function returns 0. To obtain the length of a surface within the 2D space of the surface, use the ST_Perimeter function.
- The length that is returned is measured in the unit that is specified by the spatial reference system of the geometry object.

- If you specify a geography object, this function returns an inverse geodesic distance, which is measured in meters.

Examples

Calculate the length of a geometry object within the 2D space of the object by using the default parameter settings.

```
SELECT ST_Length('LINESTRING(0 0,1 1)'::geometry);
    st_length
-----------------
 1.4142135623731
(1 row)
```

4.5.36. ST_Length2D

This topic describes the ST_Length2D function. This function calculates the length of a LineString object or MultiLineString object within the 2D space of the object. This function is equivalent to the ST_Length function.

Syntax

```
float  ST_Length2D(geometry  a2dlinestring);
```

Parameters

Parameter	Description
a2dlinestring	The 2D linear geometry object that you want to specify.

Examples

Calculate the length of a geometry object within the 2D space of the object by using the default parameter settings.

```
SELECT ST_Length('LINESTRING(0 0,1 1)'::geometry);
    st_length
-----------------
 1.4142135623731
(1 row)
```

4.5.37. ST_3DLength

This topic describes the ST_3DLength function. This function calculates the length of a LineString object or MultiLineString object within the 3D space of the object.

Syntax

```
float  ST_3DLength(geometry  a3dlinestring);
```

Parameters

Parameter	Description
a3dlinestring	The 3D LineString object that you want to specify.

Description

If you specify a 2D object, this function calculates the length of the object within the 2D space of the object. In this case, this function is equivalent to the ST_Length function and the ST_Length2D function.

Examples

Calculate the length of a geometry object within the 3D space of the object by using the default parameter settings.

```
SELECT ST_3DLength('LINESTRING(0 0 0,1 1 1)'::geometry);
   st_3dlength
------------------
 1.73205080756888
(1 row)
```

4.5.38. ST_LengthSpheroid

This topic describes the ST_LengthSpheroid function. This function calculates the 2D or 3D length of a geometry object on an ellipsoid.

Syntax

```
float  ST_LengthSpheroid(geometry  aGeometry , spheroid  aSpheroid);
```

Parameters

Parameter	Description
aGeometry	The geometry object that you want to specify.
aSpheroid	The ellipsoid that you want to define.

Description

- This function is suitable for geometry objects whose coordinates are represented by longitudes and latitudes and whose lengths are not expected to be reprojected.
- This function supports 3D objects and does not delete z coordinates.

Examples

Calculate the 2D or 3D length of a geometry object by using the default parameter settings.

```
SELECT ST_LengthSpheroid('LINESTRING(0 0,1 1)'::geometry,'SPHEROID["WGS 84",6378137,298.257
223563]');
 st_lengthspheroid
-------------------
   156899.56829134
(1 row)
```

4.5.39. ST_Length2DSpheroid

This topic describes the ST_Length2DSpheroid function. This function calculates the 2D length of a geometry object on an ellipsoid.

Syntax

```
float  ST_Length2DSpheroid(geometry  aGeometry , spheroid  aSpheroid);
```

Parameters

Parameter	Description
aGeometry	The geometry object that you want to specify.
aSpheroid	The ellipsoid that you want to define.

Description

This function is suitable for geometry objects whose coordinates are represented by longitudes and latitudes and whose lengths are not expected to be reprojected.

Examples

Calculate the 2D length of a geometry object by using the default parameter settings.

```
SELECT ST_Length2DSpheroid('LINESTRING(0 0,1 1)'::geometry,'SPHEROID["WGS 84",6378137,298.2
57223563]');
 st_length2dspheroid
---------------------
     156899.56829134
(1 row)
```

4.5.40. ST_LongestLine

This topic describes the ST_LongestLine function. This function calculates the line that represents the farthest distance between the points of two 2D geometry objects.

Syntax

```
geometry  ST_LongestLine(geometry  g1 , geometry  g2);
```

Parameters

Parameter	Description
g1	The first geometry object that you want to specify.
g2	The second geometry object that you want to specify.

Examples

Calculate the line that represents the farthest distance between the points of two 2D geometry objects by using the default parameter settings.

```
SELECT ST_AsText(ST_LongestLine('POINT(0 0)'::geometry,'LINESTRING(0 0,1 1)'::geometry));
      st_astext
--------------------
 LINESTRING(0 0,1 1)
(1 row)
```

4.5.41. ST_OrderingEquals

This topic describes the ST_OrderingEquals function. This function checks whether two geometry objects are equal and their coordinates are in the same order. If the two geometry objects are equal and their coordinates are in the same order, this function returns True. Otherwise, this function returns False.

Syntax

```
boolean  ST_OrderingEquals(geometry  a , geometry  b);
```

Parameters

Parameter	Description
a	The first geometry object that you want to specify.
b	The second geometry object that you want to specify.

Examples

The following example shows the difference between the ST_OrderingEquals function and the ST_Equals function:

```
SELECT ST_Equals(g1,g2),ST_OrderingEquals(g1,g2) FROM (SELECT 'LINESTRING(0 1,2 3)'::geometry as g1,'LINESTRING(2 3,0 1)'::geometry as g2) as test;
 st_equals | st_orderingequals
-----------+-------------------
 t         | f
(1 row)
```

4.5.42. ST_Overlaps

This topic describes the ST_Overlaps function. This function checks whether two geometry objects spatially overlap but one is not included in the other. If the two geometry objects spatially overlap but one is not included in the other, this function returns True. Otherwise, this function returns False.

Syntax

```
boolean  ST_Overlaps(geometry  a , geometry  b);
```

Parameters

Parameter	Description
a	The first geometry object that you want to specify.
b	The second geometry object that you want to specify.

Description

- This function does not support GeometryCollection objects.
- This function generates a bounding box, which allows you to use the indexes that are available on the geometry objects that you specify. If you do not want to use indexes, you can use the _ST_Overlaps function.

Examples

Compare two geometry objects by using the default parameter settings.

```
SELECT ST_Overlaps('LINESTRING(0 0,0 2)'::geometry,'LINESTRING(0 1,0 3)'::geometry);
 st_overlaps
-------------
 t
(1 row)
```

4.5.43. ST_Perimeter

This topic describes the ST_Perimeter function. This function calculates the perimeter of a geometry object or geography object.

Syntax

```
float  ST_Perimeter(geometry  g1);
float  ST_Perimeter(geography  geog , boolean  useSpheroid);
```

Parameters

Parameter	Description
g1	The geometry object that you want to specify.
geog	The geography object that you want to specify.
useSpheroid	Specifies whether to use an ellipsoid.

Description

- If the geometry or geography object that you specify is an ST_Surface object or an ST_MultiSurface object, the ST_Perimeter function returns the 2D perimeter of the object. An ST_MultiSurface object is a collection of surfaces, which can be polygons or MultiPolygons.
- If you specify a non-areal object, the ST_Perimeter function returns 0.
- If you want to calculate the perimeter of a linear geometry object, we recommend that you use the ST_Length function.
- If you specify a geometry object, the ST_Perimeter function returns the perimeter of the object in the unit that is specified by the spatial reference system of the object.
- If you specify a geography object, the ST_Perimeter function returns an inverse geodesic distance, which is measured in meters.

Examples

Calculate the perimeter of an object by using the default parameter settings.

```
SELECT ST_Perimeter('POLYGON((0 0,0 2,2 2,2 0,0 0))'::geometry);
 st_perimeter
--------------
            8
(1 row)
```

4.5.44. ST_Perimeter2D

This topic describes the ST_Perimeter2D function. This function calculates the 2D perimeter of a geometry object. This function is equivalent to the ST_Perimeter function.

Syntax

```
float  ST_Perimeter2D(geometry  geomA);
```

Parameters

Parameter	Description
geomA	The geometry object that you want to specify.

Examples

Calculate the 2D perimeter of a geometry object by using the default parameter settings.

```
SELECT ST_Perimeter2D('POLYGON((0 0,0 2,2 2,2 0,0 0))'::geometry);
 st_perimeter
--------------
            8
(1 row)
```

4.5.45. ST_3DPerimeter

This topic describes the ST_3DPerimeter function. This function calculates the 3D perimeter of a geometry object.

Syntax

```
float  ST_3DPerimeter(geometry  geomA);
```

Parameters

Parameter	Description
geomA	The geometry object that you want to specify.

Description

If you specify a 2D geometry object, this function returns the 2D perimeter of the object.

Examples

Calculate the 3D perimeter of a geometry object by using the default parameter settings.

```
SELECT ST_3DPerimeter('POLYGON((0 0 0,0 2 0,2 2 2,2 0 2,0 0 0))'::geometry);
  st_3dperimeter
------------------
 9.65685424949238
(1 row)
```

4.5.46. ST_PointOnSurface

This topic describes the ST_PointOnSurface function. This function returns a point that intersects with the surface of a geometry object.

Syntax

```
geometry  ST_PointOnSurface(geometry  g1);
```

Parameters

Parameter	Description
g1	The geometry object that you want to specify.

Description

This function supports 3D objects and does not delete z coordinates.

Examples

Find a point that intersects with the surface of a geometry object by using the default parameter settings.

```
SELECT ST_AsText(ST_PointOnSurface('POLYGON((0 0,0 2,2 2,2 0,0 0))'::geometry));
 st_astext
------------
 POINT(1 1)
(1 row)
```

4.5.47. ST_Project

This topic describes the ST_Project function. This function reprojects a point based on a distance measured in meters and an azimuth. North is azimuth 0, east is azimuth 90, south is azimuth 180, and west is azimuth 270.

Syntax

```
geography  ST_Project(geography  g1 , float  distance , float  azimuth);
```

Parameters

Parameter	Description
g1	The geography object that you want to specify.
distance	The distance that you want to specify. Unit: meters.
azimuth	The heading or bearing in navigation.

Description

This function supports negative distances and non-normalized azimuths.

Examples

Reproject a point by using the default parameter settings.

```
SELECT ST_AsText(ST_Project('POINT(0 0)'::geography, 10, pi()));
          st_astext
-----------------------------
 POINT(0 -0.000090436947705)
(1 row)
```

4.5.48. ST_Relate

This topic describes the ST_Relate function. This function checks whether two geometry objects are spatially related. For example, this function checks whether two geometry objects intersect or whether one geometry object is included in the other.

Syntax

```
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);
```

Parameters

Parameter	Description
geomA	The first geometry object that you want to specify.
geomB	The second geometry object that you want to specify.
intersectionMatrixPattern	The DE-9IM matrix pattern that you want to specify.
boundaryNodeRule	The boundary node rule that you want to specify.

Description

- Version 1: This function checks whether the geometry objects that you specify spatially intersect at their interior rings, boundaries, or exterior rings in compliance with the DE-9IM matrix pattern that you define. If all the specified conditions are met, this function returns True. Otherwise, this function returns False.
  - This usage is suitable especially when you need to test compound checks of road intersections or crosses in one step.
  - This function does not support GeometryCollection objects.
- Version 2: This function returns the DE-9IM matrices of the geometry objects that you specify. This function does not support GeometryCollection objects.
- Version 3: Version 3 is similar to version 2, except that version 3 allows you to define a boundary node rule.
  - 1: OGC/MOD2

- 1: SSL/TLS
  - 2: Endpoint
  - 3: Multivalent Endpoint
  - 4: Monovalent Endpoint

Examples

- Version 1:

```
SELECT ST_Relate('LINESTRING(0 0,0 1)'::geometry, 'LINESTRING(0 0,1 1)'::geometry,'FF1F00
102');
 st_relate
-----------
 t
(1 row)
```

- Version 2:

```
SELECT ST_Relate('LINESTRING(0 0,0 1)'::geometry, 'LINESTRING(0 0,1 1)'::geometry);
 st_relate
-----------
 FF1F00102
(1 row)
```

4.5.49. ST_RelateMatch

This topic describes the ST_RelateMatch function. This function checks whether an intersection matrix complies with an intersection pattern.

Syntax

```
boolean  ST_RelateMatch(text  intersectionMatrix , text  intersectionMatrixPattern);
```

Parameters

Parameter	Description
intersectionMatrix	The intersection matrix that you want to specify.
intersectionMatrixPattern	The intersecting pattern that you want to specify.

Examples

Check an intersection matrix by using the default parameter settings.

```
SELECT ST_RelateMatch('FF1F00102', 'FF*******') ;
 st_relatematch
----------------
 t
(1 row)
```

4.5.50. ST_ShortestLine

This topic describes the ST_ShortestLine function. This function calculates the 2D line that represents the shortest distance between two geometry objects.

Syntax

```
geometry  ST_ShortestLine(geometry  g1 , geometry  g2);
```

Parameters

Parameter	Description
g1	The first geometry object that you want to specify.
g2	The second geometry object that you want to specify.

Description

- If this function finds more than one line that represents the shortest distance between the geometry objects you specify, this function returns only the first line that is found.
- If the geometry objects that you specify intersect at only one point, this function returns a line whose start point and end point are both the intersection point.
- If the geometry objects that you specify intersect at more than one point, this function returns a line whose start point and end point are both one of the intersection points.
- The line that is returned by this function always starts in the first geometry object that you specify and ends in the second geometry object that you specify.
- For the same geometry objects, this function and the ST_Distance function always return the same result.

Examples

Calculate a 2D line by using the default parameter settings.

```
SELECT ST_AsText(ST_ShortestLine('POINT(0 0)'::geometry,'LINESTRING(0 2,1 1)'::geometry));
      st_astext
---------------------
 LINESTRING(0 0,1 1)
(1 row)
```

4.5.51. ST_Touches

This topic describes the ST_Touches function. This function checks whether two geometry objects intersect.

Syntax

```
boolean  ST_Touches(geometry  g1 , geometry  g2);
```

Parameters

Parameter	Description
g1	The first geometry object that you want to specify.
g2	The second geometry object that you want to specify.

Description

- If the points at which the two geometry objects that you specify intersect are concentrated and lie in the boundaries of the geometry objects, this function returns True.
- The intersection between the two geometry objects that you specify meets one of the following DE-9IM conditions:
  - FT*******
  - F**T*****
  - F***T****
- This function can check the intersection between two areas, between two lines, between a line and an area, between a point and an area, and between a point and a line.
- This function cannot check the intersection between two points.
- This function does not support GeometryCollection objects.
- This function generates a bounding box, which allows you to use the indexes that are available on the geometry objects that you specify. If you do not want to use the indexes, we recommend that you use the _ST_Touches function.

Examples

```
SELECT ST_Touches('LINESTRING(0 0,0 2)'::geometry, 'LINESTRING(0 2,0 3)'::geometry);
 st_touches
------------
 t
(1 row)
```

- Obtain an intersection that does not meet one of the DE-9IM conditions.

```
SELECT ST_Touches(g1,g2), ST_Relate(g1,g2) FROM (SELECT 'LINESTRING(0 0,0 2)'::geometry a
s g1, 'LINESTRING(0 1,0 3)'::geometry as g2) as test;
 st_touches | st_relate
------------+-----------
 f          | 1010F0102
(1 row)
```

4.5.52. ST_Within

This topic describes the ST_Within function. This function checks whether a geometry object is included in another geometry object.

Syntax

```
boolean  ST_Within(geometry  A , geometry  B);
```

Parameters

Parameter	Description
A	The first geometry object that you want to specify. The first geometry object is referred to as Geometry Object A.
B	The second geometry object that you want to specify. The second geometry object is referred to as Geometry Object B.

Description

- The two geometry objects that you specify in this function must use the same projection method and must have the same spatial reference identifier (SRID).
- If both ST_Within(A,B) and ST_Within(B,A) return True, Geometry Object A and Geometry B are spatially equal.
- This function does not support GeometryCollection objects.
- You cannot specify invalid geometry objects for this function. If you specify invalid geometry objects, this function returns an incorrect result.
- This function generates a bounding box, which allows you to use the indexes that are available on the geometry objects that you specify. If you do not want to use the indexes, we recommend that you use the _ST_Within function.

Examples

```
SELECT ST_Within('POLYGON((1 1,1 2,2 2,2 1,1 1))'::geometry,'POLYGON((0 0,0 3,3 3,3 0,0 0))
'::geometry);
 st_within
-----------
 t
(1 row)
```

4.6. Processing functions

4.6.1. ST_Buffer

This topic describes the ST_Buffer function. This function returns a geometry object representing all points whose distance from the input geometry object is less than or equal to the specified distance with the input geometry object as the center. The returned geometry object is the buffer that surrounds the input geometry object. The ST_Buffer function calculates the result based on a spatial reference system (SRS) of the specified geometry object.

Syntax

```
geometry  ST_Buffer(geometry  g1 , float radiusOfBuffer);
geometry  ST_Buffer(geometry g1 , float radiusOfBuffer , integer numSegQuarterCircle);
geometry  ST_Buffer(geometry g1 , float radiusOfBuffer , text bufferStyleParameters);
geography ST_Buffer(geography g1 , float radiusOfBufferInMeters);
geography ST_Buffer(geography g1 , float radiusOfBuffer , integer numSegQuarterCircle);
geography ST_Buffer(geography g1 , float radiusOfBuffer , text bufferStyleParameters);
```

Parameters

Parameter	Description
g1	The geometry or geography object that you want to specify.
radiusOfBuffer	The radius of a buffer. This parameter is valid only when you specify a geometry object.
numSegQuarterCircle	Specifies the number of segments that are used to approximate a quarter circle. Default value: 7.
bufferStyleParameters	The buffer. The value of this parameter is key-value pairs that are separated by spaces.
radiusOfBufferInMeters	The radius of a buffer. Unit: meters. This parameter is valid only when you specify a geography object.

Description

- If you specify a polygon object and set the radius of the buffer to a negative value, the ST_Buffer function scales the polygon object rather than expanding the object.
- If you specify a geography object, the radius of the buffer is measured in the unit that is specified by the SRS of the geography object.
- If you specify a geography object, the ST_Buffer function is a wrapper around a geometry object. The ST_Buffer function first selects a proper Spatial Reference Identifier (SRID) that fits the bounding box of the geography object. For the coordinate system, the ST_Buffer function preferentially uses the Universal Transverse Mercator (UTM), Lambert Azimuthal Equal Area (LAEA), or Universal Polar Stereographic (UPS). In the worst scenario, the Mercator projection is used. Then, the ST_Buffer function calculates buffers in the SRS of the geography object and transforms the buffers back to the World Geodetic System 1984 (WGS 84) coordinate system.
- The ST_Buffer function ignores z coordinates. If you specify a 3D geometry object, the ST_Buffer function still returns a 2D geometry object that represents a buffer.
- In some cases, the ST_Buffer function is used for radius searches. However, if you create a buffer for a radius search, the search is slow and pointless. We recommend that you use the ST_DWithin

function for radius searches.

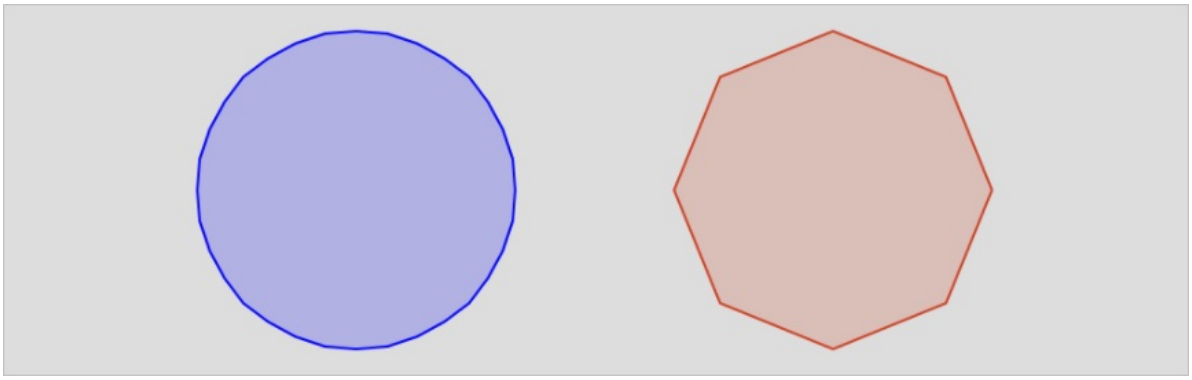
- The following table describes the fields of the bufferStyleParameters parameter.

Field	Description	Type	Default value	Description
quad_segs	The number of segments that are used to approximate a quarter circle.	integer	8	A greater value indicates a smoother quarter circle.
endcap	The endcap style of the buffer.	string	round	Valid values: round, flat, and square.
join	The join style of the buffer.	string	round	Valid values: round, mitre, and bevel.
mitre_limit	The limit of the mitre ratio.	float	5.0	This parameter is valid only when you set the join parameter to mitre.
side	The position of the buffer.	string	--	Valid values: both, left, and right. If you set this parameter to left or right, the ST_Buffer function performs a single-sided buffer on the input geometry object with the buffered side relative to the direction of the line. This parameter takes effect only when you specify a LineString object. This parameter does not affect the results for point objects or polygon objects.

Examples

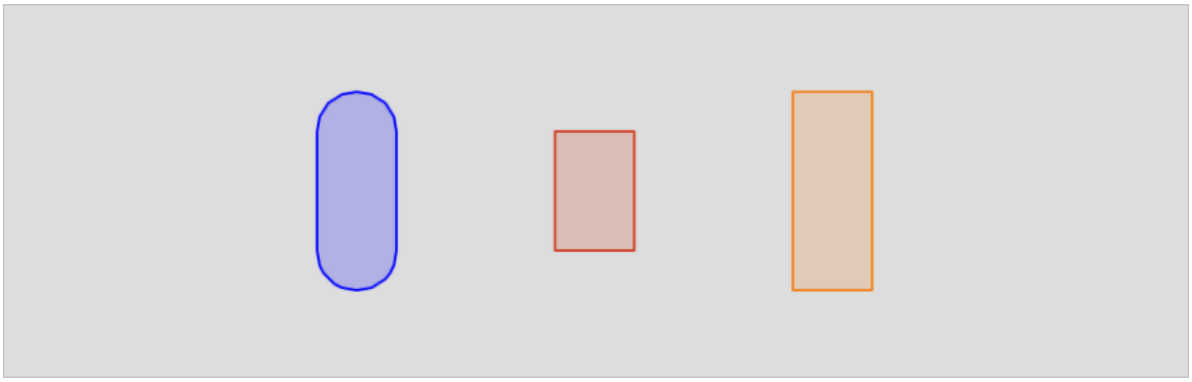
- Results returned with different values of the quad_segs parameter:

```
SELECT ST_Buffer('POINT(0 0)'::geometry,1),ST_Buffer('POINT(3 0)'::geometry,1,'quad_segs=
2');
```



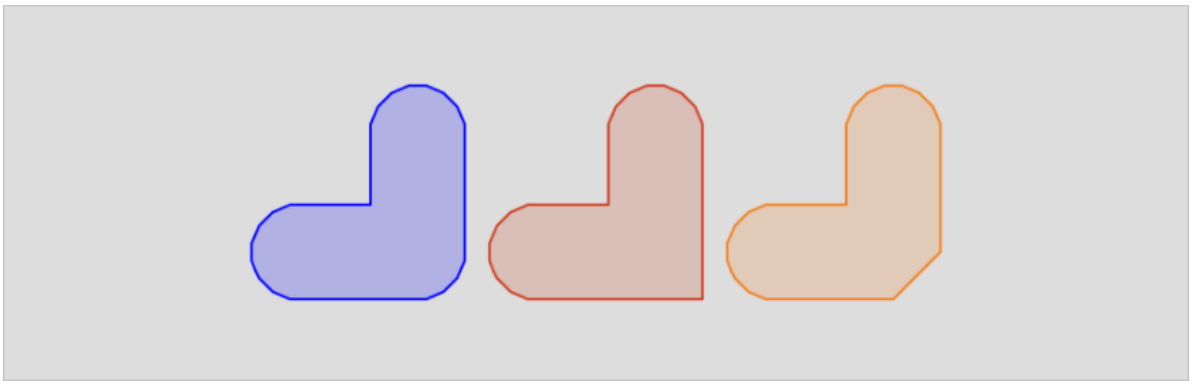
- Results returned with different values of the endcap parameter:

```
SELECT ST_Buffer('LINESTRING(0 0,0 3)'::geometry,1,'endcap=round'),
            ST_Buffer('LINESTRING(6 0,6 3)'::geometry,1,'endcap=flat'),
         ST_Buffer('LINESTRING(12 0,12 3)'::geometry,1,'endcap=square');
```



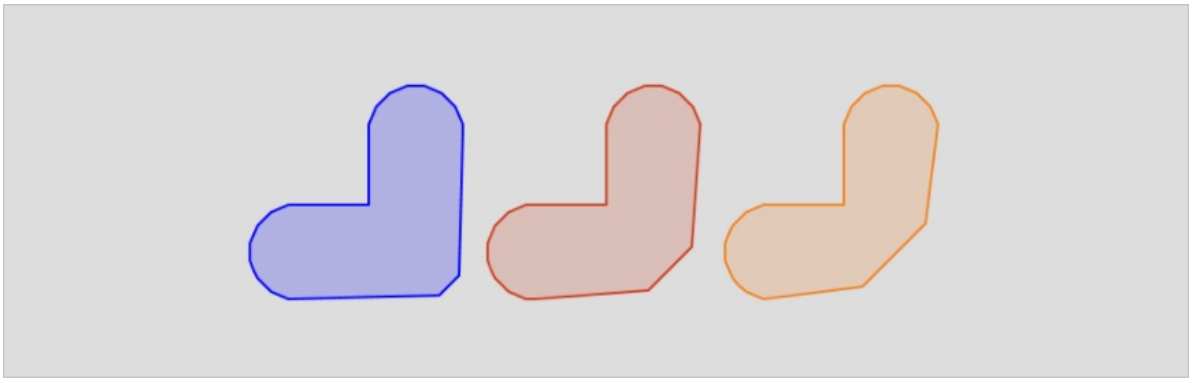
- Results returned with different values of the joins parameter:

```
SELECT ST_Buffer('LINESTRING(0 0,3 0,3 3)'::geometry,1.2,'join=round'),
         ST_Buffer('LINESTRING(6 0,9 0,9 3)'::geometry,1.2,'join=mitre'),
         ST_Buffer('LINESTRING(12 0,15 0,15 3)'::geometry,1.2,'join=bevel');
```



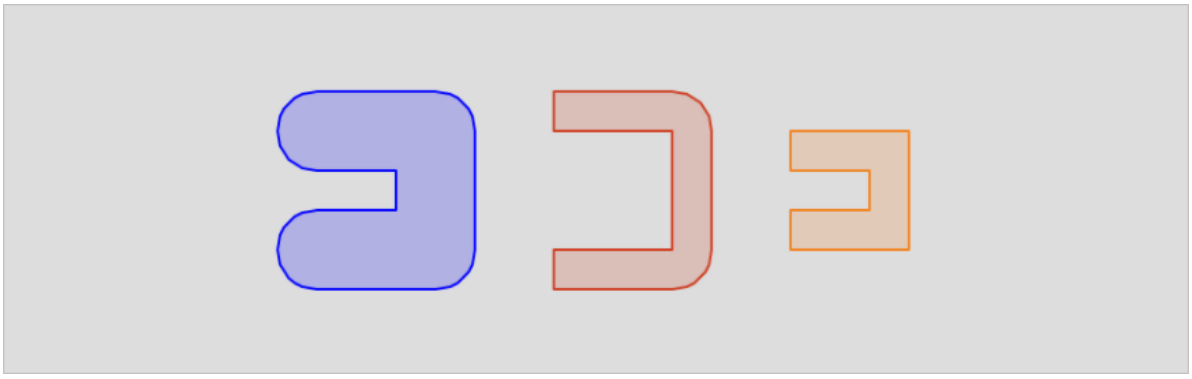
- Results returned with different values of the mitre_limit parameter:

```
SELECT ST_Buffer('LINESTRING(0 0,3 0,3 3)'::geometry,1.2,'join=mitre mitre_limit=1.0'),
         ST_Buffer('LINESTRING(6 0,9 0,9 3)'::geometry,1.2,'join=mitre mitre_limit=0.5'),
         ST_Buffer('LINESTRING(12 0,15 0,15 3)'::geometry,1.2,'join=mitre mitre_limit=0.1
');
```



- Results returned with different values of the side parameter:

```
SELECT ST_Buffer('LINESTRING(0 0,3 0,3 3,0 3)'::geometry,1,'side=both'),
       ST_Buffer('LINESTRING(6 0,9 0,9 3,6 3)'::geometry,1,'side=right'),
       ST_Buffer('LINESTRING(12 0,15 0,15 3,12 3)'::geometry,1,'side=left');
```



4.6.2. ST_BuildArea

This topic describes the ST_BuildArea function. This function creates an areal geometry object formed by the constituent lines of the input geometry object.

Syntax

```
geometry  ST_BuildArea(geometry  a);
```

Parameters

Parameter	Description
a	The geometry object that you want to specify.

Description

- This function supports LineString objects, MultiLineString objects, polygon objects, MultiPolygon objects, and GeometryCollection objects.
- This function assumes all inner geometry objects represent holes.
- This function returns a polygon object or MultiPolygon object. If the constituent lines of the input geometry object cannot form a polygon object, the ST_BuildArea function returns null.

Examples

Results returned by using the default parameter settings:

```
SELECT ST_AsText(ST_BuildArea('MultiLineString((0 0,0 1),(0 1,1 1),(1 1,0 0))'::geometry));
         st_astext
----------------------------
 POLYGON((0 0,0 1,1 1,0 0))
(1 row)
```

4.6.3. ST_ClipByBox2D

This topic describes the ST_ClipByBox2D function. This function clips an input geometry object by using a 2D box in a fast way.

Syntax

```
geometry  ST_ClipByBox2D(geometry  geom , box2d  box);
```

Parameters

Parameter	Description
geom	The geometry object that you want to specify.
box	The 2D box that is used for clipping.

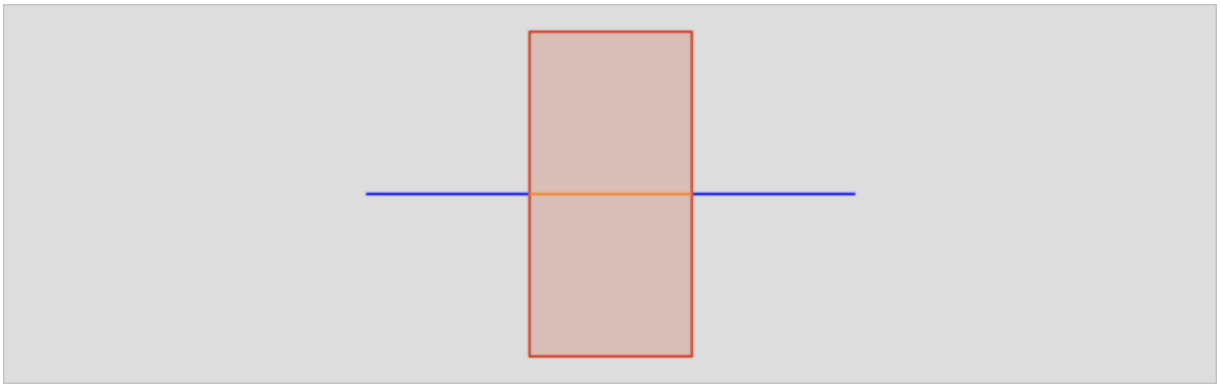
Description

- This function may return an invalid geometry object. This function may introduce a polygon object with self-intersections.
- If the input geometry object has an invalid topology, no error is reported.

Examples

Results returned by using the default parameter settings:

```
SELECT ST_AsText(ST_ClipByBox2D('LINESTRING(0 1,2 1)'::geometry,ST_MakeEnvelope(0,0,1,2)));
      st_astext
---------------------
 LINESTRING(0 1,1 1)
(1 row)
```



4.6.4. ST_Collect

This topic describes the ST_Collect function. This function returns a geometry object that represents the collection of input geometry objects.

Syntax

```
geometry  ST_Collect(geometry set  g1Field);
geometry  ST_Collect(geometry  g1 , geometry  g2);
geometry  ST_Collect(geometry[]  g1Array);
```

Parameters

Parameter	Description
g1Field	The fields of the geometry objects that you want to specify in the dataset.
g1	The first geometry object that you want to specify.
g2	The second geometry object that you want to specify.
g1Array	An array consisting of the geometry objects you want to specify.

Description

- This function returns a MULTI or GeometryCollection object. The ST_Collect function has two versions:
  - Version 1: The input of the ST_Collect function is two geometry objects. In this case, this function returns a MULTI or GeometryCollection object.
  - Version 2: The ST_Collect function works as an aggregate function. The input of the ST_Collect function is a collection of geometry objects. This function returns the geometry object that is returned by the ST_Geometry function.
- The ST_Collect function supports CircularString objects and Curve objects. This function may not return a MultiCurve object or MULTI object as you expect.
- The ST_Collect function and the ST_Union function are interchangeable. The ST_Collect function

returns a MULTI or GeometryCollection object. The ST_Union function may return a geometry object because this function dissolves the boundaries of the input geometry object. The ST_Union function also splits a LineString object at the intersection of nodes while the ST_Collect function just returns a MultiLineString object.

- In most cases, the ST_Collect function runs faster than the ST_Union function because the ST_Union function dissolves the boundaries of the specified geometry objects or checks whether the constructed MultiPolygon object has intersecting parts. The ST_Collect function collects geometry objects into a MULTI object or collects MULTI objects into a GeometryCollection object.
- You can use the ST_Dump function to convert MULTI objects into geometry objects, and then specify the geometry objects to the ST_Collect function. This prevents the ST_Collect function from returning a GeomotryCollection object for MULTI objects.

Examples

- Execute the following statement to return a MULTI object:

```
SELECT ST_AsText(ST_Collect('POINT(0 0)'::geometry, 'POINT(0 1)'::geometry));
      st_astext
---------------------
 MULTIPOINT(0 0,0 1)
(1 row)
```

- Execute the following statement to return a GeometryCollection object:

```
SELECT ST_AsText(ST_Collect('POINT(0 0)'::geometry, 'LINESTRING(0 2,0 3)'::geometry));
                     st_astext
-----------------------------------------------------
 GEOMETRYCOLLECTION(POINT(0 0),LINESTRING(0 2,0 3))
(1 row)
```

- Execute the following statement to aggregate the input dataset:

```
select ST_AsText(ST_Collect(t.geom)) from (select (ST_DumpPoints(st_buffer('POINT(0 0)'::
geometry,1,'quad_segs=2'))).geom as geom) as t;
                           st_astext
-------------------------------------------------------------
 MULTIPOINT(1 0,0.707106781186548 -0.707106781186547,0 -1,-0.70.
.7106781186546 -0.707106781186549,-1 0,-0.70710678118655 0.7071.
.06781186545,0 1,0.707106781186544 0.707106781186551,1 0)
(1 row)
```

4.6.5. ST_ConcaveHull

This topic describes the ST_ConcaveHull function. This function returns the concave hull of the input geometry objects.

Syntax

```
geometry  ST_ConcaveHull(geometry  geomA , float  targetPercent , boolean  allowHoles);
```

Parameters

Parameter	Description
geomA	The geometry object that you want to specify.
targetPercent	The percent of the area of the concave hull to the area of the convex hull before the ST_ConcaveHull function returns the result.
allowHoles	Specifies whether the ST_ConcaveHull function allows a polygon object with holes. Default value: false.

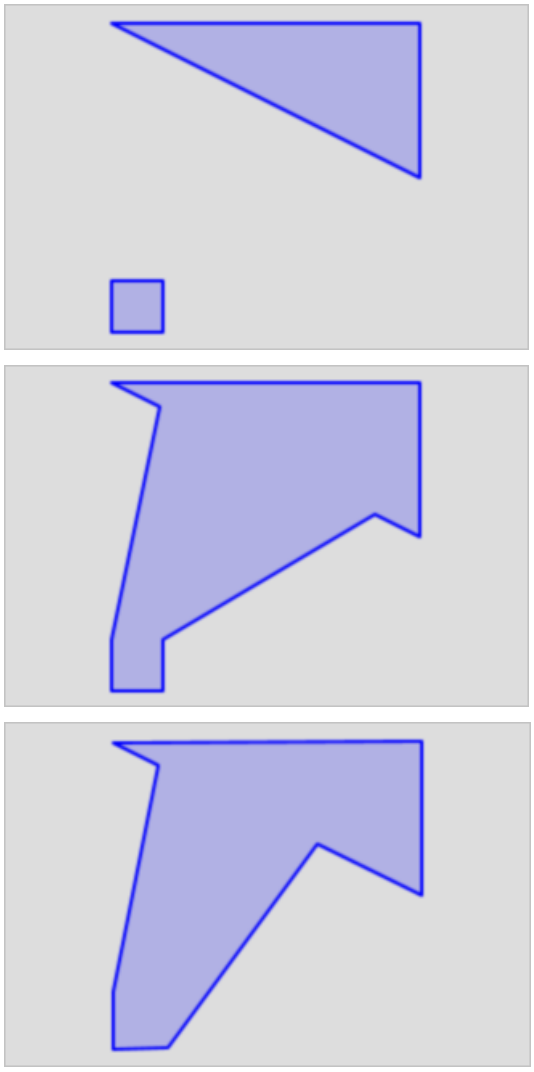
Description

- The dimension of the returned concave hull is not higher than the dimension of the input polygon object.
- If you specify point objects, LineString objects, or GeometryCollection objects, use the ST_Collect function. If you specify polygon objects, use the ST_Union function. If you specify invalid geometry objects, the ST_ConcaveHull function fails.
- A small value of the targetPercent parameter indicates that a long period of time is required to return the concave hull. A small value also increases the possibilities of topology exceptions. However, a small value allows the function to return a more precise concave hull. Suggestions on the configuration of the targetPercent parameter:
  - Set the targetPercent parameter to 0.99. In this case, the ST_ConcaveHull function returns the concave hull in a short period of time. In some scenarios, the function returns the result as fast as calculating the convex hull. In most cases, the proportion of the area of the concave hull to the area of the convex hull is smaller than 99%.
  - Try a value that is smaller than 0.99 to find the most appropriate value. The smaller the value, the longer the response time.
- If you want to reduce the precision of the return value, you can use the ST_SimplifyPreserveTopology function or the ST_SnapToGrid function to process the results that are returned by the ST_ConcaveHull function. In most cases, the ST_SnapToGrid runs faster than the ST_SimplifyPreserveTopology function because the ST_SimplifyPreserveTopology function needs to verify the validity of the returned results. However, the ST_SnapToGrid function may return invalid geometry objects.
- In most cases, the ST_ConcaveHull function is used for MULTI and GeometryCollection objects. The ST_ConcaveHull function is not an aggregate function. If you want to obtain the concave hull of a set of point objects, LineString objects, or polygon objects, you can use this function with the ST_Collect function or the ST_Union function. Example: ST_ConcaveHull(ST_Collect(somepointfield),0.80).
- This function runs faster when it calculates the convex hull of input geometry objects than when it calculates the concave hull. However, the ST_ConcaveHull function encloses all the input geometry objects better and is useful for image recognition.

Examples

Comparison among the original object, the object returned with the targetPercent parameter set to 0.99, and the object returned with the targetPercent parameter set to 0.98:

```
SELECT g,ST_ConcaveHull(g,0.99),ST_ConcaveHull(g,0.98),g from
    (select 'MULTIPOLYGON(((0 0,1 0,1 1,0 1,0 0)),((0 6,6 3,6 6,0 6)))'::geometry as g) as
test
```



4.6.6. ST_ConvexHull

This topic describes the ST_ConvexHull function. This function returns the smallest convex hull of the input geometry objects.

Syntax

```
geometry  ST_ConvexHull(geometry  geomA);
```

Parameters

Parameter	Description
geomA	The geometry object that you want to specify.

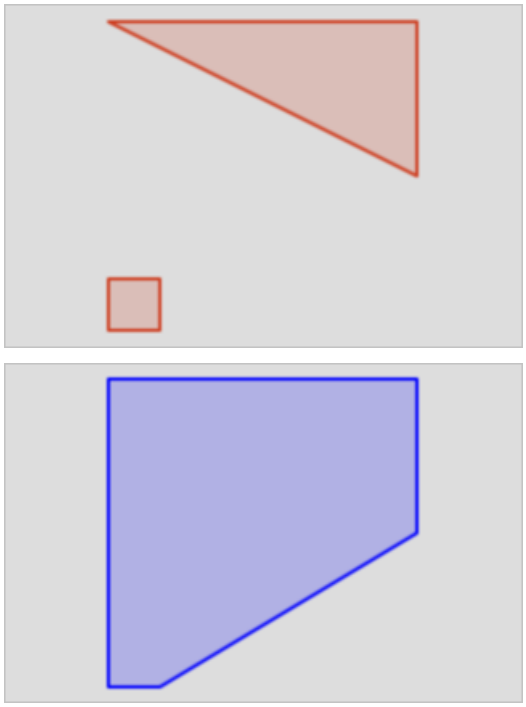
Description

- This function is different from the ST_ConcaveHull function. In most cases, this function is used to determine an affected area based on a set of observation points for an input MULTI object or GeometryCollection object.
- This function supports 3D objects and does not delete z coordinates.

Examples

Comparison between the convex hull and the input geometry object:

```
SELECT ST_ConvexHull(g),g from
    (select 'MULTIPOLYGON(((0 0,1 0,1 1,0 1,0 0)),((0 6,6 3,6 6,0 6)))'::geometry as g) as
test
```



4.6.7. ST_CurveToLine

This topic describes the ST_CurveToLine function. This function converts a CircularString object into a regular LineString object or converts a CurvedPolygon object into a regular Polygon object.

Syntax

```
geometry  ST_CurveToLine(geometry  curveGeom , float  tolerance , integer  toleranceType ,
integer  flags);
```

Parameters

Parameter	Description
curveGeom	The geometry object that you want to specify.

Parameter	Description
tolerance	The tolerance that you want to specify. Default value: 0.
toleranceType	The interpretation of the tolerance parameter. Valid values: • 0: This is the default value. The value of the tolerance parameter indicates the maximum number of segments that are allowed for each quadrant. • 1: The value of the tolerance parameter indicates the maximum deviation between a line and a curve based on the source unit. • 2: The value of the tolerance parameter indicates the maximum angle in radians between generated radii.
flags	The bitfield that you want to specify. Valid values: • 0: This is the default value. • 1: This value indicates that the output is a symmetric object regardless of directions. • 2: This value indicates that the angle is retained to prevent the angle or segment lengths from being reduced when the function generates a symmetric output. This parameter does not take effect when the symmetric output flag is off.

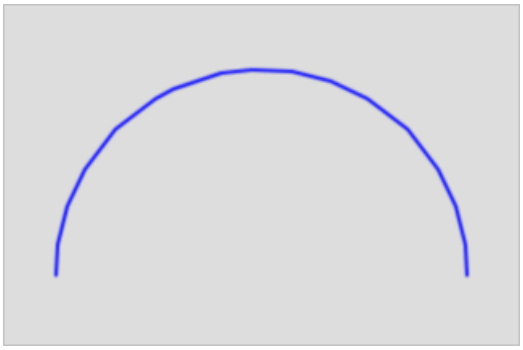
Description

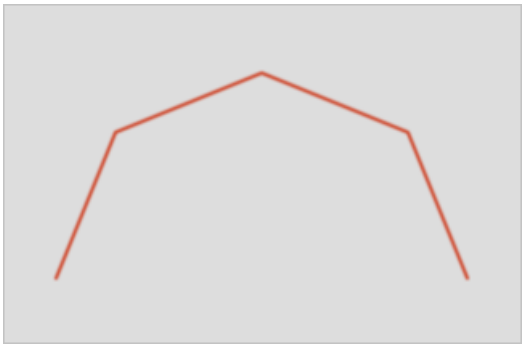
- This function converts each curved geometry object or segment into a linear approximation by dividing each quarter circle into 32 segments.
- This function is useful in scenarios in which CircularString objects are not supported.

Examples

Comparison between the returned object by using the default parameter settings and the returned object by using the custom parameter settings:

```
select ST_CurveToLine(g),ST_CurveToLine(g,pi()/4,2) from (select 'CIRCULARSTRING(0 0,0.5 0.
5,1 0)'::geometry as g) as test;
```





4.6.8. ST_DelaunayTriangles

This topic describes the ST_DelaunayTriangles function. This function returns a Delaunay triangulation around the vertices of the input geometry object.

Syntax

```
geometry ST_DelaunayTriangles(geometry g1 , float tolerance , int4 flags);
```

Parameters

Parameter	Description
g1	The geometry object that you want to specify.
tolerance	The tolerance of the vertices of the input geometry object.
flags	The bitfield that you want to specify. Valid values: • 0: This value indicates that the function returns the collection of polygon objects. This is the default value. • 1: This value indicates that the function return a MultiLineString object. • 2: This value indicates that the function returns a triangulated irregular network (TIN) surface.

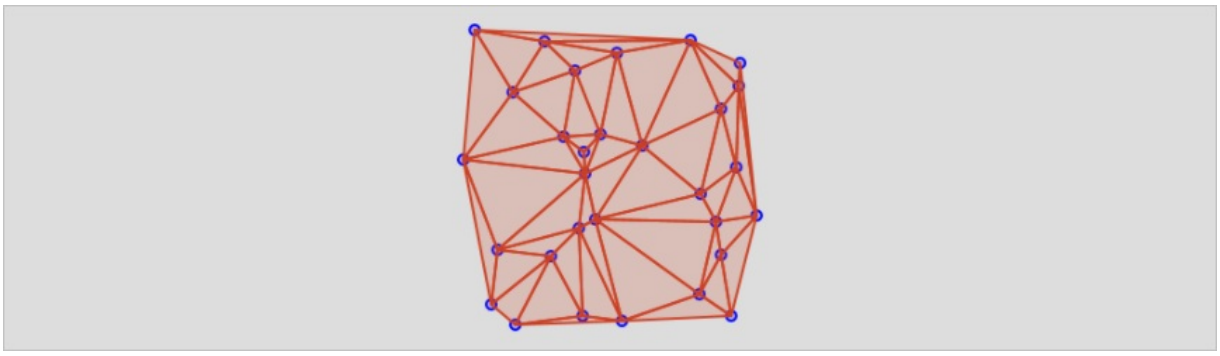
Description

This function supports triangles and triangulated irregular network (TIN) surfaces.

Examples

Results returned by using the default parameter settings:

```
select g,ST_DelaunayTriangles(g) from (select ST_GeneratePoints('POLYGON((0 0,1 0,1 1,0 1,0
0))'::geometry,30) as g) as t;
```



4.6.9. ST_Different

This topic describes the ST_Different function. This function returns a geometry object representing the part that Geometry Object A does not intersect with Geometry Object B.

Syntax

```
geometry  ST_Difference(geometry  geomA , geometry  geomB);
```

Parameters

Parameter	Description
geomA	Geometry Object A that you want to specify.
geomB	Geometry Object B that you want to specify.

Description

- If Geometry Object A is completely contained in Geometry Object B, the ST_Different returns an empty object.
- The order of the input geometry objects affects the output geometry object.
- This function does not support GeometryCollection objects.
- This function supports 3D objects and does not delete z coordinates. This function processes only the x and y coordinates of the input geometry objects but returns the output geometry object with the original z coordinates.

Examples

Results returned by using the default parameter settings:

```
SELECT ST_AsText(ST_Difference('LINESTRING(0 0,0 2)'::geometry,'LINESTRING(0 1,0 3)'::geometry));
      st_astext
--------------------
 LINESTRING(0 0,0 1)
(1 row)
```

4.6.10. ST_Dump

This topic describes the ST_Dump function. This function returns a set of geometry_dump rows of the input geometry object.

Syntax

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

Parameters

Parameter	Description
g1	The geometry object that you want to specify.

Description

- This function extracts the components of the input geometry object and returns a set of geometry_dump rows of the input geometry object. Each value of a row contains a geometry object and an array of integers. The geometry object is represented as geom and the array of integers is represented as path.
  - If you specify a point object, LineString object, or polygon object, this function returns a single record that consists of an empty path array and the input geometry as geom.
  - If you specify a collection object or MULTI object, this function returns the components of the collection and an array of integers that represents the position of each component.
- This function is useful for expanding geometry objects. The ST_Dump function is the reverse of a GROUP BY clause. For example, you can use this function to expand a MultiPolygon object to a polygon object.
- This function supports circular strings, curves, polyhedral surfaces, triangles, triangulated irregular network (TIN) surfaces, and 3D objects.

Examples

Results returned by using the default parameter settings:

```
SELECT (t.dump).path,ST_AsText((t.dump).geom) from (select ST_Dump('MULTILINESTRING((0 0,0
2),(0 1,0 3))'::geometry) as dump) as t;
 path |      st_astext
------+--------------------
 {1}  | LINESTRING(0 0,0 2)
 {2}  | LINESTRING(0 1,0 3)
(2 rows)
```

4.6.11. ST_DumpPoints

This topic describes the ST_DumpPoints function. This function returns a set of geometry_dump rows of the input geometry object. Each value of a row contains a geometry object and an array of integers. The geometry object is represented as geom and the array of integers is represented as path.

Syntax

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

Parameters

Parameter	Description
geom	The geometry object that you want to specify.

Description

- The value of the path parameter is an array of the position coordinates of components in the input geometry object. The component of the path parameter is an index reference enumerating the points of the input geometry object.
  - If you specify a LineString object, an array of {i} that represents the position coordinates of all points in the LineString object is returned.
  - If you specify a polygon object, an array of {i,j} is returned. The field i represents the position order of rings. The value 1 indicates the outermost ring. The field j represents the position order of points in the ring. The position starts from 1.
- This function supports circular strings, curves, polyhedral surfaces, triangles, triangulated irregular network (TIN) surfaces, and 3D objects.

Examples

Results returned by using the default parameter settings:

```
SELECT (t.dump).path,ST_AsText((t.dump).geom) from (select ST_DumpPoints('MULTILINESTRING((
0 0,0 2),(0 1,0 3))'::geometry) as dump) as t;
 path  | st_astext
-------+------------
 {1,1} | POINT(0 0)
 {1,2} | POINT(0 2)
 {2,1} | POINT(0 1)
 {2,2} | POINT(0 3)
(4 rows)
```

4.6.12. ST_DumpRings

This topic describes the ST_DumpRings function. This function extracts the outer rings and inner rings of the input polygon object.

Syntax

```
geometry_dump[]  ST_DumpRings(geometry  aPolygon);
```

Parameters

Parameter	Description
aPolygon	The polygon object or that you want to specify.

Description

- This function returns a set of geometry_dump rows of the input polygon object. Each value of a row contains the geom field and an array of the path field.
  - The path field indicates the indexes of rings. The value 0 indicates the outer ring. A value that is greater than 0 indicates an inner ring.
  - The geom field represents the corresponding ring as a polygon object.
- This function supports only polygon objects. The MultiPolygon objects are not supported.
- This function supports 3D objects.

Examples

Results returned by using the default parameter settings:

```
SELECT (t.dump).path,ST_AsText((t.dump).geom) from (select ST_DumpRings('POLYGON((0 0,0 4,4
4,4 0,0 0),(1 1,2 1,2 2,1 2,1 1))'::geometry) as dump) as t;
 path |           st_astext
------+--------------------------------
 {0}  | POLYGON((0 0,0 4,4 4,4 0,0 0))
 {1}  | POLYGON((1 1,2 1,2 2,1 2,1 1))
(2 rows)
```

4.6.13. ST_FlipCoordinates

This topic describes the ST_FlipCoordinates function. This function interchanges the x and y coordinates of the input geometry object.

Syntax

```
geometry  ST_FlipCoordinates(geometry  geom);
```

Parameters

Parameter	Description
geom	The geometry object that you want to specify.

Description

- This function applies when the x and y coordinates are entered in reverse. You can use this function to fix this issue.
- This function supports circular strings, curves, polyhedral surfaces, triangles, triangulated irregular network (TIN) surfaces, and 3D objects.
- This function supports m coordinates.

Examples

Results returned by using the default parameter settings:

```
SELECT ST_AsText(ST_FlipCoordinates('POINT(0 1)'::geometry));
 st_astext
------------
 POINT(1 0)
(1 row)
```

4.6.14. ST_GeneratePoints

This topic describes the ST_GeneratePoints function. This function generates a specified number of pseudo-random points within the input geometry object.

Syntax

```
geometry  ST_GeneratePoints(g  geometry , npoints  numeric);
```

Parameters

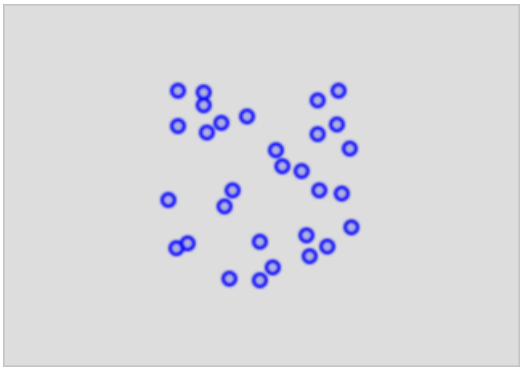
Parameter	Description
geometry	The geometry object that you want to specify.

Parameter	Description
numeric	The number of pseudo-random points.

Examples

Results returned by using the default parameter settings:

```
select ST_GeneratePoints('POLYGON((0 0,1 0,1 1,0 1,0 0))'::geometry,30);
```



4.6.15. ST_Intersection

This topic describes the ST_Intersection function. This function returns a geometry object corresponding to the set of points where the input geometry objects intersect.

Syntax

```
geometry  ST_Intersection(geometry  geomA , geometry  geomB);
geography  ST_Intersection(geography  geogA , geography  geogB);
```

Parameters

Parameter	Description
geomA/geomB	The two geometry objects that you want to specify.
geogA/geogB	The two geography objects that you want to specify.

Description

- If the two input geometry objects do not intersect, this function returns an empty geometry object.
- If you specify geography objects, this function processes the objects by using geometry functions. This function first converts the input geography objects into geometry objects, calculates the intersection of input objects in the planar spatial reference, and then transforms the intersection to a geography object based on the World Geodetic System 1984 (WGS 84) coordinate system.
- This function does not support GeometryCollection objects.
- This function deletes the m coordinates of the input objects.

Examples

Results returned by using the default parameter settings:

```
SELECT ST_AsText(ST_Intersection('POLYGON((0 0,0 2,2 2,2 0,0 0))'::geometry,'POLYGON((0 0,3
0,3 1,0 1,0 0))'::geometry));
           st_astext
-------------------------------
 POLYGON((0 0,0 1,2 1,2 0,0 0))
(1 row)
```

4.6.16. ST_LineToCurve

This topic describes the ST_LineToCurve function. This function converts a LineString or polygon object into a CircularString or Curved Polygon object.

Syntax

```
geometry  ST_LineToCurve(geometry  geomANoncircular);
```

Parameters

Parameter	Description
geomANoncircular	The geometry object that you want to specify.

Description

- The points that are required to represent the equivalent curves are much fewer than the input LineString or polygon object.
- If the input LineString or polygon object is not curved enough to clearly represent a curve, the function returns the input geometry object.

Examples

Results returned by using the default parameter settings:

```
SELECT ST_AsText(ST_LineToCurve(ST_Buffer('POINT(0 0)', 1)));
                 st_astext
--------------------------------------------
 CURVEPOLYGON(CIRCULARSTRING(1 0,-1 0,1 0))
(1 row)
```

4.6.17. ST_MakeValid

This topic describes the ST_MakeValid function. This function converts an invalid input geometry object into a valid geometry object without deleting vertexes.

Syntax

```
geometry  ST_MakeValid(geometry  input);
```

Parameters

Parameter	Description
input	The geometry object that you want to specify.

Description

- This function processes only invalid geometry objects.
- This function supports point objects, MultiPoint objects, LineString objects, MultiLineString objects, polygon objects, MultiPolygon objects, and GeometryCollection objects. The combination of the preceding objects is also supported.
- If all or partial dimensions of the input geometry object are missing, this function returns a geometry object from lower dimensions or a collection of geometry objects from lower dimensions.
- If you specify a polygon object that has self-intersections, this function may return a MULTI object.

Examples

Results returned by specifying a polygon object with self-intersections:

```
SELECT ST_AsText(ST_MakeValid('POLYGON((0 0,0 1,1 0,1 1,0 0))'));
                           st_astext
-------------------------------------------------------------
 MULTIPOLYGON(((0 0,0 1,0.5 0.5,0 0)),((0.5 0.5,1 1,1 0,0.5 0.5.
.)))
(1 row)
```

4.6.18. ST_MemUnion

This topic describes the ST_MemUnion function. This function returns a geometry object that represents the union of input geometry objects. This function is the same as the ST_Union function.

Syntax

```
geometry  ST_MemUnion(geometry set  geomfield);
```

Parameters

Parameter	Description
geomfield	The field of the geometry object that you want to specify in the geometry set.

Description

This function is the same as the ST_Union function. However, this function consumes less memory and requires more time to return results than the ST_Union function.

Examples

Results returned by using the default parameter settings:

```
select ST_asText(ST_MemUnion(g))
         from (select unnest(array['POLYGON((0 0,1 0,1 2,0 2,0 0))'::geometry,'POLYGON((1 0
,3 0,3 1,1 1,1 0))'::geometry]) as g) as t;
                 st_astext
--------------------------------------------
 POLYGON((1 0,0 0,0 2,1 2,1 1,3 1,3 0,1 0))
(1 row)
```

4.6.19. ST_MinimumBoundingCircle

This topic describes the ST_MinimumBoundingCircle function. This function returns the smallest circle polygon object that can fully cover the geometry object. By default, this function uses 48 segments to approximate a quarter circle.

Syntax

```
geometry  ST_MinimumBoundingCircle(geometry  geomA , integer  numSegsPerQtCirc);
```

Parameters

Parameter	Description
geomA	The geometry object that you want to specify.
numSegsPerQtCirc	The number of segments that are used to approximate a quarter circle. Default value: 48.

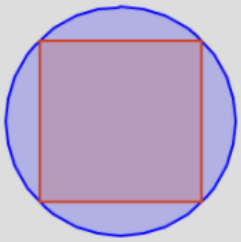
Description

- A larger value of the numSegsPerQtCirc parameter indicates a more accurate result, but the performance may deteriorate.
- In most cases, you can use this function to process MULTI and GeometryCollection objects. This function is not an aggregate function. However, you can use this function with the ST_Collect function to obtain the smallest bounding circle of a set of geometry objects. Example: ST_MinimumBoundingCircle(ST_Collect(somepointfield)).

Examples

Comparison between the smallest bounding circle object and the input geometry object:

```
select ST_CurveToLine(ST_MinimumBoundingCircle(g)),g from (select 'POLYGON((0 0,1 0,1 1,0 1
,0 0))'::geometry as g) as t
```



4.6.20. ST_MinimumBoundingRadius

This topic describes the ST_MinimumBoundingRadius function. This function returns the center point and radius of the smallest circle that can fully cover the input geometry object.

Syntax

```
(geometry , double precision)  ST_MinimumBoundingRadius(geometry  geom);
```

Parameters

Parameter	Description
geom	The geometry object that you want to specify.

Description

You can use this function with the ST_Collect function to obtain the smallest bounding circle of a set of geometry objects.

Examples

Results returned by using the default parameter settings:

```
select ST_AsText(center), radius from ST_MinimumBoundingRadius('POLYGON((0 1,-1 0,0 -1,1 0,
0 1))'::geometry);
 st_astext  | radius
------------+--------
 POINT(0 0) |      1
(1 row)
```

4.6.21. ST_OrientedEnvelope

This topic describes the ST_OrientedEnvelope function. This function returns the smallest rotated rectangle that encloses the input geometry object.

Syntax

```
geometry  ST_OrientedEnvelope(geometry  geom);
```

Parameters

Parameter	Description
geom	The geometry object that you want to specify.

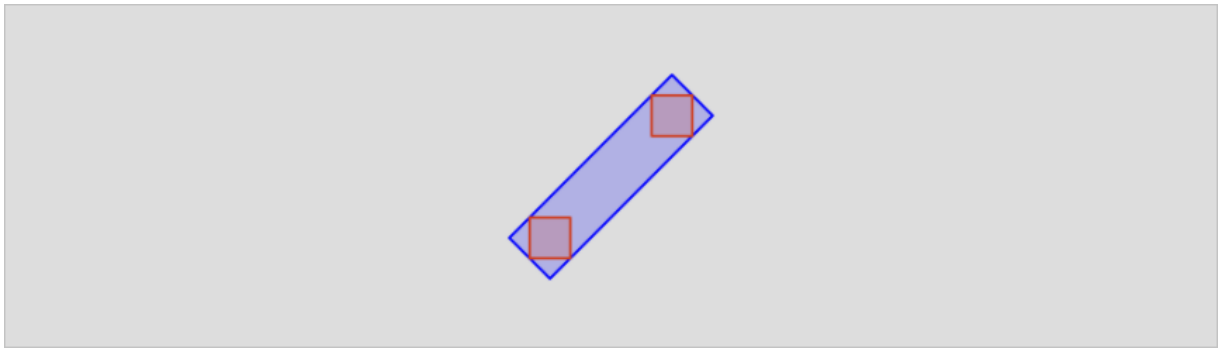
Description

- The number of smallest rotated rectangles of the input geometry object may be more than one.
- In the case of degenerate input, this function may return a point object or LineString object.

Examples

Comparison between the input geometry object and one of the smallest rotated rectangles:

```
select ST_OrientedEnvelope(g),g from (select 'POLYGON((0 0,1 0,1 1,0 1,0 0),(3 3,4 3,4 4,3
4,3 3))'::geometry as g) as t;
```



4.6.22. ST_Polygonize

This topic describes the ST_Polygonize function. This function is an aggregate function. This function returns a GeometryCollection object that contains possible polygon objects formed by the constituent lines of input geometry objects.

Syntax

```
geometry  ST_Polygonize(geometry set  geomField);
geometry  ST_Polygonize(geometry[]  geomArray);
```

Parameters

Parameter	Description
geomfield	The fields of the geometry objects that you want to specify in the dataset.
geomArray	The array of geometry objects that you want to specify.

Description

- In most cases, third-party tools do not support GeometryCollection objects. You can use this function with the ST_Dump function to expand a collection of polygon objects into separate polygon objects.
- Before you use this function, the constituent lines of input geometry objects must be correctly noded.

Examples

Results returned by using the default parameter settings:

```
select ST_AsText(ST_Polygonize(array['LINESTRING(0 0,0 1)'::geometry,
                                     'LINESTRING(0 1,1 1)'::geometry,
                                     'LINESTRING(1 1,1 0)'::geometry,
                                     'LINESTRING(0 0,1 0)'::geometry]));
                     st_astext
--------------------------------------------------
 GEOMETRYCOLLECTION(POLYGON((0 0,0 1,1 1,1 0,0 0)))
(1 row)
```

4.6.23. ST_OffsetCurve

This topic describes the ST_OffsetCurve function. This function returns a geometry object representing all offset lines at the specified distance and margin from the input geometry object.

Syntax

```
geometry  ST_OffsetCurve(geometry  line , float  signedDistance , text  styleParameters);
```

Parameters

Parameter	Description
line	The line object that you want to specify.
signedDistance	The distance that you want to specify.
styleParameters	The style of the offset lines. The default value is an empty string.

Description

- The distance between points on the returned geometry object and the input geometry object does not exceed the specified distance.
- If the value of the signedDistance parameter is a positive number, the offset lines are on the left side of the input geometry object and in the same direction as the input geometry object. If the value of the signedDistance parameter is a negative number, the offset lines are on the right side of the input geometry object and in the opposite direction of the input geometry object.
- This function ignores z coordinates. If you specify a 3D geometry object, this function returns a 2D geometry object.

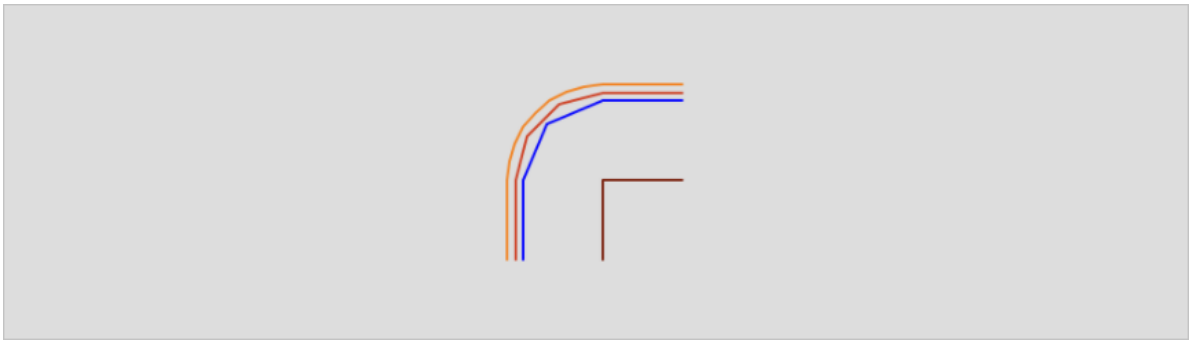
- This function supports GeometryCollection objects and MultiLineString objects.
- The following table describes the fields of the styleParameters parameter.

Field	Description	Type	Default value	Description
quad_segs	The number of segments that are used to approximate a quarter circle.	integer	8	A greater value indicates a smoother quarter circle.
join	The join style of offset lines.	string	round	Valid values: ○ round ○ mitre ○ bevel
mitre_limit	The limit of the miter ratio.	float	--	This parameter is valid only when you set the join parameter to mitre.

Examples

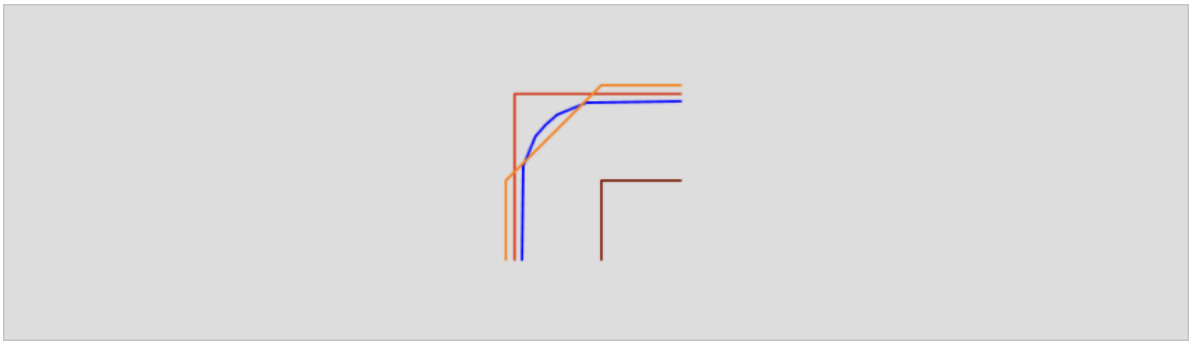
- Results returned with different values of the quad_segs field:

```
select st_curvetoline(ST_OffsetCurve(g,1,'quad_segs=2')),
             st_curvetoline(ST_OffsetCurve(g,1.1,'quad_segs=3')),
             st_curvetoline(ST_OffsetCurve(g,1.2))g
       from (select 'LINESTRING(0 0,0 1,1 1)'::geometry as g) as t;
```



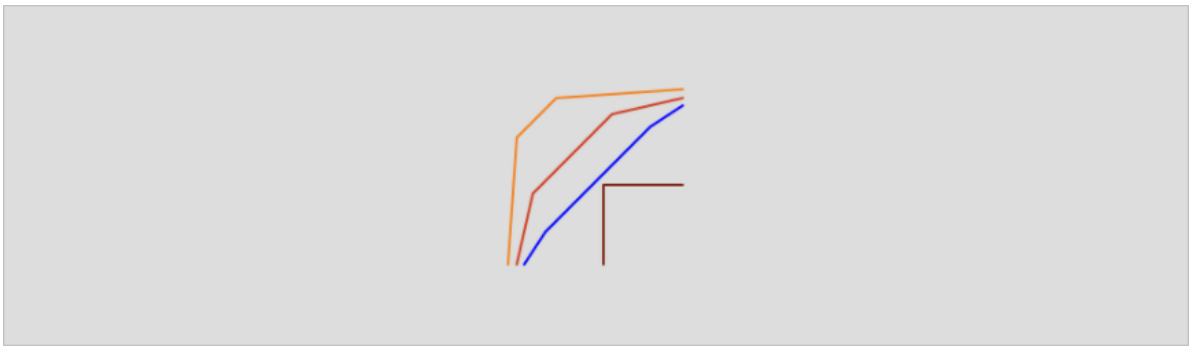
- Results returned with different values of the join field:

```
select st_curvetoline(ST_OffsetCurve(g,1,'join=round')),
             st_curvetoline(ST_OffsetCurve(g,1.1,'join=mitre')),
             st_curvetoline(ST_OffsetCurve(g,1.2,'join=bevel')),g
       from (select 'LINESTRING(0 0,0 1,1 1)'::geometry as g) as t;
```



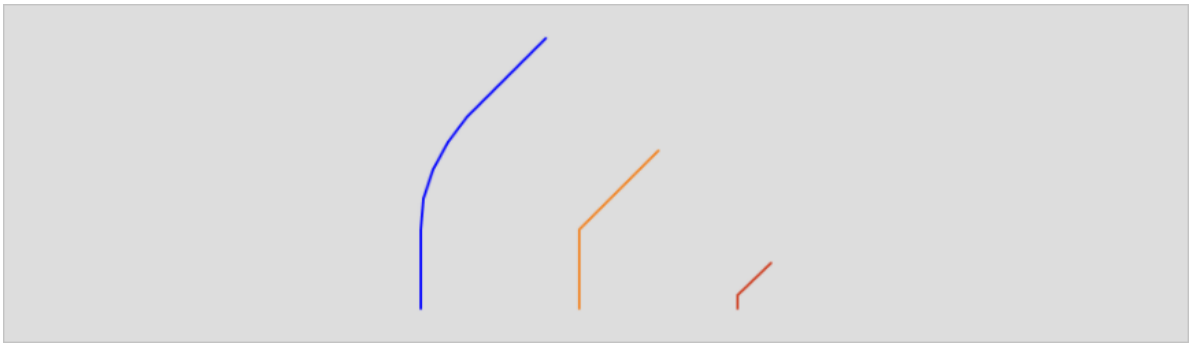
- Results returned with different values of the mitre_limit field:

```
select st_curvetoline(ST_OffsetCurve(g,1,'join=mitre mitre_limit=0.1')),
             st_curvetoline(ST_OffsetCurve(g,1.1,'join=mitre mitre_limit=0.5')),
             st_curvetoline(ST_OffsetCurve(g,1.2,'join=mitre mitre_limit=1')),g
       from (select 'LINESTRING(0 0,0 1,1 1)'::geometry as g) as t;
```



- Results returned with a positive value and a negative value of the signedDistance parameter:

```
select st_curvetoline(ST_OffsetCurve(g,2)),
         st_curvetoline(ST_OffsetCurve(g,-2)),g
       from (select 'LINESTRING(0 0,0 1,1 2)'::geometry as g) as t;
```



4.6.24. ST_RemoveRepeatedPoints

This topic describes the ST_RemoveRepeatedPoints function. This function deletes the duplicated points of the input geometry object and returns a new geometry object.

Syntax

```
geometry  ST_RemoveRepeatedPoints(geometry  geom , float8  tolerance);
```

Parameters

Parameter	Description
geom	The geometry object that you want to specify.
tolerance	The tolerance that you want to specify.

Description

Results returned by using the default parameter settings:

```
select ST_AsText(ST_RemoveRepeatedPoints('POLYGON((0 0,1 0,1 0,1 1,0 0))'::geometry));
         st_astext
---------------------------
 POLYGON((0 0,1 0,1 1,0 0))
(1 row)
```

4.6.25. ST_SharedPaths

This topic describes the ST_SharedPaths function. This function returns a collection of paths that are shared by two input geometry objects.

Syntax

```
geometry  ST_SharedPaths(geometry  lineal1 , geometry  lineal2);
```

Parameters

Parameter	Description
lineal1	The shared path 1.
lineal2	The shared path 2.

Description

- The objects in the same direction are in the first element of the returned collection, and the objects in the opposite direction are in the second element of the returned collection.
- The direction of paths in the returned collection is based on the direction of the first input geometry object.

Examples

Results returned by using the default parameter settings:

```
select ST_AsText(ST_GeometryN(g,1)),ST_AsText(ST_GeometryN(g,2)) from (select ST_SharedPath
s('LINESTRING(-1 0,0 0,0 1)'::geometry,'LINESTRING(0 0,0 1,1 1)'::geometry) as g) as t;
         st_astext          |       st_astext
----------------------------+-----------------------
 MULTILINESTRING((0 0,0 1)) | MULTILINESTRING EMPTY
(1 row)
```

4.6.26. ST_ShiftLongitude

This topic describes the ST_ShiftLongitude function. This function converts the y coordinates of the input geometry object from -180 to 0 degrees to 180 to 360 degrees, or from 180 to 360 degrees to -180 to 0 degrees.

Syntax

```
geometry  ST_ShiftLongitude(geometry  geomA);
```

Parameters

Parameter	Description
geomA	The geometry object that you want to specify.

Description

- This function processes only objects represented by latitude and longitude coordinate systems, such as the World Geodetic System 1984 (WGS 84) coordinate system.
- This function supports 3D objects and does not delete z coordinates.
- This function supports polyhedral surfaces, triangles, and triangulated irregular network (TIN) surfaces.

Examples

```
SELECT ST_AsText(ST_ShiftLongitude('POINT(181 30)'));
   st_astext
----------------
 POINT(-179 30)
(1 row)
```

4.6.27. ST_Simplify

This topic describes the ST_Simplify function. This function returns a geometry object representing a simplified version of the input geometry object. This function uses the Douglas-Peucker algorithm.

Syntax

```
geometry  ST_Simplify(geometry  geomA , float  tolerance , boolean  preserveCollapsed);
```

Parameters

Parameter	Description
geomA	The geometry object that you want to specify.
tolerance	The tolerance that you want to specify.
preserveCollapsed	The flag that specifies whether to keep small geometry objects in the returned geometry object.

Description

- This function supports simplification operations only on MultiLine objects, MultiPolygon objects, and MultiPoint objects. However, you can use this function to process any geometry objects.
- The simplification operation is performed on the input geometry objects one by one. Therefore, you can use this function to process GeometryCollection objects.
- If the scale of the input geometry object is much smaller than the value of the tolerance parameter and the preserveCollapsed parameter is specified, the input geometry object does not disappear.

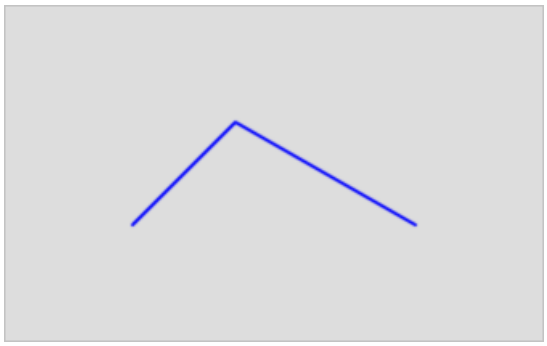
  The preserveCollapsed parameter is useful to render engines to prevent a large number of small geometry objects from disappearing from maps and leaving abnormal gaps.
- The returned geometry object may lose its simplicity.
- This function may change the topology of the input geometry object and return an invalid geometry object.

  You can use the ST_SimplifyPreserveTopology function to preserve the topology of the input geometry object.

Examples

Results returned with different values of the tolerance parameter:

```
select g,ST_Simplify(g,0.25),
             ST_Simplify(g,0.5)
       from (select 'LINESTRING(0 0,2 2,3 1,3.5 1.5,5 0,5.25 0.25,5.5 0)'::geometry as g) a
s t;
```



4.6.28. ST_SimplifyPreserveTopology

This topic describes the ST_SimplifyPreserveTopology function. This function returns a geometry object representing a simplified version of the input geometry object. This function uses the Douglas-Peucker algorithm. This function prevents the return of an invalid geometry object.

Syntax

```
geometry  ST_SimplifyPreserveTopology(geometry  geomA , float  tolerance);
```

Parameters

Parameter	Description
geomA	The geometry object that you want to specify.
tolerance	The tolerance that you want to specify.

Description

- This function supports simplification operations only on MultiLine objects, MultiPolygon objects, and MultiPoint objects. However, you can use this function to process any geometry objects.
- The simplification operation is performed on the input geometry objects one by one. Therefore, you can use this function to process GeometryCollection objects.
- This function prevents the return of a derived and invalid geometry object, especially polygon objects.

Examples

Comparison between the results of the ST_SimplifyPreserveTopology function and the ST_Simplify function:

```
SELECT ST_ASText(ST_SimplifyPreserveTopology(g,1)) as PreserveTopology ,ST_ASText(ST_Simpli
fy(g,1)) as Normal from (select 'POLYGON((0 0,1 0,0 1,1 1,0 0))'::geometry as g) as t;
      preservetopology       | normal
----------------------------+--------
 POLYGON((0 0,1 0,1 1,0 0)) |
(1 row)
```

4.6.29. ST_SimplifyVW

This topic describes the ST_SimplifyVW function. This function returns a geometry object representing a simplified version of the input geometry object. This function uses the Visvalingam-Whyatt algorithm.

Syntax

```
geometry  ST_SimplifyVW(geometry  geomA , float  tolerance);
```

Parameters

Parameter	Description
geomA	The geometry object that you want to specify.
tolerance	The tolerance that you want to specify.

Description

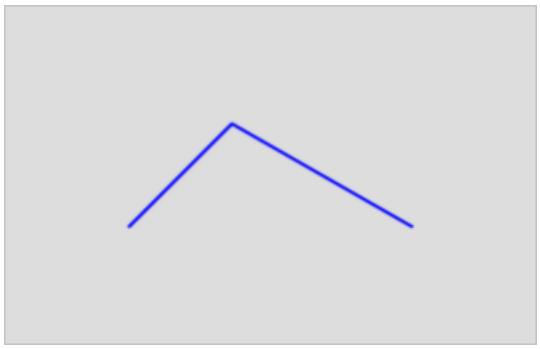
- This function supports simplification operations only on MultiLine objects, MultiPolygon objects, and MultiPoint objects. However, you can use this function to process any geometry objects.
- The simplification operation is performed on the input geometry objects one by one. Therefore, you can use this function to process GeometryCollection objects.
- The returned geometry object may lose its simplicity.
- This function may change the topology of the input geometry object and return an invalid geometry object.

  You can use the ST_SimplifyPreserveTopology function to preserve the topology of the input geometry object.
- This function supports 3D objects, and the 3D dimension of the input geometry object affects the returned geometry object.

Examples

Comparison between the results of the ST_Simplify function and the ST_SimplifyVW function:

```
select g,ST_Simplify(g,0.5),
             ST_SimplifyVW(g,0.5)
       from (select 'LINESTRING(0 0,2 2,3 1,3.5 1.5,5 0,5.25 0.25,5.5 0)'::geometry as g) a
s t;
```



4.6.30. ST_ChaikinSmoothing

This topic describes the ST_ChaikinSmoothing function. This function returns a geometry object representing a smoother version of the input geometry object. This function uses the Chaikin algorithm.

Syntax

```
geometry  ST_ChaikinSmoothing(geometry  geom , integer  nIterations , boolean  preserveEndP
oints);
```

Parameters

Parameter	Description
geom	The geometry object that you want to specify.
nIterations	The number of iterations that you want to specify. Maximum value: 5. Default value: 1.
preserveEndPoints	The flag that specifies whether to keep the endpoints of the input geometry object. Default value: false. This parameter takes effects only when the input geometry object is a polygon object.

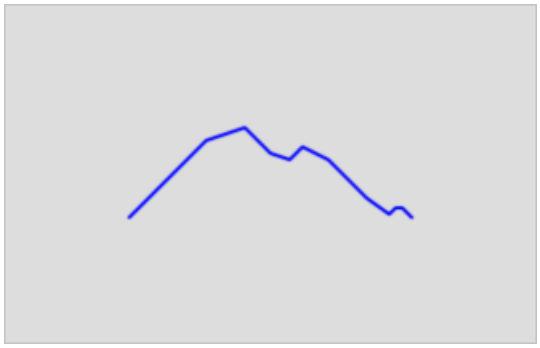
Description

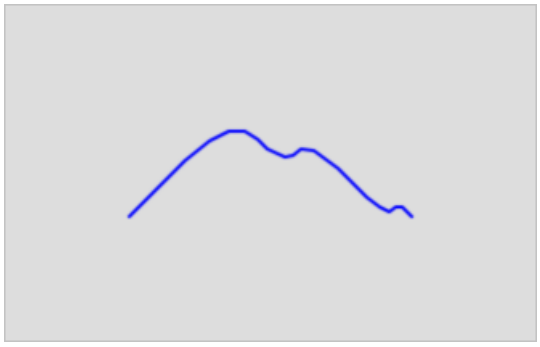
- For each iteration operation, this function doubles the number of vertexes of the input geometry object.
  - The function places a new vertex at one fourth of the lines before and after each point of the input geometry object and removes the original point.
  - The new points are interpolated values for all inherent dimensions of the input geometry object, including z and m dimensions.
- The returned geometry object contains more points than the input geometry object. If you want to reduce the number of points of the returned geometry object, you use the ST_Simplify function or the ST_SimplifyVW function.

Examples

Results returned with different values of the nIterations parameter:

```
select g,ST_ChaikinSmoothing(g,1),
             ST_ChaikinSmoothing(g,5)
       from (select 'LINESTRING(0 0,2 2,3 1,3.5 1.5,5 0,5.25 0.25,5.5 0)'::geometry as g) a
s t;
```





4.6.31. ST_FilterByM

This topic describes the ST_FilterByM function. This function filters out the vertexes of the input geometry object based on the m coordinates of the input geometry object and returns a new geometry object.

Syntax

```
geometry ST_FilterByM(geometry geom , double precision min , double precision max , boo
lean returnM);
```

Parameters

Parameter	Description
geom	The geometry object that you want to specify.
min	The minimum value of the m coordinate for the input geometry object. Default value: null.
max	The maximum value of the m coordinate for the input geometry object. Default value: null.
returnM	Specifies whether to return the m coordinates of the new geometry object. Default value: false.

Description

- This function returns a new geometry object for which the m coordinate of a vertex is greater than or equal to the value of the min parameter and less than or equal to the value of the max parameter.

If this function ignores the max parameter, only the min parameter takes effect.

- If a small number of vertexes are generated after the filtering, this function returns an empty geometry object.
- If you specify a GeometryCollection object, this function ignores the objects that do not contain enough points.
- If the input geometry object does not contain enough points, the ST_SimplifyVW function returns a geometry object that contains enough points while the ST_FilterByM function returns an empty geometry object.
- This function may return an invalid geometry object.
- The returned geometry object includes all dimensions of the input geometry object, including z and m coordinates.

Examples

- Results returned by using the default parameter settings:

```
SELECT ST_AsText(ST_FilterByM('LINESTRINGM(0 0 0,1 1 1,2 2 2,3 3 3)'::geometry,2,4,true))
;
         st_astext
----------------------------
 LINESTRING M (2 2 2,3 3 3)
(1 row)
```

- Results returned when the function cannot return a geometry object:

```
SELECT ST_AsText(ST_FilterByM('LINESTRINGM(0 0 0,1 1 1,2 2 2,3 3 3)'::geometry,3,4,true))
;
     st_astext
--------------------
 LINESTRING M EMPTY
(1 row)
```

4.6.32. ST_SetEffectiveArea

This topic describes the ST_SetEffectiveArea function. This function sets the effective area for each vertex of the input geometry object. This function uses the Visvalingam-Whyatt algorithm. The effective area of each vertex is stored as the M value.

Syntax

```
geometry  ST_SetEffectiveArea(geometry  geomA , float  threshold , integer  setArea);
```

Parameters

Parameter	Description
geomA	The geometry object that you want to specify.

Parameter	Description
threshold	The tolerance that you want to specify. Default value: 0. If you specify this parameter, this function returns only the vertex whose effective area is greater than or equal to the value of this parameter.
setArea	Default value: 1.

Description

- If you specify the threshold parameter, this function can be used for server-side simplification.
- If you do not specify the threshold parameter, this function can be used to simplify geometry objects.
- This function supports simplification operations only on MultiLine objects, MultiPolygon objects, and MultiPoint objects. However, you can use this function to process any geometry objects.
- The simplification operation is performed on the input geometry objects one by one. Therefore, you can use this function to process GeometryCollection objects.
- The returned geometry object may lose its topology or be invalid.
- The returned geometry object may lose its previous information stored in the M value.
- The returned geometry object may lose its simplicity.
- This function supports 3D objects, and the 3D dimension of the input geometry object affects the returned geometry object.

Examples

Results returned by using the default parameter settings:

```
SELECT ST_AsText(ST_SetEffectiveArea('LINESTRING(0 0,1 1,2 2 )'::geometry));
                      st_astext
-----------------------------------------------------
 LINESTRING M (0 0 3.40282e+38,1 1 0,2 2 3.40282e+38)
(1 row)
```

4.6.33. ST_Split

This topic describes the ST_Split function. This function returns a GeometryCollection object by splitting the input geometry object.

Syntax

```
geometry  ST_Split(geometry  input , geometry  blade);
```

Parameters

Parameter	Description
input	The geometry object that you want to specify.

Parameter	Description
blade	The geometry object that you use to split the input geometry object.

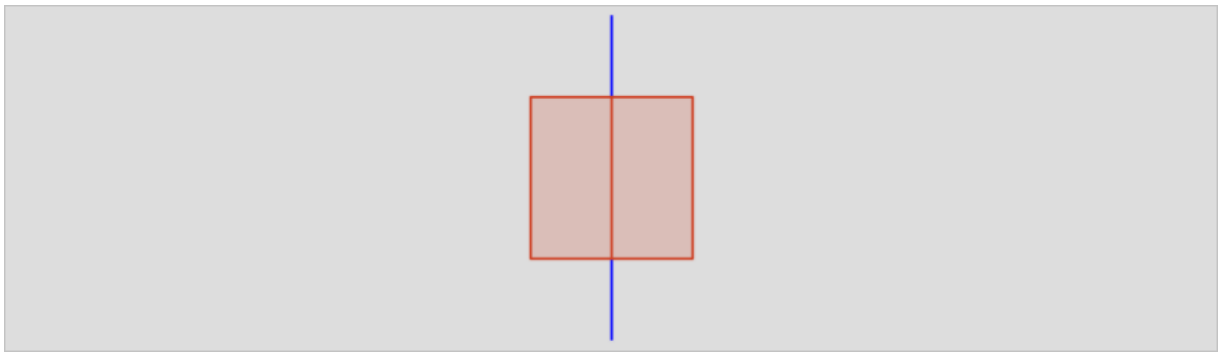
Description

- This function can split a line object by using a MultiPoint object, MultiLine object, or MultiPolygon object. This function can also split a MultiPolygon object by using a line object.
- This function always returns a GeometryCollection object.
- Theoretically, if you use the ST_Union function to process the result returned by the ST_Split function, the ST_Union function returns the input geometry object of the ST_Split function.
- If you set the blade parameter to a MultiPolygon object, this function splits the input geometry object by using the boundaries of the MultiPolygon object.

Examples

Execute the following statements to split a polygon object by using a LineString object:

```
SELECT blade,
       ST_Split(input,blade)
       from (select ST_MakeEnvelope(0,0,2,2) as input,'LINESTRING(1 -1,1 3)'::geometry as b
lade) as t;
```



4.6.34. ST_SymDifference

This topic describes the ST_SymDifference function. This function returns a geometry object representing the part that Geometry Object A does not intersect with Geometry Object B.

Syntax

```
geometry  ST_SymDifference(geometry  geomA , geometry  geomB);
```

Parameters

Parameter	Description
geomA	Geometry Object A that you want to specify.

Parameter	Description
geomB	Geometry Object B that you want to specify.

Description

- This function does not support GeometryCollection objects.
- This function supports 3D objects and does not delete z coordinates. This function processes only the x and y coordinates of the input geometry objects. This function returns the output geometry object with the original z coordinates.

Examples

Results returned by using the default parameter settings:

```
SELECT ST_AsText(ST_SymDifference('LINESTRING(0 0,2 0)'::geometry,'LINESTRING(1 0,3 0)'::ge
ometry));
              st_astext
-------------------------------------
 MULTILINESTRING((0 0,1 0),(2 0,3 0))
(1 row)
```

4.6.35. ST_Subdivide

This topic describes the ST_Subdivide function. This function divides the input geometry object into parts until the number of vertexes in each part is less than the value of the maxVertices parameter.

Syntax

```
setof  geometry ST_Subdivide(geometry  geom , integer  maxVertices);
```

Parameters

Parameter	Description
geom	The geometry object that you want to specify.
maxVertices	The maximum number of vertexes. Default value: 256. Minimum value: 5.

Description

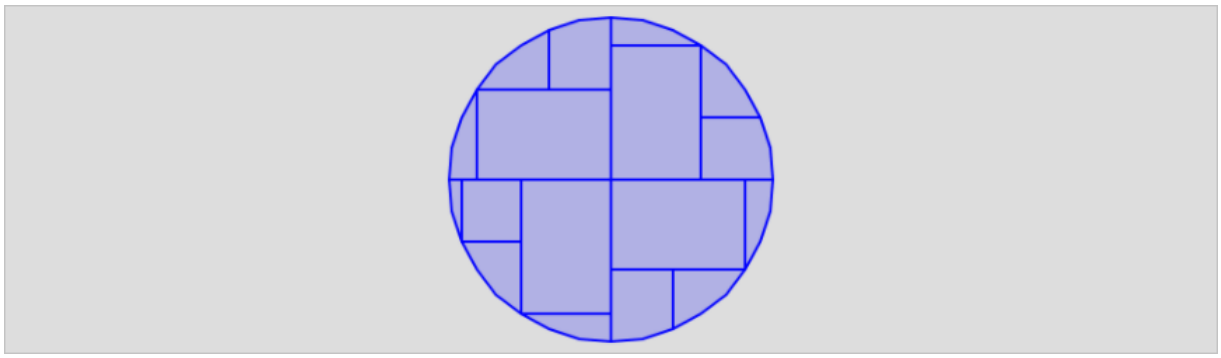
After the division, operations of obtaining points within a polygon object and spatial operations run fast for the indexed divided parts for the following reasons:

- In most cases, the bounding boxes of all parts are smaller than the box of the input geometry object, so missed points are obtained faster.
- The recheck operations are performed on fewer points, so hit points are obtained faster.

Examples

Execute the following statement to divide a circle object:

```
SELECT ST_Subdivide(st_buffer('POINT(0 0)'::geometry,1),6);
```



4.6.36. ST_SwapOrdinates

This topic describes the ST_SwapOrdinates function. This function swaps the specified coordinates of a geometry object.

Syntax

```
geometry  ST_SwapOrdinates(geometry  geom , cstring  ords);
```

Parameters

Parameter	Description
geom	The geometry object that you want to specify.
ords	The coordinates that you want to swap. The value of this parameter is a 2-character string. Valid names: x, y, z, and m.

Description

- This function supports circular strings, curves, polyhedral surfaces, triangles, and triangulated irregular network (TIN) surfaces.
- The ST_SwapOrdinates function supports 3D objects and does not delete z coordinates.
- The ST_SwapOrdinates function supports m coordinates.

Examples

Process a geometry object by using the default parameter settings.

```
SELECT ST_AsText(ST_SwapOrdinates('POINT(1 0)'::geometry,'xy'));
 st_astext
------------
 POINT(0 1)
(1 row)
```

4.6.37. ST_Union

This topic describes the ST_Union function. This function constructs a geometry object that represents the union of multiple specified geometry objects.

Syntax

```
geometry  ST_Union(geometry set  g1Field);
geometry  ST_Union(geometry  g1 , geometry  g2);
geometry  ST_Union(geometry[]  g1Array);
```

Parameters

Parameter	Description
g1field	The fields of the geometry objects that you want to specify in the dataset.
g1	The first geometry object that you want to specify.
g2	The second geometry object that you want to specify.
g1Array	An array that consists of the geometry objects you want to specify.

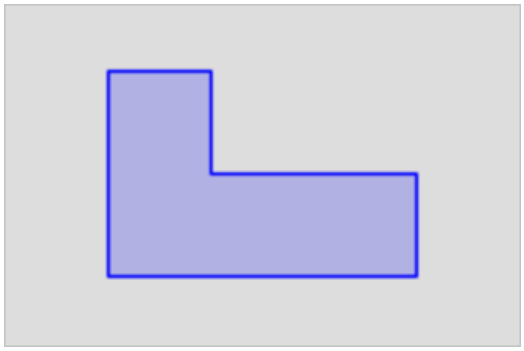
Description

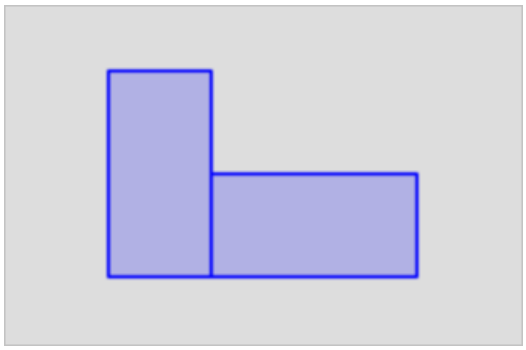
- The output type of this function is MULTI or GeometryCollection. The ST_Union function has two versions:
  - Version 1: The input of the ST_Union function is two geometry objects. The output can be a MULTI, NON-MULTI, or GeometryCollection object. If one of the two geometry objects is NULL, the ST_Union function returns NULL.
  - Version 2: The ST_Union function works as an aggregate function. The input is a collection of geometry objects. The output can be a MULTI or NON-MULTI object.
- The ST_Collect function and the ST_Union function are interchangeable. The ST_Union function tries to dissolve the boundaries of the specified geometry objects to check whether the constructed MultiPolygon object has intersecting parts. Therefore, in most cases, the ST_Union runs at a lower speed than the ST_Collect function.

Examples

The following example shows the differences between the ST_Union function and the ST_Collect function:

```
SELECT ST_Union(g1,g2),ST_Collect(g1,g2)
             from (select 'POLYGON((0 0,1 0,1 2,0 2,0 0))'::geometry as g1,'POLYGON((1 0,3
0,3 1,1 1,1 0))'::geometry as g2) as t;
```





4.6.38. ST_VoronoiLines

This topic describes the ST_VoronoiLines function. This function computes a two-dimensional Voronoi diagram by using the vertices of a geometry object and returns the boundaries between cells in the Voronoi diagram as a MultiLineString object.

Syntax

```
geometry ST_VoronoiLines(geometry g1, float tolerance, geometry extend_to);
```

Parameters

Parameter	Description
g1	The geometry object that you want to specify.
tolerance	The tolerance. Default value: 0.0. If the distance between the two vertices of the geometry object is smaller than the value of the tolerance parameter, the two vertices are coincident. You can specify a value that is not zero for the tolerance parameter to increase the robustness of the algorithm.
extend_to	The extended envelope. Default value: null. If you set this parameter to null, the default envelope is the bounding box that is extended by approximately 50% in each direction of the geometry object. If you specify this parameter, the Voronoi diagram is extended to cover the extended envelope, unless the extended envelope is smaller than the default envelope.

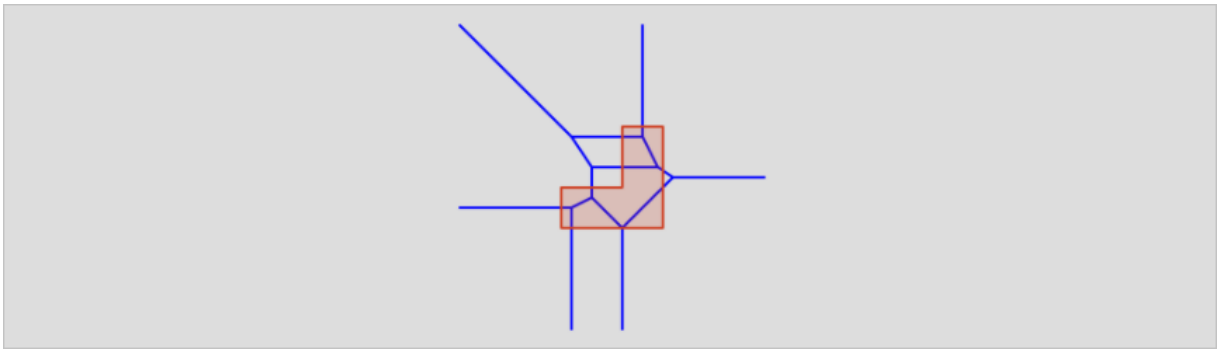
Description

- If the specified geometry object is NULL, the ST_VoronoiLines function returns NULL.
- If the specified geometry object contains only one vertex, the ST_VoronoiLines function returns an empty GeometryCollection object.
- If the area of the extended envelope that is specified by the extend_to parameter is zero, the ST_VoronoiLines function returns an empty GeometryCollection object.

Examples

Process a geometry object by using the default parameter settings.

```
SELECT ST_VoronoiLines(g),g
             from (select ST_Buffer('LINESTRING(0 0,3 0,3 3)'::geometry,1,'join=mitre endca
p=square') as g) as t;
```



4.6.39. ST_VoronoiPolygons

This topic describes the ST_VoronoiPolygons function. This function computes a two-dimensional Voronoi diagram by using the vertices of a geometry object.

Syntax

```
geometry  ST_VoronoiPolygons(geometry  g1, float  tolerance, geometry  extend_to);
```

Parameters

Parameter	Description
g1	The geometry object that you want to specify.
tolerance	The tolerance. Default value: 0.0. If the distance between the two vertices is smaller than the value of the tolerance parameter, the two vertices are coincident. You can specify a value that is not zero for the tolerance parameter to increase the robustness of the algorithm.

Parameter	Description
extend_to	The extended envelope. Default value: null. If you set this parameter to null, the default envelope is the bounding box that is extended by approximately 50% in each direction of the specified geometry object. If you specify this parameter, the Voronoi diagram is extended to cover the extended envelope, unless the extended envelope is smaller than the default envelope.

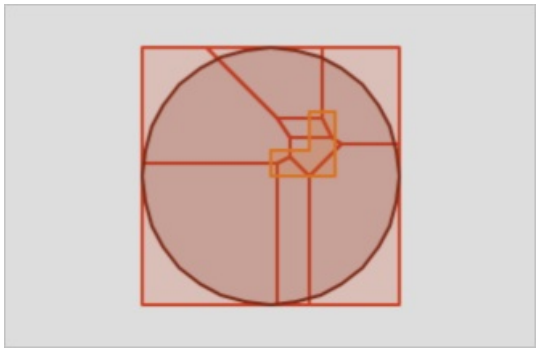
Description

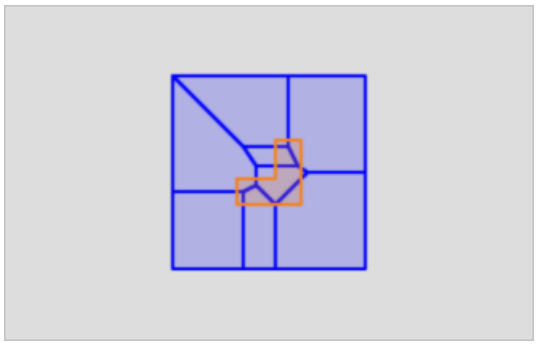
- If the return value is a GeometryCollection object of Polygons, the extended envelope is larger than the extent of the vertices of the specified geometry object.
- If the specified geometry object is NULL, the ST_VoronoiPolygons function returns NULL.
- If the specified geometry object contains only one vertex, the ST_VoronoiPolygons function returns an empty GeometryCollection object. If the specified geometry object contains only one vertex, the ST_VoronoiPolygons function returns an empty GeometryCollection object.
- If the area of the extended envelope that is specified by the extend_to parameter is zero, the ST_VoronoiPolygons function returns an empty GeometryCollection object.

Examples

The following example shows the differences between the return results that are obtained when the extend_to parameter is set to different values:

```
SELECT ST_VoronoiPolygons(g,0,e),g,e
             from (select ST_Buffer('LINESTRING(0 0,3 0,3 3)'::geometry,1,'join=mitre endca
p=square') as g,
                    ST_Buffer('POINT(-1 -1)'::geometry,10) as e) as t;
```





4.7. Aggregate functions

4.7.1. ST_ClusterDBSCAN

This topic describes the ST_ClusterDBSCAN function. This function returns a cluster number for each specified geometry object by using the 2D density-based spatial clustering of applications with noise (DBSCAN) algorithm.

Syntax

```
integer  ST_ClusterDBSCAN(geometry winset  geom , float8  eps , integer  minpoints);
```

Parameters

Parameter	Description
geom	The geometry objects that you want to specify.
eps	The minimum distance that you want to specify.
minpoints	The minimum number of geometry objects that must be located within the minimum distance specified by the eps parameter. This parameter is used to identify whether a geometry object is the core geometry object of a cluster.

Description

- Unlike the ST_ClusterKMeans function, the ST_ClusterDBSCAN function does not require you to specify the number of clusters. The ST_ClusterDBSCAN function uses the distance and density that you specify to construct each cluster.
- If a geometry object that you specify meets one of the following conditions in a cluster, the geometry object is added to the cluster:
  - At least the minimum number of geometry objects are located within the minimum distance to the geometry object, which, in this case, is considered to be the **core** geometry object in the cluster.
  - The core geometry object is located within the minimum distance to the geometry object, which, in this case, is considered to be a **boundary** geometry object in the cluster.

    A boundary geometry object can be located within the minimum distance to core geometry objects in multiple clusters. If a boundary geometry object is located within the minimum distance to core geometry objects in multiple clusters, the boundary geometry object is assigned at random to one of the available clusters, and the ST_ClusterDBSCAN function may generate a cluster in which the number of geometry objects is less than the specified minimum number.
- If a geometry object that you specify does not meet any of the preceding conditions in all clusters, the ST_ClusterDBSCAN function assigns a cluster number NULL to the geometry object.
- The ST_ClusterDBSCAN function is a window function.

Examples

```
SELECT ST_ClusterDBSCAN(geom,2,1) over() ,st_AsText(geom)
    from (select unnest(ARRAY['POINT (0 0)'::geometry,
                              'POINT(1 1)'::geometry,
                              'POINT (-1 -1)'::geometry,
                              'POINT (-3 -3)'::geometry]) as geom) as test;
 st_clusterdbscan |  st_astext
------------------+--------------
                0 | POINT(0 0)
                0 | POINT(1 1)
                0 | POINT(-1 -1)
                1 | POINT(-3 -3)
(4 rows)
```

4.7.2. ST_ClusterIntersecting

This topic describes the ST_ClusterIntersecting function. This function returns an array that consists of GeometryCollection objects, each of which represents an interconnected set of geometry objects.

Syntax

```
geometry[]  ST_ClusterIntersecting(geometry set  g);
```

Parameters

Parameter	Description
g	The geometry dataset that you want to specify.

Description

This function returns an array that consists of GeometryCollection objects. Each GeometryCollection object represents an interconnected set of geometry objects.

Examples

```
SELECT ST_AsText(unnest(ST_ClusterIntersecting(geom)))
    from (select ARRAY['LINESTRING (0 0,0 1)'::geometry,
                       'LINESTRING (0 1,3 3)'::geometry,
                       'POINT (-1 -1)'::geometry] as geom) as test;
                            st_astext
-------------------------------------------------------------
 GEOMETRYCOLLECTION(LINESTRING(0 0,0 1),LINESTRING(0 1,3 3))
 GEOMETRYCOLLECTION(POINT(-1 -1))
(2 rows)
```

4.7.3. ST_ClusterKMeans

This topic describes the ST_ClusterKMeans function. This function returns a cluster number for each specified geometry object by using the 2D-based K-means algorithm.

Syntax

```
integer  ST_ClusterKMeans(geometry winset  geom , integer  numberOfClusters);
```

Parameters

Parameter	Description
geom	The geometry objects that you want to specify.
numberOfClusters	The number of clusters that you want to specify.

Description

- The distance that is used for clustering is the distance between the centroids of the geometry objects that you specify.
- The ST_ClusterKMeans function is a window function.

Examples

```
SELECT ST_ClusterKMeans(geom,2) over() ,st_AsText(geom)
    from (select unnest(ARRAY['POINT (0 0)'::geometry,
                              'POINT(1 1)'::geometry,
                              'POINT (-1 -1)'::geometry,
                              'POINT (-2 -2)'::geometry]) as geom) as test;
 st_clusterkmeans |  st_astext
------------------+--------------
                 0 | POINT(0 0)
                 0 | POINT(1 1)
                 1 | POINT(-1 -1)
                 1 | POINT(-2 -2)
(4 rows)
```

4.7.4. ST_ClusterWithin

This topic describes the ST_ClusterWithin function. This function returns an array of GeometryCollection objects, each of which represents a set of geometry objects that are separated by no more than the specified distance.

Syntax

```
geometry[]  ST_ClusterWithin(geometry set  g , float8  distance);
```

Parameters

Parameter	Description
g	The geometry objects that you want to specify.
distance	The distance that you want to use for clustering.

Description

The distance that you specify is a cartesian distance that is expressed by using spatial reference identifiers (SRIDs).

Examples

- Obtain an array of GeometryCollection objects with the default parameter settings retained.

```
SELECT ST_AsText(unnest(ST_ClusterWithin(geom,1)))
 from (select ARRAY['LINESTRING (0 0,0 1)'::geometry,
                    'LINESTRING (2 3,3 3)'::geometry,
                    'LINESTRING (0 1,2 3)'::geometry,
                    'POINT (-1 -1)'::geometry] as geom) as test;
                          st_astext
----------------------------------------------------------------
 GEOMETRYCOLLECTION(LINESTRING(0 0,0 1),LINESTRING(2 3,3 3),LINESTRING(0 1,2 3))
 GEOMETRYCOLLECTION(POINT(-1 -1))
(2 rows)
```

- Obtain an array of GeometryCollection objects with the distance for clustering set to 2.

```
SELECT ST_AsText(unnest(ST_ClusterWithin(geom,2)))
 from (select ARRAY['LINESTRING (0 0,0 1)'::geometry,
                    'LINESTRING (2 3,3 3)'::geometry,
                    'LINESTRING (0 1,2 3)'::geometry,
                    'POINT (-1 -1)'::geometry] as geom) as test;
                          st_astext
----------------------------------------------------------------
 GEOMETRYCOLLECTION(LINESTRING(0 0,0 1),LINESTRING(2 3,3 3),LINESTRING(0 1,2 3),POINT(-1
-1))
(1 row)
```

4.8. Linear referencing functions

4.8.1. ST_LineInterpolatePoint

This topic describes the ST_LineInterpolatePoint function. This function interpolates a point at the specified location of a LineString object and returns the point.

Syntax

```
geometry  ST_LineInterpolatePoint(geometry  aLinestring , float8  aFraction);
```

Parameters

Parameter	Description
aLinestring	The LineString object that you want to specify.
aFraction	The fraction based on which the point is interpolated into the line. The value of this parameter is a floating-point number within the range of 0 to 1.

Description

- This function supports 3D objects and does not delete z coordinates.
- This function supports m coordinates.

Examples

```
SELECT ST_AsText(ST_LineInterpolatePoint(geom, 0.5)) FROM (SELECT 'LINESTRING(0 0,2 2)'::
geometry as geom) As test;
 st_astext
------------
 POINT(1 1)
(1 row)
```

- Obtain the closest point on a line to a geometry object.

```
SELECT ST_AsText(ST_LineInterpolatePoint(geom, ST_LineLocatePoint(geom, 'POINT(1 1)'::geo
metry))) FROM (SELECT 'LINESTRING(0 0,0 2)'::geometry As geom) As test;
 st_astext
------------
 POINT(0 1)
(1 row)
```

4.8.2. ST_LineInterpolatePoints

This topic describes the ST_LineInterpolatePoints function. This function interpolates a point at one or more specified locations of a LineString object and returns the points.

Syntax

```
geometry  ST_LineInterpolatePoints(geometry  aLinestring , float8  aFraction , boolean  rep
eat);
```

Parameters

Parameter	Description
aLinestring	The LineString object that you want to specify.

Parameter	Description
aFraction	The fraction based on which the point is interpolated into the line. The value of this parameter is a floating-point number within the range of 0 to 1.
repeat	Specifies whether to allow the repeated interpolation of the point. Default value: true.

Description

- If you set the repeat parameter to True, this function interpolates a point at every location that is represented as a fraction along the line.
- If the result is zero or one point, this function returns the result as a point object. Otherwise, this function returns the result as a MultiPoint object.
- This function supports 3D objects and does not delete z coordinates.
- This function supports m coordinates.

Examples

- Interpolate a point by using the default parameter settings.

```
SELECT ST_AsText(ST_LineInterpolatePoints('LINESTRING(0 0,0 5)', 0.20));
            st_astext
---------------------------------
 MULTIPOINT(0 1,0 2,0 3,0 4,0 5)
(1 row)
```

- Interpolate a point with the repeat parameter set to False.

```
SELECT ST_AsText(ST_LineInterpolatePoints('LINESTRING(0 0,0 5)', 0.20, false));
 st_astext
------------
 POINT(0 1)
(1 row)
```

4.8.3. ST_LineLocatePoint

This topic describes the ST_LineLocatePoint function. This function returns a floating-point number within the range of 0 to 1 to represent the location of the closest point in a LineString object to a point object in contrast to the total length of the LineString object.

Syntax

```
float8  ST_LineLocatePoint(geometry  aLinestring , geometry  aPoint);
```

Parameters

Parameter	Description
aLinestring	The LineString object that you want to specify.
aPoint	The point object that you want to specify.

Description

This function can be used to estimate the number of addresses.

Examples

```
SELECT ST_LineLocatePoint('LINESTRING(0 0,0 2)'::geometry, 'POINT(1 1)'::geometry);
 st_linelocatepoint
--------------------
                0.5
(1 row)
```

4.8.4. ST_LineSubstring

This topic describes the ST_LineSubstring function. This function returns a substring of a LineString object.

Syntax

```
geometry  ST_LineSubstring(geometry  aLinestring , float8  startfraction , float8  endfraction);
```

Parameters

Parameter	Description
aLinestring	The LineString object that you want to specify.
startfraction	The start position of the substring.
endfraction	The end position of the substring.

Description

- If you set the startfraction parameter and the endfraction parameter to the same value, the ST_LineSubstring function is equivalent to the ST_LineInterpolatePoint function.
- Before you can obtain a substring of a MultiLineString object, you must use the ST_LineMerge function to merge the LineStrings of the MultiLineString object.
- The ST_LineSubstring function supports 3D objects and does not delete z coordinates.
- The ST_LineSubstring function supports m coordinates.

Examples

```
SELECT ST_AsText(ST_LineSubstring('LINESTRING(0 0,10 0)'::geometry,0.3,0.6));
      st_astext
---------------------
 LINESTRING(3 0,6 0)
(1 row)
```

4.8.5. ST_LocateAlong

This topic describes the ST_LocateAlong function. This function returns the element that has the specified m coordinate in a geometry object.

Syntax

```
geometry  ST_LocateAlong(geometry  ageomWithMeasure , float8  aMeasure , float8  offset);
```

Parameters

Parameter	Description
ageomWithMeasure	The geometry object that you want to specify.
aMeasure	The m coordinate that you want to specify.
offset	The offset that is applied during the computation.

Description

- This function does not support polygon objects.
- This function is valid only for objects that have m coordinates.
- If you specify an offset, this function counts a specific number of units to the left or to the right along the input line.

  If you specify a positive offset, this function counts a specific number of units to the left. If you specify a negative offset, this function counts a specific number of units to the right.

Examples

```
SELECT ST_AsText(ST_LocateAlong(ST_GeomFromText('MultiPointM((1 2 3),(5 4 3),(4 5 6))'),3))
;
         st_astext
----------------------------
 MULTIPOINT M (1 2 3,5 4 3)
(1 row)
```

4.8.6. ST_LocateBetween

This topic describes the ST_LocateBetween function. This function returns the GeometryCollection object whose m coordinates are within the specified range.

Syntax

```
geometry  ST_LocateBetween(geometry  geomA , float8  measureStart , float8  measureEnd , fl
oat8  offset);
```

Parameters

Parameter	Description
geomA	The GeometryCollection object that you want to specify.
measureStart	The minimum m coordinate that is allowed.
measureEnd	The maximum m coordinate that is allowed.
offset	The offset that is applied during the computation.

Description

- This function does not support polygon objects.
- If you specify an offset, this function counts a specific number of units to the left or to the right along the input line.

  If you specify a positive offset, this function counts a specific number of units to the left. If you specify a negative offset, this function counts a specific number of units to the right.

Examples

```
SELECT ST_AsText(ST_LocateBetween(ST_GeomFromText('MultiPointM((1 2 3),(4 5 6),(7 8 9))'),3
,6));
         st_astext
----------------------------
 MULTIPOINT M (1 2 3,4 5 6)
(1 row)
```

4.8.7. ST_LocateBetweenElevations

This topic describes the ST_LocateBetweenElevations function. This function returns the GeometryCollection object whose heights are within the specified range.

Syntax

```
geometry  ST_LocateBetweenElevations(geometry  geomMline , float8  elevationStart , float8
 elevationEnd);
```

Parameters

Parameter	Description
geomMline	The GeometryCollection object that you want to specify.
elevationStart	The minimum height that is allowed.
elevationEnd	The maximum height that is allowed.

Description

This function supports 3D objects and does not delete z coordinates.

Examples

```
SELECT ST_AsText(ST_LocateBetweenElevations(ST_GeomFromText('MultiPoint((1 2 3),(4 5 6),(7
8 9))'),3,6));
         st_astext
----------------------------
 MULTIPOINT Z (1 2 3,4 5 6)
(1 row)
```

4.8.8. ST_InterpolatePoint

This topic describes the ST_InterpolatePoint function. This function returns the m coordinate of the closest point in a geometry object to a point object.

Syntax

```
float8  ST_InterpolatePoint(geometry  line , geometry  point);
```

Parameters

Parameter	Description
line	The LineString object that you want to specify.
point	The point object that you want to specify.

Description

This function supports 3D objects and does not delete z coordinates.

Examples

```
SELECT ST_InterpolatePoint('LINESTRING M (0 0 0, 2 0 2)', 'POINT(1 1)');
 st_interpolatepoint
---------------------
                   1
(1 row)
```

4.8.9. ST_AddMeasure

This topic describes the ST_AddMeasure function. This function linearly interpolates m coordinates into the specified range of a geometry object and returns a new geometry object.

Syntax

```
geometry  ST_AddMeasure(geometry  geomMline , float8  measureStart , float8  measureEnd);
```

Parameters

Parameter	Description
geomMline	The LineString object that you want to specify.
measureStart	The minimum m coordinate that is allowed.
measureEnd	The maximum m coordinate that is allowed.

Description

- If the geometry object that you specify does not have an m dimension, add an m dimension to the geometry object. If the geometry object that you specify has an m dimension, this function overwrites the original m coordinates with the new m coordinates.
- This function supports only MultiLineString objects.
- This function supports 3D objects and does not delete z coordinates.

Examples

```
SELECT ST_AsText(ST_AddMeasure('LINESTRING(1 0, 2 0)'::geometry,1,2));
         st_astext
----------------------------
 LINESTRING M (1 0 1,2 0 2)
(1 row)
```

5.Raster SQL reference

5.1. Basic concepts

This topic introduces the basic concepts related to Raster SQL.

Term	Description
raster object	The raster that is short for a raster object. It is regular grids into which space is divided. Each grid is called a cell. Each cell is assigned an attribute value to represent a data model of an entity. A raster object can be a satellite image, digital elevation model (DEM), or picture.
cell/pixel	The raster cell. It is also called a pixel. This indicates that a cell is a grid in a raster object. Each cell can have different data types, such as Byte, Short, Int, Double.
band	The raster band. A band is a single matrix of cell values in a raster object. A raster object can have multiple bands.
chunk	The raster chunk. You can customize the size of a chunk, such as 256 × 256 × 3.
pyramid	The raster pyramid. The downsampled version of a source raster object. A pyramid can contain multiple downsampled layers. Consecutive pyramid layers are downsampled at a scale of 2:1. Layer 0 stores the raw data.
pyramid level	The layer of a raster pyramid.
mosaic	The operation to integrate multiple raster objects into an existing raster dataset.
interleaving	The interleaving method of pixels in a raster. The interleaving methods are band sequential (BSQ), band interleaved by pixel (BIP), and band interleaved by line (BIL).
world space	The world coordinate space. It indicates the geographic coordinate space of a raster object.
raster space	The coordinate space of a raster. It indicates the pixel coordinate space of a raster object. The **upper-left corner** of the raster is used as the starting point.
metadata	The metadata of a raster, including the spatial range, projection type, and pixel type. The metadata of the remote sensing platform is excluded.

5.2. Parallel operations

Ganos uses multiple CPUs to increase the performance of queries or computing. This feature is called parallel operation. Ganos supports parallel SQL statements and parallel raster-level operations.

Execute SQL statements in parallel

- How it works

PostgreSQL uses multiple CPUs to generate multiple query plans in parallel. Then, PostgreSQL balances the new query plans across these CPUs. This increases the query performance.

- Scenario

  If you want to scan a large number of raster objects to identify those that meet the specified spatial scope or attribute requirements, we recommend that you execute parallel SQL statements. This significantly reduces the time that is required to run queries.

- Scope

  All the read-only functions that are used to compute raster objects in Ganos support parallel queries. These functions include the function that is used to query the attributes of a raster object.

Manage raster objects in parallel

- How it works

  In most cases, Ganos uses multiple CPUs to perform multiple operations in parallel on the subsets of a raster object. This reduces the time that is required for the overall computation on the raster object. Each subset is independently computed. When the computations on all the subsets are complete, the parallel operations on the raster object end.

- Scenario

  The raster object that you want to compute is large. In this case, a long period of time is required to complete the overall computation.

- Scope

  For more information about the functions that support parallel raster-level operations, see the following topics:

  - ST_ImportFrom
  - ST_BuildPyramid
  - ST_RPCRectify

Before you start

- Set the maximum number of transactions that can be used in the prepared state.

  You can use the max_prepared_transactions parameter to set the maximum number. The default value of this parameter is 0. We recommend that you set this parameter to the value of the max_connections parameter. This ensures that one transaction in the Prepared state is assigned to each connection.

  > **Note** After you reconfigure this parameter, you must restart your instance.

- Set the degree of parallelism

  Before you invoke a function that supports parallel raster-level operations, you must set the degree of parallelism. By default, the value of a Grand Unified Configuration (GUC) parameter is used as the degree of parallelism. This applies if you do not specify the degree of parallelism or set it to 0. The GUC parameter is set to ganos.parallel.degree. The default value of the ganos.parallel.degree is 1. The value 1 indicates that the function does not support parallel raster-level operations.

  You can also set this parameter to a positive integer for a function that supports parallel raster-level operations. Then, you can split the computation on a raster object into parallel tasks. Example:

```
select ST_ImportFrom('chunk_table','OSS://<akxxxx>:<ak_secretxxxx>@oss-cn-beijing-internal.aliyuncs.com/mybucket/data/image.nc:hcc', '{}', '{"parallel": 4}');
```

- Set the transactional consistency

  You can specify the transactional consistency by using the ganos.parallel.transaction GUC parameter. Valid values:

  - **transaction_commit** specifies that Ganos can commit or roll back parallel transactions based on the main transaction. This is the default value.
  - **fast_commit**: specifies that Ganos cannot commit or roll back parallel transactions based on the main transaction.

> **Note**
> - To ensure the optimal performance, you can use the ST_CreateChunkTable function to create a chunk table in advance.
> - If a parallel function cannot return a temporary chunk table that is created by an anonymous user, you must first create a chunk table. Then, specify the chunk table in the chunktable parameter.

5.3. Object storage paths

Ganos allows you to create, import, and export raster data stored in Alibaba Cloud Object Storage Service (OSS) buckets and self-managed Multi-Cloud Object Storage (MinIO) buckets. This topic describes the paths of the raster data in the buckets.

Paths of raster data in OSS buckets

The path of raster data in an OSS bucket is in the following format:

```
oss://<access_id>:<secrect_key>@[<Endpoint>]/path_to/file[:<subset>]
```

> **Note** If the OSS bucket is in Network Common Data Form (NetCDF) and contains a subset, you must specify the **:<subset>** field.

The Endpoint parameter is optional. If you do not specify this parameter, Ganos obtains the required endpoint. If you do not specify this parameter, make sure that the path starts with a forward slash (/). The Endpoint parameter specifies the region where the OSS bucket resides. To import data at high performance, make sure that your ApsaraDB RDS instance and the OSS bucket reside in the same region. For more information, see OSS domain names.

Paths of raster data in MinIO buckets

The path of raster data in a MinIO bucket is in the following format. For more information about MinIO buckets, see Multi-Cloud Object Storage.

```
mio://<id>:<key>@<host>[:<port>]/path_to/file[:<subset>]
```

> **Note** If the MinIO bucket is in NetCDF and contains a subset, you must specify the **:<subset>** field.

If you do not specify the port parameter, the HTTPS port is used. To import data at high performance, make sure that your ApsaraDB RDS instance and the self-managed MinIO bucket reside in the same region and can communicate over an internal network.

5.4. Raster creation

5.4.1. ST_CreateRast

This topic describes the ST_CreateRast function. This function creates a raster object by using the data stored in an Alibaba Cloud Object Storage Service (OSS) bucket or a self-managed Multi-Cloud Object Storage (MinIO) bucket.

Syntax

```
raster ST_CreateRast(cstring url);
raster ST_CreateRast(cstring url, cstring storageOption);
```

Parameters

Parameter	Description
url	The path of the object based on which you want to create a raster object.
storageOption	A JSON string that describes the chunks for storing the pyramid of the raster object.

The following table describes the fields of the storageOption parameter.

Field	Description	Type	Format	Default value	Description
chunkdim	The size of each chunk that is used to store the data of the raster object.	String	(w, h, b)	Same as the size of each chunk in the original object	N/A

Field	Description	Type	Format	Default value	Description
interleaving	The interleaving type of the raster object.	String	N/A	bsq	Valid values: • bip: band interleaved by pixel (BIP) • bil: band interleaved by line (BIL) • bsq: band sequential (BSQ) • auto: an interleaving method that is specified by this function

Note You need to change the default values of the chunkdim and interleaving fields only in some cases:

- Users want to view the raster object based on multiband red, green, and blue (RGB) combination, but the value of the interleaving field is bsq. In this case, you must change the value of the interleaving field to bip.
- The chunks of some images that are used to render the raster object contain 1 row and n columns in size. However, users request chunks that contain 256 rows and 256 columns in size. In this case, you must change the value of the chunkdim field to the requested chunk size.

Description

You can obtain the supported data types by using the ST_RasterDrivers function.

Examples

```
-- Specify the AccessKey ID, AccessKey secret, and endpoint in the URL of an OSS object to
create a raster object.
Select ST_CreateRast('OSS://<ak>:<ak_secret>@oss-cn-beijing-internal.aliyuncs.com/mybucket/
data/image.tif');
-- Specify the host and port number in the URL of a MinIO bucket to create a raster object.
Select ST_CreateRast('OSS://<ak>:<ak_secret>@10.0.0.1:443/mybucket/data/image.tif');
-- Specify the chunk size and the interleaving type of a raster object.
Select ST_CreateRast('OSS://<ak>:<ak_secret>@oss-cn-beijing-internal.aliyuncs.com/mybucket/
data/image.tif', '{"chunkdim":"(256,256,3)","interleaving":"auto"}');
-- Specify a NetCDF image that contains subsets.
Select ST_CreateRast('OSS://<ak>:<ak_secret>@oss-cn-beijing-internal.aliyuncs.com/mybucket/
data/image.nc:hcc');
```

5.5. Import and export

5.5.1. ST_ImportFrom

This topic describes the ST_ImportFrom function. This function is used to import an Object Storage Service (OSS) object into a raster object in PolarDB.

Syntax

```
raster ST_ImportFrom(cstring chunkTableName,
                     cstring url,
                     cstring storageOption default '{}',
                     cstring importOption default '{}');
```

Parameters

Parameter	Description
chunkTableName	The name of the OSS object. The OSS object is a chunk table. The name must comply with the table naming conventions of PolarDB.
url	The URL of the OSS object. For more information, see the description of the url parameter in ST_CreateRast.
storageOption	The JSON string that is used to specify the storage information about the raster object.
importOption	The JSON string that is used to configure the parameters for OSS object import.

The following table describes the parameters in storageOption.

Parameter	Type	Default value	Description
chunking	boolean	true	Specifies whether to store the data as chunks.
chunkdim	string	The default value is the same as the value for the raw data.	The dimensions that are used to chunk data. Format: w, h, b. **Note** This parameter takes effect only if you set the chunking parameter to **true**.
compression	string	lz4	The format that is used to compress the data of the OSS object. Valid values: • none • jpeg • zlib • png • lzo • lz4 • snappy • zstd • jp2k

Parameter	Type	Default value	Description
quality	integer	75	The image quality of the raster object after compression. This parameter is valid only for the JPEG and JPEG 2000 compression formats.
interleaving	string	The default value is the same as the value for the raw data.	The method that is used to interleave the data of the OSS object. Valid values: • bip : Band interleaved by pixel • bil : Band nterleaved by pixel • bsq : Band Sequential
blockendian	string	'NDR'	The sequence that is used to store the bytes in each data chunk. Valid values: • NDR: little endian • XDR: big endian
celltype	string	The default value is the same as the value for the raw data.	The pixel type of the raster object. Valid values: • 1bb: 1 bit • 2bui: 2-bit unsigned integer • 4bui : 4bit unsigned integer • 8bs : 8bit signed integer • 8bui : 8bit unsigned integer • 16bsi : 16bit signed integer • 16bui : 16bit unsigned integer • 32bsi : 32bit signed integer • 32bui : 32bit unsigned integer • 32bf : 32bit float • 64bsi : 64bit signed integer • 64bui : 64bit unsigned integer • 64bf : 64bit float

The following table describes the parameters in importOption.

Parameter	Type	Default value	Description

Parameter	Type	Default value	Description
mapping_oss_file	boolean	false	Specifies whether to map the OSS object to a memory object. **Note** When raster data is read, some drivers do not optimize the data stored in OSS. As a result, a large number of requests that require a small volume of data are generated. This reduces performance. This parameter can map the OSS object to a memory object to improve performance. You can use the Global User Configuration (GUC) parameter ganos.raster.memory_oss_file_max_size to specify the size of the OSS object.
parallel	integer	Value of the GUC parameter ganos.parallel.degree	The degree of parallelism. Valid values: 1 to 64.

Description

This function is used to create a raster object and import an OSS object into the created raster object.

You can use the ST_RasterDrivers function to query the data types supported by the ST_ImportFrom function. For more information about the ST_RasterDrivers function, see ST_RasterDrivers.

Examples

```
-- Specify only the name and path of the OSS object that you want to import.
Select ST_ImportFrom('chunk_table','OSS://<ak>:<ak_secret>@oss-cn-beijing-internal.aliyuncs
.com/mybucket/data/image.tif');
-- Specify a NetCDF image that contains subsets.
Select ST_ImportFrom('chunk_table','OSS://<ak>:<ak_secret>@oss-cn-beijing-internal.aliyuncs
.com/mybucket/data/image.nc:hcc');
-- Specify the size per chunk and the compression format in the process of importing the OS
S object.
Select ST_ImportFrom('chunk_table','OSS://<ak>:<ak_secret>@oss-cn-beijing-internal.aliyuncs
.com/mybucket/data/image.tif', '{"chunkdim":"(128,128,3)", "compression":"none"}');
-- Specify the degree of parallelism in the process of importing the OSS object.
Select ST_ImportFrom('chunk_table','OSS://<ak>:<ak_secret>@oss-cn-beijing-internal.aliyuncs
.com/mybucket/data/image.nc:hcc', '{}', '{"parallel": 4}');
```

5.5.2. ST_ExportTo

This function exports a raster object as an Object Storage Service (OSS) object.

Syntax

```
boolean ST_ExportTo(raster source, cstring format, cstring url, integer level = 0);
```

Parameters

Parameter	Description
source	The raster object to be exported.
format	The format of the exported data, such as GTiff or BMP.
url	The URL of the exported OSS object. For more information, see the description of the url parameter in ST_CreateRast.
level	The pyramid level.

Description

If the raster object is successfully exported, the function returns true. If the raster object fails to be exported, the function returns false.

The format parameter specifies the format of the exported data. The following table lists the common formats.

Format	Full name
BMP	Microsoft Windows Device Independent Bitmap(.bmp)
EHdr	ESRI .hdr Labelled
ENVI	ENVI .hdr Labelled Raster
GTiff	TIFF/BigTIFF/GeoTIFF(.tif)
HFA	Erdas Imagine .img
RST	Idrisi Raster Format
INGR	Intergraph Raster Format

Example

```
Select ST_ExportTo(raster, 'GTiff', 'OSS://ABCDEFG:1234567890@oss-cn.aliyuncs.com/mybucket/
data/4.tif') from raster_table where id=1;
```

5.5.3. ST_AsImage

This topic describes the ST_AsImage function, which converts a raster into a BYTEA-type image.

Syntax

```
bytea ST_AsImage(raster raster_obj,
        box extent,
        integer pyramidLevel default 0,
        cstring bands default '',
        cstring format default 'PNG',
        cstring option default '');
```

Parameters

Parameter	Description
raster_obj	The raster that you want to convert.
extent	The extent of the image. The geographic coordinate system is used by default.
pyramidLevel	The pyramid layer at which the image resides. Valid values start from 0. Default value: 0.
bands	The list of bands based on which you want to obtain the image. Valid values start from 0. Examples: `'0-2'` and `'1,2,3'`. This parameter is empty by default. If the image is in the JPEG format, set this parameter to 1 or 3. If the image is in the PNG format, set this parameter to 1, 2, 3, or 4. The first three bands are used by default.
format	The format of the image. Valid values: • PNG • JPEG
option	The options that are used to convert JSON-formatted strings.

The following table describes the fields in the option parameter.

Field	Description	Type	Default value	Setting notes
nodata	Specifies whether to use NoData values.	bool	false	• true: NoData values are processed. • false: NoData values are processed as normal values.
nodataValue	The NoData value of the raster.	integer	0	If the nodata parameter is set to true, you must specify the NoData value.
rast_coord	Specifies whether the input box is specified by using pixel coordinates.	bool	false	If pixel coordinates are used, x indicates the column No. of the pixel and y indicates the row No. of the pixel. Both the column No. and the row No. start from 0.

Field	Description	Type	Default value	Setting notes
strength	Specifies whether to implement enhancement in the display.	string	none	Valid values: • none: Enhancement is not implemented. • stats: Enhancement is implemented by using stretching based on statistical values.
quality	The quality of compression.	integer	75	Valid values: 1 to 100.

Description

- This function returns a BYTEA-type image.
- Up to 100 MB of cropped data can be cached by default, which indicates that up to 100 MB of data can be returned. To adjust the limit on the image size, you can specify the cache size by using the **ganos.raster.clip_max_buffer_size** parameter.
- The following values are valid for this parameter:
  - 1: The raster has a single band based on which the raster can be converted into a grayscale image.
  - 2: The raster has a single band, based on which the raster can be converted into a grayscale image, and the Alpha band.
  - 3: The raster has three bands, which are the R band, G band, and B band.
  - 4: The raster has four bands, which are the R band, G band, B band, and Alpha band.

Examples

```
--Specify the size of cropped data that can be cached.
SELECT ST_AsImage(raster_obj,
                  '(-180,-90), (0,0)'::Box)
FROM raster_table
WHERE id=1;
--Specify the pyramid layer at which the image resides.
SELECT ST_AsImage(raster_obj,
                  '(-180,-90), (0,0)'::Box,
                 1)
FROM raster_table
WHERE id=1;
--Specify the cropping range of a band.
SELECT ST_AsImage(raster_obj,
                  '(-180,-90), (0,0)'::Box,
                 1,
                 '0-2')
FROM raster_table
WHERE id=1;
--Specify the compression format.
SELECT ST_AsImage(raster_obj,
                  '(-180,-90), (0,0)'::Box,
                 1,
                 '0-2',
                 'PNG')
FROM raster_table
WHERE id=1;
--Specify to implement enhancement by using stretching based on statistical values.
SELECT ST_AsImage(rast,
                  '(-180,-90), (0,0)'::Box,
                  0,
                  '',
                  'PNG',
                  '{"nodata":"false", "nodatavalue":"0","rast_coord":"false", "strength":"s
tats", "quality":"75"}')
FROM raster_table
WHERE id=1;
--Specify the pixel coordinates based on which you want to crop the image and implement enh
ancement by using stretching based statistical values.
SELECT ST_AsImage(rast,
                  '(0,0), (200,100)'::Box,
                  0,
                  '',
                  'PNG',
                  '{"nodata":"false", "nodatavalue":"0","rast_coord":"true", "strength":"st
ats", "quality":"75"}')
FROM raster_table
WHERE id=1;
```

5.5.4. ST_AsPNG

This topic describes the ST_AsPNG function, which converts a raster into a BYTEA-type PNG image.

Syntax

```
bytea ST_AsPNG(raster raster_obj,box extent,
               integer pyramidLevel default 0,
               cstring bands default '',
               cstring option default '');
```

Parameters

Parameter	Description
raster_obj	The raster that you want to convert.
extent	The extent of the image. The geographic coordinate system is used by default.
pyramidLevel	The pyramid layer at which the image resides. Valid values start from 0. Default value: 0.
bands	The list of bands based on which you want to obtain the image. Valid values start from 0. Examples: `'0-2'` and `'1,2,3'`. This parameter is empty by default. The image is in the PNG format. Therefore, set this parameter to 1, 2, 3, or 4. The first three bands are used by default.
option	The options that are used to convert JSON-formatted strings.

The following table describes the fields in the option parameter.

Field	Description	Type	Default value	Setting notes
nodata	Specifies whether to use NoData values.	bool	false	• true: NoData values are processed. • false: NoData values are processed as normal values.
nodataValue	The NoData value of the raster.	integer	0	If the nodata parameter is set to true, you must specify the NoData value.
rast_coord	Specifies whether the input box is specified by using pixel coordinates.	bool	false	If pixel coordinates are used, x indicates the column number of the pixel and y indicates the row number of the pixel. Both the column number and the row number start from 0.

Field	Description	Type	Default value	Setting notes
strength	Specifies whether to implement enhancement in the display.	string	none	Valid values: • none: Enhancement is not implemented. • stats: Enhancement is implemented by using stretching based on statistical values.
quality	The quality of compression.	integer	75	Valid values: 1 to 100.

Description

- This function returns a BYTEA-type PNG image.
- Up to 100 MB of cropped data can be cached by default, which indicates that up to 100 MB of data can be returned. To adjust the limit on the image size, you can specify the cache size by using the **ganos.raster.clip_max_buffer_size** parameter.
- The following values are valid for this parameter:
  - 1: The raster has a single band based on which the raster can be converted into a grayscale image.
  - 2: The raster has a single band, based on which the raster can be converted into a grayscale image, and the Alpha band.
  - 3: The raster has three bands, which are the R band, G band, and B band.
  - 4: The raster has four bands, which are the R band, G band, B band, and Alpha band.

Examples

```
--Specify the size of cropped data that can be cached.
SELECT ST_AsPNG(raster_obj,
                  '(-180,-90), (0,0)'::Box)
FROM raster_table
WHERE id =1;
--Specify the pyramid layer at which the image resides.
SELECT ST_AsPNG(raster_obj,
                  '(-180,-90), (0,0)'::Box,
                 1)
FROM raster_table
WHERE id =1;
--Specify the cropping range of a band.
SELECT ST_AsPNG(raster_obj,
                  '(-180,-90), (0,0)'::Box,
                 1,
                 '0-2')
FROM raster_table
WHERE id =1;
--Specify to implement enhancement by using stretching based on statistical values.
SELECT ST_AsPNG(rast,
                  '(-180,-90), (0,0)'::Box,
                  0,
                  '',
                  '{"nodata":"false", "nodatavalue":"0","rast_coord":"false", "strength":"s
tats", "quality":"75"}')
FROM raster_table
WHERE id =1;
```

5.5.5. ST_AsJPEG

This topic describes the ST_AsJPEG function, which converts a raster into a BYTEA-type JPEG image.

Syntax

```
bytea ST_AsJPEG(raster raster_obj,
                box extent,
                integer pyramidLevel default 0,
                cstring bands default '',
                cstring option default '');
```

Parameters

Parameter	Description
raster_obj	The raster that you want to convert.
extent	The extent of the image. The geographic coordinate system is used by default.
pyramidLevel	The pyramid layer at which the image resides. Valid values start from 0. Default value: 0.

Parameter	Description
bands	The list of bands based on which you want to convert the raster. Valid values start from 0. Examples: `'0-2'` and `'1,2,3'`. This parameter is empty by default. The image is in the JPEG format. Therefore, set this parameter to 1, 2, 3, or 4. The first three bands are used by default.
option	The options that are used to convert JSON-formatted strings.

The following table describes the fields in the option parameter.

Field	Description	Type	Default value	Setting notes
nodata	Specifies whether to use NoData values.	bool	false	• true: NoData values are processed. • false: NoData values are processed as normal values.
nodataValue	The NoData value of the raster.	integer	0	If the nodata parameter is set to true, you must specify the NoData value.
rast_coord	Specifies whether the input box is specified by using pixel coordinates.	bool	false	If pixel coordinates are used, x indicates the column No. of the pixel and y indicates the row No. of the pixel. Both the column No. and the row No. start from 0.
strength	Specifies whether to implement enhancement in the display.	string	none	Valid values: • none: Enhancement is not implemented. • stats: Enhancement is implemented by using stretching based on statistical values.
quality	The quality of compression.	integer	75	Valid values: 1 to 100.

Description

- This function returns a BYTEA-type image.
- Up to 100 MB of cropped data can be cached by default, which indicates that up to 100 MB of data can be returned. To adjust the limit on the image size, you can specify the cache size by using the **ganos.raster.clip_max_buffer_size** parameter.
- The following values are valid for this parameter:
  - 1: The raster has a single band based on which the raster can be converted into a grayscale image.
  - 3: The raster has three bands, which are the R band, G band, and B band.

Examples

```
--Specify the size of cropped data that can be cached.
SELECT ST_AsJPEG(raster_obj,
                  '(-180,-90), (0,0)'::Box)
FROM raster_table
WHERE id =1;
--Specify the pyramid layer at which the image resides.
SELECT ST_AsJPEG(raster_obj,
                  '(-180,-90), (0,0)'::Box,
                 1)
FROM raster_table
WHERE id =1;
--Specify the cropping range of a band.
SELECT ST_AsJPEG(raster_obj,
                  '(-180,-90), (0,0)'::Box,
                 1,
                 '0-2')
FROM raster_table
WHERE id =1;
--Specify to implement enhancement by using stretching based on statistical values.
SELECT ST_AsJPEG(rast,
                  '(-180,-90), (0,0)'::Box,
                  0,
                  '',
                  '{"nodata":"false", "nodatavalue":"0","rast_coord":"false", "strength":"s
tats", "quality":"75"}')
FROM raster_table
WHERE id =1;
```

5.5.6. ST_AsDatasetFile

This topic describes the ST_AsDatasetFile function. This function is used to convert a specific part of a raster object into a BYTEA-type file.

Syntax

```
setof record ST_AsDatasetFile(raster raster_obj,
                              box extent,
                              integer pyramidLevel default 0,
                              cstring bands default '',
                              cstring format default 'GTiff',
                              cstring create_option default '{}',
                              cstring process_option default '{}',
                              out ext cstring,
                              out data bytea);
```

Parameters

Parameter	Description
raster_obj	The raster object whose part you want to convert.

Parameter	Description
extent	The part of data that you want to extract. By default, the part of data is specified based on a geographic coordinate system (GCS).
pyramidLevel	The pyramid level from which you want to extract data. Valid values start from 0. Default value: 0.
bands	The IDs of the bands from which you want to extract data. Valid band IDs start from 0. The value of this parameter can be a range or array of band IDs. Examples: `'0-2'` and `'1,2,3'`. Default value: `''`. The default value specifies that Ganos extracts data from all bands.
format	The format of the file. For more information, see ST_RasterDrivers.
create_option	The options based on which you want to create a dataset by using the extracted data. The value of this parameter is a JSON string. For more information, see ST_RasterDrivers.
process_option	The options based on which you want to process the extracted data. The value of this parameter is a JSON string. The rast_coord option specifies whether pixel coordinates are used to specify the input bounding box. If pixel coordinates are used to specify the input bounding box, the x-coordinate specifies the column No. of a pixel and the y-coordinate specifies the row No. of a pixel. The valid values of the x-coordinate and y-coordinate start from 0.
ext	The format of the file. The supported file formats include TIF and XML.
data	The BYTEA-type data of the file.

Description

- The file format that you specify must be supported by a driver in Ganos. Therefore, you must set the can_asfile parameter in the ST_RasterDrivers function to true. For more information, see ST_RasterDrivers.
- The default size of the cache that stores the extracted data is 100 MB. This means that up to 100 MB of extracted data can be returned. If you want to adjust the size of the extracted data that is returned, you can change the cache size by using the ganos.raster.clip_max_buffer_size parameter.
- The value of the create_option parameter can be obtained from the create_options parameter of the ST_RasterDrivers function. For more information, see ST_RasterDrivers.

Examples

```
--Specify the part of data that you want to extract.
SELECT ST_AsDatasetFile(raster_obj,
                  '(-180,-90), (0,0)'::Box)
FROM raster_table
WHERE id =1;
--Specify the pyramid level from which you want to extract data.
SELECT ST_AsDatasetFile(raster_obj,
                  '(-180,-90), (0,0)'::Box,
                 1)
FROM raster_table
WHERE id =1;
--Specify the IDs of the bands from which you want to extract data.
SELECT ST_AsDatasetFile(raster_obj,
                  '(-180,-90), (0,0)'::Box,
                 1,
                 '0-2')
FROM raster_table
WHERE id =1;
--Specify that Ganos saves the extracted data as a GIF file.
SELECT ST_AsDatasetFile(raster_obj,
                  '(-180,-90), (0,0)'::Box,
                 1,
                 '0-2',
                       'GIF')
FROM raster_table
WHERE id =1;
--Specify the file format and options that are used to process the extracted data.
SELECT ST_AsDatasetFile(raster_obj,
                  '(-180,-90), (0,0)'::Box,
                 1,
                 '0-2',
                       'GTiff',
                       '{"blockxsize":256, "blockysize":256, "compress": "DEFLATE"}')
FROM raster_table
WHERE id =1;
--Specify that pixel coordinates are used to specify the input bounding box.
SELECT ST_AsDatasetFile(raster_obj,
                  '(0,0), (100,100)'::Box,
                 1,
                 '0-2',
                       'GTiff',
                       '{}',
                       '{"rast_coord":"true"}')
FROM raster_table
WHERE id =1;
```

5.6. Pyramid operations

5.6.1. ST_BuildPyramid

This topic describes the ST_BuildPyramid function, which is used to create a pyramid for a raster object.

Syntax

```
raster ST_BuildPyramid(raster source,
                       integer pyramidLevel default -1,
                       ResampleAlgorithm algorithm default 'Near',
                       cstring chunkTableName default '',
                       cstring storageOption default '{}',
                       cstring buildOption default '{}');
```

Parameters

Parameter	Description
source	The name of the raster object.
chunkTableName	The name of the chunk table that is stored in the pyramid. This parameter takes effect only when the raster object is stored in an Object Storage Service (OSS) bucket.
pyramidLevel	The number of levels in the pyramid. The value -1 specifies that the largest number of levels are created in the pyramid.
algorithm	The resampling algorithm that is used to create the pyramid. Valid values: • Near • Average • Bilinear • Cubic
storageOption	The JSON string that is used to specify the storage-related parameters. These parameters are used to describe the chunk storage information of the pyramid. This JSON string takes effect only when the raster object is stored in an OSS bucket.
buildOption	The JSON string that is used to specify the pyramid building-related parameters. PolarDB supports the parallel parameter. This parameter specifies the degree of parallelism. The data type of this parameter is INTEGER. The value of this parameter ranges from 1 to 64. If you do not specify the parallel parameter, the value of the GUC parameter ganos.parallel.degree is used as the default degree of parallelism. For more information, see ganos.parallel.degree. **Note** If you run threads in parallel to create the pyramid, transactions are not supported. If the pyramid cannot be created or you need to roll transactions back, you can invoke the ST_deletePyramid function to delete the pyramid. For more information, see ST_deletePyramid.

The following table describes the parameters in storageOption.

Parameter	Type	Description

Parameter	Type	Description
chunkdim	string	The dimensions that are used to chunk the data of the raster object. The value of this parameter follows the `(w, h, b)` format. By default, the size per chunk is obtained from the original image data.
interleaving	string	The method that is used to interleave the data of the raster object. • bip: band interleaved by pixel (BIP) • bil: band interleaved by line (BIL) • bsq: band sequential (BSQ) (This is the default value.) • auto: an interleaving method that is specified by this function
compression	string	The format into which you want to compress the data of the raster object. Valid values: • none • jpeg • zlib • png • lzo • LZ4 (This is the default value.) • zstd • snappy • jp2k
quality	integer	The quality of compression. This parameter takes effect only when you use the JPEG or JPEG 2000 compression format. Default value: 75.

Description

This function supports GPU-accelerated computing. By default, in an active environment with GPUs, Ganos automatically enables GPU-accelerated computing.

Examples

```
Update raster_table set raster_obj = ST_BuildPyramid(raster_obj) where id = 1;
Update raster_table set raster_obj = ST_BuildPyramid(raster_obj, 'chunk_table') where id =
2;
-- Use JPEG 2000 to compress the data of the raster object.
-- Make sure that all bands are within one chunk.
Update raster_table set raster_obj = ST_BuildPyramid(
    raster_obj,
    -1,
    'Near',
    'chunk_table',
    '{"compression":"jp2k", "quality": 75, "chunkdim":"(256,256,4)","interleaving":"auto"}'
)
where id = 3;
-- Create the pyramid by using parallel operations.
Update raster_table set raster_obj = ST_BuildPyramid(
    raster_obj,
    -1,
    'Near',
    'chunk_table',
    '{"compression":"jp2k", "quality": 75, "chunkdim":"(256,256,4)","interleaving":"auto"}'
,
    '{"parallel":4}')
where id = 3;
```

5.6.2. ST_deletePyramid

This function deletes the pyramid for a raster object.

Syntax

```
raster ST_deletePyramid(raster source);
```

Parameters

Parameter	Description
source	The raster object.

Description

This function deletes the pyramid, resets the metadata, and deletes the chunk data for the raster object.

Example

```
Update raster_table set raster_obj = ST_deletePyramid(raster_obj) where id = 1;
```

5.6.3. ST_BestPyramidLevel

This function computes the best pyramid level based on the world space, width, and height of a viewport.

Syntax

```
integer ST_BestPyramidLevel(raster rast, Box extent, integer width, integer height);
```

Parameters

Parameter	Description
rast	The raster object.
box	The world space of the viewport, in the format of `((minX,minY),(maxX,maxY))`.
width	The width of the viewport, in pixels.
height	The height of the viewport, in pixels.

Description

The raster object must have a valid spatial reference system identifier (SRID).

Examples

```
Select ST_BestPyramidLevel(raster_obj, '((128.0, 30.0),(128.5, 30.5))', 800, 600) from rast
er_table where id = 10;
---------------------
3
```

5.7. Coordinate system conversion

5.7.1. ST_Rast2WorldCoord

This function computes the world coordinates of a cell by using an affine transformation formula based on the pixel coordinates and pyramid level of the cell.

Syntax

```
point ST_Rast2WorldCoord(raster raster_obj, integer pyramidLevel, integer row, integer colu
mn);
geometry ST_Rast2WorldCoord(raster raster_obj, integer pyramidLevel, geometry geom);
```

Parameters

Parameter	Description
raster_obj	The raster object.

Parameter	Description
pyramidLevel	The pyramid level.
row	The row number of the cell.
column	The column number of the cell.
geom	The geometry must be converted. The x coordinate of the geometry represents the column number in the raster and the y coordinate of the geometry represents the row number .

Description

The raster object must have a valid spatial reference system identifier (SRID).

Example

```
Select ST_Rast2WorldCoord(raster_obj, 0, 0, 0) from raster_table;
```

5.7.2. ST_World2RastCoord

This function computes the pixel coordinates of a cell by using an inverse affine transformation formula based on the world coordinates and pyramid level of the cell.

Syntax

```
point ST_World2RastCoord(raster raster_obj, integer pyramidLevel, point coord);
geometry ST_World2RastCoord(raster raster_obj, integer pyramidLevel, geometry geom);
```

Parameters

Parameter	Description
raster_obj	The raster object.
pyramidLevel	The pyramid level.
coord	The world coordinates of the cell.
geom	The geometry to be converted.

Description

The raster object must have a valid spatial reference system identifier (SRID).

A geometry will be returned. The x coordinate is the column number in raster and the y coordinate is the row number.

Example

```
select st_world2rastcoord(rast, 0, '(117.3378,26.9020)'::point) from tb_dem where id = 2;
 st_world2rastcoord
--------------------
 (53205,32518)
SELECT ST_AsText(ST_world2RastCoord(rast, 0, ST_Rast2WorldCoord(rast, 0, 'POINT(511 0)'::ge
ometry)))
FROM tb_world2rast;
  st_astext
--------------
 POINT(511 0)
```

5.8. Pixel value operations

5.8.1. ST_AddZ

This topic describes the ST_AddZ function, which specifies the Z coordinate of a geometry based on the band of the raster converted from the geometry.

Syntax

```
geometry ST_AddZ(raster source,
                 geometry geom,
                 integer pyramid,
                 integer band);
```

Parameters

Parameter	Description
source	The raster from which the geometry is converted.
geom	The geometry whose Z coordinate you want to specify.
pyramid	The pyramid level at which the raster resides. Valid values start from 0. Default value: 0.
band	The band of the raster. Valid values start from 0. Default value: 0.

Description

This function specifies the Z coordinate of a geometry based on the band of the raster converted from the geometry. If the raster is georeferenced, this function returns a geographic coordinate of the geometry. Otherwise, this function returns a pixel coordinate of the geometry.

> **Note** All of the points on the geometry must fall within the raster.

Example

```
SELECT ST_AddZ(rast_object, ST_GeomFromText('POINT(120.5 30.6)', 4326), 0, 0)
FROM raster_table;
------------------------
POINT Z(120.5 30.6 27)
SELECT ST_AddZ(rast_object, ST_GeomFromText('POINT(120.5 30.6)', 4326), 1, 1)
FROM raster_table;
------------------------
POINT Z(120.5 30.6 115)
```

5.8.2. ST_ClipDimension

This function computes the raster space of a clipped area.

Syntax

```
box ST_ClipDimension(raster raster_obj, integer pyramidLevel, box extent);
```

Parameters

Parameter	Description
raster_obj	The raster object.
pyramidLevel	The pyramid level.
box	The world space of the clipped area.

Description

The raster object must have a valid spatial reference system identifier (SRID).

Examples

```
Select ST_ClipDimension(raster_obj, 2, '((128.0, 30.0),(128.5, 30.5))') from raster_table w
here id = 10;
------------------------------------
'((200, 300),(600, 720))'
```

5.8.3. ST_Clip

This function clips a raster object.

Syntax

```
bytea ST_Clip(raster raster_obj,integer pyramidLevel, box extent, BoxType boxType);
bytea ST_Clip(raster raster_obj,integer pyramidLevel, box extent, BoxType boxType, integer
destSrid);
record ST_Clip(raster raster_obj,
                     geometry geom,
                     integer pyramidLevel default 0,
                     cstring bands default '',
                     float8[] nodata default NULL,
                     cstring clipOption default '',
                     cstring storageOption default '',
                     out box outwindow,
                     out bytea rasterblob)
```

Parameters

Parameter	Description
raster_obj	The raster object.
pyramidLevel	The pyramid level.
extent	The area to be clipped, in the format of `'((minX,minY),(maxX,maxY))'`.
boxType	The coordinate type of the area to be clipped. Valid values: • Raster: pixel coordinates • World: world coordinates
destSrid	The spatial reference system identifier (SRID) of the output cell subset.
geometry	The geometry object used for clipping.
bands	The sequence numbers of bands to be clipped, in the format of `'0-2'` or `'1,2,3'`. The sequence number starts from 0. Default value: empty string (`''`). It indicates that all bands are to be clipped.
nodata	The array of NoData values in the format of float8[]. If the number of NoData values is fewer than the number of bands to be clipped, the predefined NoData value of a band is used to fill the area after the band is clipped. If a band has no predefined NoData value, the value 0 is used to fill the area after the band is clipped.
clipOption	The clipping options. The value is a JSON-formatted string.
storageOption	The storage options for the output. The value is a JSON-formatted string.

The following table describes the clipOption parameters.

Parameter	Type	Default value	Description

Parameter	Type	Default value	Description
window_clip	BOOLEAN	false	Specifies whether to use the bounding box of the geometry object to clip the raster object. Valid values: • true: uses the minimum bounding rectangle (MBR) of the geometry object. • false: uses the geometry object.
rast_coord	BOOLEAN	false	Specifies whether the geometry object to clip is in raster coordinates. If true, the x coordinate of the geometry represents the column number and the y coordinate represents the row number.

The following table describes fields in the storageOption parameters.

Parameter	Type	Default value	Description
compression	STRING	lz4	The type of the compression algorithm. Valid values: • none • jpeg • zlib • png • lzo • lz4
quality	INTEGER	75	The compression quality. This parameter takes effect only when the value of the compression parameter is set to jpeg.
interleaving	STRING	Same as the original raster object	The interleaving type. Valid values: • bip: band interleaved by pixel (BIP) • bil: band interleaved by line (BIL) • bsq: band sequential (BSQ)
endian	STRING	Same as the original raster object	The endian format. Valid values: • NDR: little endian format • XDR: big endian format

Description

The default clipping cache is 100 MB, which indicates that only 100 MB of data can be returned. To adjust the limit on the output size, you can use the ganos.raster.clip_max_buffer_size parameter to set the size of the cache.

Examples

```
Select ST_Clip(raster_obj, 0, '((128.980,30.0),(129.0,30.2))', 'World');
Select ST_Clip(raster_obj, 0, '((128.980,30.0),(129.0,30.2))', 'World', 4326);
-- Use a geometry object to clip a raster object.
-- Use the default clipping settings.
SELECT (ST_CLIP(rast, ST_geomfromtext('Polygon((0 0, 45 45, 90 45, 45 0, 0 0))', 4326), 0))
.* from clip_table where id =1
-- Use white as the background color and compress the output into a PNG image.
SELECT (ST_CLIP(rast, ST_geomfromtext('Polygon((0 0, 45 45, 90 45, 45 0, 0 0))', 4326), 0,
'', ARRAY[254,254,254], '', '{"compression":"png","interleaving":"bip"}')).* from clip_tabl
e where id =1;
-- Use the window of a geometry object to clip a raster object.
SELECT (ST_CLIP(rast, ST_geomfromtext('Polygon((0 0, 45 45, 90 45, 45 0, 0 0))', 4326), 0,
'', NULL, '{"window_clip":true}', '')).* from clip_table where id =1;
```

5.8.4. ST_ClipToRast

This function clips a raster object by using a specified geometry object and returns the clipping result as a new raster object.

Syntax

```
raster ST_ClipToRast(raster raster_obj,
                     geometry geom,
                     integer pyramidLevel default 0,
                     cstring bands default '',
                     float8[] nodata default NULL,
                     cstring clipOption default '',
                     cstring storageOption default '')
```

Parameters

Parameter	Description
raster_obj	The raster object to be clipped.
pyramidLevel	The pyramid level.
geometry	The geometry object that is used for clipping.
bands	The bands to be clipped, in the format of `'0-2'` or `'1,2,3'`. It starts from 0. Default value: `''`. The default value indicates that all the bands are clipped.
nodata	The nodata values in the format of float8[]. If the number of the values is less than the number of bands, the nodata values that are specified for the bands are filled. If no nodata value is specified for the bands, the value 0 is filled.
clipOption	The clipping option that is represented by a JSON string.
storageOption	The storage option that is represented by a JSON string in the returned result.

The following table describes the parameters of clipOption.

Parameter	Type	Default value	Description
window_clip	bool	false	Specifies whether to use the bounding box of a geometry for clipping. Valid values: • true: uses the minimum bounding rectangle (MBR) of the geometry for clipping. • false: uses the geometry object for clipping.
rast_coord	bool	false	Specifies whether the passed geometry uses pixel coordinates. If the pixel coordinates are used, the x coordinate represents the column number of the pixel and the y coordinate represents the row number of the pixel.

The following table describes the parameters of storageOption.

Parameter	Type	Default value	Description
chunking	boolean	Same as the original raster	Specifies whether to store data as chunks.
chunkdim	string	Same as the original raster	The dimension information about the chunk. This parameter only takes effect when the chunking parameter is set to true.
chunktable	string	''	The name of the chunk table. If the `''` value is passed, a temporary chunk table that has a random name is generated to store data. This temporary table is valid in only the current session. If you need to retain an accessible clipping object, you must specify the name of the chunk table.
compression	string	lz4	The type of the compression algorithm. Valid values: • none • jpeg • zlib • png • lzo • lz4
quality	integer	75	The compression quality. This parameter takes effect for only the jpeg compression algorithm.
interleaving	string	Same as the original raster	The interleaving method. Valid values: • bip: band interleaved by pixel (BIP) • bil: band interleaved by line (BIL) • bsq: band sequential (BSQ)

Parameter	Type	Default value	Description
endian	string	Same as the original raster	The endian. Valid values: • NDR: little endian • XDR: big endian

Description

- If NULL or `''` is passed for the chunktable parameter, a temporary chunk table that has a random name is generated to store data. This temporary table is valid in only the current session. If you need to retain an accessible clipping object, you must specify the name of the chunk table.
- The default cache size for clipping is 100 MB. This indicates that up to 100 MB of clipping result data can be returned. If you need to adjust the size of the returned result, you can specify the cache size by using the `ganos.raster.clip_max_buffer_size` parameter.

Examples

```
-- Create a permanent table.
CREATE TEMP TABLE rast_clip_result(id integer, rast raster);
-- Create a temporary table.
CREATE TEMP TABLE rast_clip_result_temp(id integer, rast raster);
-- Use the default clipping settings and store the result in a temporary table.
INSERT INTO rast_clip_result_temp(id, rast)
select 1, ST_ClipToRast(rast, ST_geomfromtext('Polygon((0 0, 45 45, 90 45, 45 0, 0 0))', 43
26), 0)
from clip_table
where id =1;
-- Use white as the background color for padding and store the result in a permanent table.
INSERT INTO rast_clip_result(id, rast)
select 2, ST_ClipToRast(rast, ST_geomfromtext('Polygon((0 0, 45 45, 90 45, 45 0, 0 0))', 43
26), 0, '', ARRAY[254,254,254], '', '{"chunktable":"clip_rbt"}')
from clip_table
where id =1;
-- Use the window of a geometry for clipping.
INSERT INTO rast_clip_result_temp(id, rast)
SELECT 3, ST_ClipToRast(rast, ST_geomfromtext('Polygon((0 0, 45 45, 90 45, 45 0, 0 0))', 43
26), 0, '', NULL, '{"window_clip":true}', '')
from clip_table
where id =1;
```

5.8.5. ST_Values

This function returns the geographic coordinates and values of all the cells in a raster object that intersect with a geometry object.

Syntax

```
set of record ST_Values(raster raster_obj,
                        geometry geom,
                        integer pyramidLevel default 0,
                        cstring bands default '', /* All bands */
                        cstring clipOption default '',
                        out point coords,
                        out integer band,
                        out float8 value)
```

Parameters

Parameter	Description
raster_obj	The raster object.
pyramidLevel	The pyramid level.
geometry	The geometry object used for clipping.
bands	The sequence numbers of bands to be clipped, in the format of `'0-2'` or `'1,2,3'`. The sequence number starts from 0. Default value: null string `''`. It indicates that all bands are to be clipped.
clipOption	The clipping options. The value is a JSON-formatted string.
coords	The output field that indicates the geographic coordinates of a pixel value.
band	The output field that indicates the band sequence number of a pixel value.
value	The output field that indicates a pixel value.

The following table describes the clipOption parameter.

Parameter	Type	Default value	Description
window_clip	Boolean	false	Specifies whether to use the bounding box of the geometry object to clip the raster object. Valid values: • true: uses the minimum bounding rectangle (MBR) of the geometry object. • false: uses the geometry object.

Description

- Both the geometry object and the raster object must have a valid spatial reference system identifier (SRID). Their SRIDs must be consistent.
- The default clipping cache is 100 MB, which indicates that only 100 MB of data can be returned. To adjust the limit on the output size, you can use the ganos.raster.clip_max_buffer_size parameter to set the size of the cache.

Examples

```
-- Use a geometry object of the point type to clip the raster object.
SELECT ( ST_Values(rast, ST_geomfromtext('POINT(128.135 29.774)', 4326)::geometry, 0,'{"win
dow_clip":"true"}')).*
from rat_clip WHERE id=1;
    coord      | band | value
--------------+------+-------
 (127.8,29.7) |    0 |    11
 (127.8,29.7) |    1 |    10
 (127.8,29.7) |    2 |    50
(3 rows)
-- Use a geometry object of the linestring type to clip the raster object.
SELECT ( ST_Values(rast, ST_geomfromtext('LINESTRING(0 0, 45 45, 90 0, 135 45)', 4326)::geo
metry, 0)).*
from rat_clip WHERE id=1 limit 10;
    coord     | band | value
-------------+------+-------
 (44.1,45)   |    0 |   115
 (44.1,45)   |    1 |   112
 (44.1,45)   |    2 |    69
 (45,45)     |    0 |   122
 (45,45)     |    1 |   117
 (45,45)     |    2 |    75
 (134.1,45)  |    0 |    37
 (134.1,45)  |    1 |    64
 (134.1,45)  |    2 |    13
 (43.2,44.1) |    0 |    66
(10 rows)
```

5.8.6. ST_PointValues

This topic describes the ST_PointValues function. This function is used to obtain the pixel values of all bands.

Syntax

```
setof record ST_PointValues(raster raster_obj,
        integer column_sn,
        integer row_sn,
        boolean exclude_nodata default true,
        out integer band,
        out float8 value);
setof record ST_PointValues(raster raster_obj,
        float8  x,
        float8  y,
        boolean exclude_nodata default true,
        out integer band,
        out float8 value);
```

Parameters

Parameter	Description
raster_obj	The raster object that you want to clip.
column_sn	The number of the column in which the pixel resides. The pixel value is calculated from the upper-left corner of the raster object.
row_sn	The number of the row for the pixel. The pixel value is calculated from the upper-left corner of the raster object.
x	The x coordinate of the pixel.
y	The y coordinate of the pixel.
exclude_nodata	Specifies whether to return the NoData value of the raster object. If you specify this parameter to true and the pixel value is NoData, this function does not return the NoData value of the raster object.
band	The number of the band in which the pixel resides.
value	The value of the pixel.

Description

This function returns the pixel values of all bands based on the number of the column row in which the pixel resides and the x and y coordinates of the pixel.

Examples

```
select * from
    ( select (st_pointValues(rast, 125.84382034243441 , 47.67709555783107, false)).*
      from t_pixel where id = 3) a
ORDER by band;
 band | value
------+-------
    0 |    66
    1 |    87
    2 |    28
select * from
    ( select (st_pointValues(rast, 125 , 47)).*
      from t_pixel where id = 3) a
ORDER by band;
 band | value
------+-------
    0 |    39
    1 |    66
    2 |    11
```

5.8.7. ST_Update

This function uses a source raster object to update a destination raster object.

Syntax

```
raster ST_Update(raster source, raster dest);
```

Parameters

Parameter	Description
source	The source raster object.
dest	The destination raster object.

Examples

```
Update raster_table Set raster_obj=ST_Update(raster_obj, (Select raster_obj from raster_tab
le where id=2)) where id = 1;
```

5.8.8. ST_MosaicFrom

This topic describes the ST_MosaicFrom function, which performs a mosaic operation to combine multiple raster objects into a new raster object.

Syntax

```
raster ST_MosaicFrom(raster source[],  cstring chunkTableName);
```

Parameters

Parameter	Description
source	The name of the raster objects that you want to combine.
chunkTableName	The name of the chunk table for storing the new raster object. The name must comply with the table naming conventions of ApsaraDB RDS for PostgreSQL.

Description

This function combines multiple raster objects into a new one.

All of the raster objects that you want to combine must meet the following requirements:

- They have the same number of bands.
- They are all geographically referenced, or none of them is geographically referenced. If all of them are geographically referenced, world coordinates (geographic coordinates) are used for the mosaic operation.
- Although they can have different pixel types, they must have the same spatial reference system identifier (SRID) and affine parameters if world coordinates are used for the mosaic operation.

You must configure the following parameter.

Parameter	Type	Description
ganos.raster.mosaic_must_same_nodata	boolean	Specifies whether the values of NoData in a data source must be the same during the mosaic operation. Valid values: true and false. The values of NoData are not changed during the mosaic operation. If you set this parameter to false, the semantics of the pixels after the mosaic operation may be changed. Example: `Set ganos.raster.mosaic_must_same_nodata = false;`

Example

```
INSERT INTO raster_table VALUES(1, ST_MosaicFrom(Array(SELECT raster_obj FROM raster_table
WHERE id < 10), 'chunk_table_mosaic'))
UPDATE raster_table SET raster_obj = ST_MosaicFrom(Array(SELECT raster_obj FROM raster_tabl
e WHERE id < 10), 'chunk_table_mosaic') WHERE id = 11;
```

5.8.9. ST_MosaicTo

This topic describes the ST_MosaicTo function, which performs a mosaic operation to integrate a raster object or object set into another raster object.

Syntax

```
raster ST_MosaicTo(raster raster_obj,raster source[]);
```

Parameters

Parameter	Description
raster_obj	The raster object into which you want to integrate another raster object or object set.
source	The raster object or object set that you want to integrate it into another raster object or object set.

Description

The source and destination raster objects must meet the following requirements:

- They have the same number of bands.
- They are all geographically referenced, or none of them is geographically referenced. If all of them are geographically referenced, world coordinates (geographic coordinates) are used for the mosaic operation.
- Although they can have different pixel types, they must have the same spatial reference system identifier (SRID) and affine parameters if world coordinates are used for the mosaic operation.

You must configure the following parameter.

Parameter	Type	Description
ganos.raster.mosaic_must_same_nodata	boolean	Specifies whether the values of NoData in a data source must be the same during the mosaic operation. Valid values: true and false. The values of NoData are not changed during the mosaic operation. If you set this parameter to false, the semantics of the pixels after the mosaic operation may be changed. Example: `Set ganos.raster.mosaic_must_same_nodata = false;`

Example

```
Update raster_table Set raster_obj = ST_MosaicTo(raster_obj, Array(select raster_obj from r
aster_table where id < 10)) where id = 11;
```

5.8.10. ST_PixelAsPolygon

This function is used to convert a specified pixel of a raster into a polygon object.

Syntax

```
geometry ST_PixelAsPolygon(raster raster_obj,
                           integer row default 0,
                           integer column default 0
                           integer pyramid default 0) ;
```

Parameters

Parameter	Description
raster_obj	The raster that you want to query.
row	The number of the row in which the pixel resides. Valid row numbers start from 0, which indicates the pixel in the upper-left corner of the raster.
column	The number of the column in which the pixel resides. Valid column numbers start from 0, which indicates the pixel in the upper-left corner of the raster.
pyramid	The pyramid level in which the pixel resides.

Examples

```
SELECT ST_AsText(ST_PixelAsPolygon(rast, 10, 10))
FROM rast_table
WHERE id = 10;
------------------
POLYGON((3427927.75 5793244,3427927.8 5793244,3427927.8 5793243.95,...
```

5.8.11. ST_PixelAsPolygons

This function is used to convert the pixels at a specified position of a raster into polygon objects.

Syntax

```
setof record ST_PixelAsPolygons(raster raster_obj,
    integer band default  0,
    integer pyramid default 0,
    boolean exclude_nodata_value default true,
    out integer rowsn,
    out integer columnsn,
    out integer bandsn,
    out double value,
    out geometry geom);
```

Parameters

Parameter	Description
raster_obj	The raster that you want to query.
band	The IDs of the bands that you want to query. Valid band IDs start from 0.
pyramid	The ID of the pyramid level that you want to query.
exclude_nodata_value	Specifies whether to exclude NoData pixels.
rowsn	The number of the row in which the pixel resides.
columnsn	The number of the column in which the pixel resides.
bandsn	The ID of the band to which the pixel belongs.
value	The value of the pixel.
geom	The spatial extent of the pixel.

Examples

```
WITH tmp AS (
SELECT (ST_PixelAsPolygons(rast, 1)).*
FROM rast_table
WHERE id = 10 )
SELECT rowsn, columnsn, bandsn, value, ST_AsEWKT(geom)  FROM tmp WHERE value > 38.0;
----------------------------------------------------
    100 |      100 |      0 |    43 | SRID=4326;POLYGON((-180 90,-180 89.1,-179.1 89.1,-179
.1 90,-180 90))
```

5.8.12. ST_PixelAsPoint

This function is used to convert a specified pixel of a raster into a geometric object that is represented by a specified point in the geometric object.

Syntax

```
geometry ST_PixelAsPoint(raster raster_obj,
    integer row default 0,
    integer column default 0,
    integer pyramid default 0,
    cstring ref_point default 'UPPERLEFT') ;
```

Parameters

Parameter	Description
raster_obj	The raster that you want to query.
row	The number of the row in which the pixel resides. Valid row numbers start from 0, which indicates the pixel in the upper-left corner of the raster.
column	The number of the column in which the pixel resides. Valid column numbers start from 0, which indicates the pixel in the upper-left corner of the raster.
pyramid	The pyramid level in which the pixel resides.
ref_point	The reference position of the point in the geometric object. Valid values: • UPPERLEFT: specifies the point that resides in the upper-left corner of the geometric object. • CENTER: specifies the point that resides in the center of the geometric object.

Description

This function converts a specified pixel of a raster into a geometric object that is represented by a specified point in the geometric object. The reference position of the point is specified by the ref_point parameter.

Examples

```
SELECT ST_AsText(ST_PixelAsPoint(rast, 10, 10))
FROM rast_table
WHERE id = 10;
------------------
POINT(-178.2 88.2)
```

5.8.13. ST_PixelAsCentroid

This function is used to convert a specified pixel of a raster into a geometric object that is represented by the point in the center of the geometric object.

Syntax

```
geometry ST_PixelAsPoint(raster raster_obj,
    integer row default 0,
    integer column default 0,
    integer pyramid default 0) ;
```

Parameters

Parameter	Description
raster_obj	The raster that you want to query.
row	The number of the row in which the pixel resides. Valid row numbers start from 0, which indicates the pixel in the upper-left corner of the raster.
column	The number of the column in which the pixel resides. Valid column numbers start from 0, which indicates the pixel in the upper-left corner of the raster.
pyramid	The pyramid level in which the pixel resides.

Description

This function converts a specified pixel of a raster into a geometric object that is represented by the point in the center of the geometric object. You can obtain the same result for a pixel if you set the ref_point parameter of the ST_PixelAsPoint function to **CENTER**.

Examples

```
SELECT ST_AsText(ST_PixelAsCentroid(rast, 10, 10))
FROM rast_table
WHERE id = 10;
------------------
POINT(-178.2 88.2)
```

5.8.14. ST_PixelAsPoints

This function is used to convert the pixels at a specified position of a raster into geometric objects, each of which is represented by a specified point of the geometric object.

Syntax

```
setof record ST_PixelAsPoints(raster raster_obj,
    integer band default  0,
    integer pyramid default 0,
    boolean exclude_nodata_value default true,
    cstring ref_point default 'UPPERLEFT',
    out integer rowsn,
    out integer columnsn,
    out integer bandsn,
    out double value,
    out geometry geom);
```

Parameters

Parameter	Description
raster_obj	The raster that you want to query.
band	The IDs of the bands that you want to query. Valid band IDs start from 0.
pyramid	The ID of the pyramid level that you want to query.
exclude_nodata_value	Specifies whether to exclude NoData pixels.
ref_point	The reference position of the point in the geometric object. Valid values: • UPPERLEFT: specifies the point that resides in the upper-left corner of the geometric object. • CENTER: specifies the point that resides in the center of the geometric object.
rowsn	The number of the row in which the pixel resides.
columnsn	The number of the column in which the pixel resides.
bandsn	The ID of the band to which the pixel belongs.
value	The value of the pixel.
geom	The spatial extent of the pixel.

Description

This function converts the pixels at a specified position of a raster into geometric objects, each of which is represented by a specified point of the geometric object. The reference position of the point is specified by the ref_point parameter.

Examples

```
WITH tmp AS (
SELECT (ST_PixelAsPoints(rast, 1)).*
FROM rast_table
WHERE id = 10 )
SELECT rowsn, columnsn, bandsn, value, ST_AsEWKT(geom)  FROM tmp WHERE value > 38.0;
----------------------------------------------------
    100 |      100 |      0 |    43 | SRID=4326;POINT(-90 0)
```

5.8.15. ST_PixelAsCentroids

This function is used to convert the pixels at a specified position of a raster into geometric objects, each of which is represented by the point in the center of the geometric object.

Syntax

```
setof record ST_PixelAsCentroids(raster raster_obj,
    integer band default  0,
    integer pyramid default 0,
    boolean exclude_nodata_value default true,
    out integer rowsn,
    out integer columnsn,
    out integer bandsn,
    out double value,
    out geometry geom);
```

Parameters

Parameter	Description
raster_obj	The raster that you want to query.
band	The IDs of the bands that you want to query. Valid band IDs start from 0.
pyramid	The ID of the pyramid level that you want to query.
exclude_nodata_value	Specifies whether to exclude NoData pixels.
rowsn	The number of the row in which the pixel resides.
columnsn	The number of the column in which the pixel resides.
bandsn	The ID of the band to which the pixel belongs.
value	The value of the pixel.
geom	The spatial extent of the pixel.

Description

This function converts the pixels at a specified position of a raster into geometric objects, each of which is represented by the point in the center of the geometric object. You can obtain the same results for the specified pixels if you set the ref_point parameter of the ST_PixelAsPoints function to **CENTER**.

Examples

```
WITH tmp AS (
SELECT (ST_PixelAsCentroids(rast, 1)).*
FROM rast_table
WHERE id = 10 )
SELECT rowsn, columnsn, bandsn, value, ST_AsEWKT(geom)  FROM tmp WHERE value > 38.0;
----------------------------------------------------
    100 |      100 |      0 |    43 | SRID=4326;POINT(-89.5 0.5)
```

5.9. Overview operations

5.9.1. ST_BuildOverview

This function creates an overview.

Syntax

```
raster ST_BuildOverview(cstring srcTableName, cstring srcColumnName, integer srcPyramidLevel, cstring chunkTableName);
```

Parameters

Parameter	Description
tableName	The name of the table.
columnName	The name of the raster object column.
pyramidLevel	The pyramid level of the overview to be created.
chunkTableName	The name of the chunk table for storing the raster object created as an overview.

This function creates a table-based raster overview.

All the raster objects involved must meet the following requirements:

- They have the same number of bands.
- Either all or one of raster objects are geographically referenced. If all of them are geographically referenced, world coordinates (geographic coordinates) are used for the mosaic operation.
- Their pixel types can be different. If world coordinates are used for the mosaic operation, they must have the same spatial reference system identifier (SRID) and pixel resolution.

Example

```
Insert into raster_table_overview values(1, ST_BuildOverview('raster_table','raster_obj', 0,'chunk_table_overview'));
```

5.9.2. ST_UpdateOverview

This function uses a raster object or object set to update an overview.

Syntax

```
raster ST_UpdateOverview(raster raster_obj,raster source[]);
```

Parameters

Parameter	Description
raster_obj	The destination raster object.
source	The source raster object or object set.

Description

All the specified raster objects must meet the following requirements:

- They have the same number of bands.
- Either all of them are geographically referenced, or none of them is geographically referenced. If all of them are geographically referenced, world coordinates (geographic coordinates) are used for the mosaic operation.
- Their pixel types can be different. If world coordinates are used for the mosaic operation, they must have the same spatial reference system identifier (SRID) and pixel resolution.

Examples

```
Update raster_table set raster_obj = ST_UpdateOverview(raster_obj, Array(select raster_obj
from raster_table_new)) where id = 1;
```

5.9.3. ST_EraseOverview

This function clears the specified area of a raster object.

Syntax

```
raster ST_EraseOverview(raster raster_obj, Box extent, BoxType type, boolean useNodata);
```

Parameters

Parameter	Description
raster_obj	The raster object.
extent	The area to be cleared, in the format of `'((minX,minY),(maxX,maxY))'`.
type	The coordinate type of the area to be cleared. The value must be Raster or World, where Raster indicates pixel coordinates and World indicates world coordinates.
useNodata	Indicates whether to use the predefined NoData value to fill the area to be cleared. If this parameter is set to false or the NoData value is not defined, 0 is used to fill the area.

Examples

```
Select ST_EraseOverview(raster_obj, '((0,0),(100,100))', 'Raster', false) from raster_table
where id=100;
```

5.10. DEM operations

5.10.1. ST_Aspect

This function calculates the aspect from each cell on the surface of a raster object and returns an array of aspects.

Syntax

```
raster ST_Aspect(raster rast, integer pyramid_level, integer band, Box extent, BoxType type
, cstring storageOption);
```

Parameters

Parameter	Description
rast	The raster object.
pyramid_level	The pyramid level.
Band	The sequence number of the band.
box	The area that you want to analyze. Specify the value in the `'((minX,minY),(maxX,maxY))'` format.
type	The type of coordinate that is used to identify the area to be analyzed. You can specify only one value. Valid values: • Raster: pixel coordinates • World: world coordinates
storageOption	The storage option of the raster object. For more information, see ST_ClipToRast.

Description

The ST_Aspect function is used to calculate the aspect from the surface of a raster object. The aspect identifies the downslope direction of the maximum rate of change in a value from each cell to neighboring cells. The aspect can be considered the slope direction. The value of each cell in the output raster object indicates the compass direction of the aspect. The aspect is measured clockwise from the north. Unit: degrees. Valid values: 0 to 360. Cells in the input raster object that are flat and do not have a downslope direction are assigned an aspect of -1 in the output raster object.

The value of a cell in the aspect dataset of the output raster object indicates the slope direction of the cell.

Example:

```
select st_aspect(rast, 0, 0, '(0,0), (5,5)', 'Raster') from t_surface where id=1;
```

5.10.2. ST_Slope

This function calculates the slope from each cell over the surface of a raster object and returns an array of slopes that are expressed in radians.

Syntax

```
raster ST_Slope(raster rast, integer pyramid_level, integer band, Box extent, BoxType type,
float8 zfactor, cstring storageOption);
```

Parameters

Parameter	Description
rast	The raster object.
pyramid_level	The pyramid level.
Band	The sequence number of the band.
box	The area that you want to analyze. Specify the value in the `'((minX,minY),(maxX,maxY))'` format.
type	The type of coordinate that is used to identify the area to be analyzed. You can specify only one value. Valid values: • Raster: pixel coordinates • World: world coordinates
zfactor	The conversion factor that is used to adjust the unit of measurement for the elevation units when the elevation units are different from the horizontal coordinates of the input surface. Default value: 1.
storageOption	The storage option of the raster object. For more information,see ST_ClipToRast.

Description

The ST_Slope function calculates the maximum rate of change in a value from each cell to neighboring cells. The maximum change in elevation over the distance between a cell and eight neighboring cells identifies the steepest downhill descent from the cell.

Example:

```
select st_slope(rast, 0, 0, '(0,0), (5,5)', 'Raster', 2.0) from t_surface where id=1;
```

5.10.3. ST_Hillshade

This function calculates the hillshade from each cell over the surface of a raster object and returns an array of hillshades.

Syntax

```
raster ST_Hillshade(raster rast, integer pyramid_level, integer band, Box extent, BoxType type, float8 zfactor, float8 azimuth, float8 altitude, cstring storageOption);
```

Parameters

Parameter	Description
rast	The raster object.
pyramid_level	The pyramid level.
band	The sequence number of the band.
extent	The area that you want to analyze. Specify the value in the `'((minX,minY),(maxX,maxY))'` format.
type	The type of coordinate that is used to identify the area to be analyzed. You can specify only one value. Valid values: • Raster: pixel coordinates • World: world coordinates
zfactor	The conversion factor that is used to adjust the unit of measurement for the elevation units when the elevation units are different from the horizontal coordinates of the input surface. Default value: 1.
azimuth	The azimuth angle of the sun. The azimuth angle is measured clockwise from the north. Unit: degrees. Valid values: 0 to 360. Default value: 315 (northwest).
altitude	The altitude angle of the sun above the horizon. Unit: degrees. Valid values: 0 to 90. A value of 90 indicates that the sun is directly overhead.
storageOption	The storage option of the raster object. For more information, see ST_ClipToRast.

Description

The ST_Hillshade function obtains the hypothetical illumination of the surface of a raster object by determining the illumination value from each cell of the raster object. This function assigns a position for a hypothetical light source and calculates the illumination value of each cell in relation to neighboring cells. This function can greatly enhance the visualization of a raster surface for analysis or graphical display, especially when transparency is used.

By default, shadow and light are displayed as shades in gray that are associated with integers from 0 to 255. The darkest shade is represented by 0, and the brightest shade is represented by 255.

Example:

```
select st_hillshade(rast, 0, 0, '(0,0), (5,5)', 'Raster', 4, 180, 80) from t_surface where id=1;
```

5.10.4. ST_Overflow

This function computes the water depth in an area based on the geographic coordinates and precipitation of the area.

Syntax

```
float8[] ST_Overflow(raster rast, Box box, float8 value);
```

Parameters

Parameter	Description
rast	The raster object, which currently must be a digital elevation model (DEM) that has only one band.
box	The world space of the area to be analyzed, which must be consistent with that of the DEM.
value	The precipitation of the area. Unit: cubic meters.

Description

This function computes the water depth in an area based on the geographic coordinates and precipitation of the area. Currently, the raster object must be a DEM that has only one band.

Examples

```
select st_overflow(rast, '((-195516.0,2230052.0),(-194626.0,2230942.0))'::box, 10000::float
8) from t_overflow where id =1;

st_overflow
-----------------------------------------------------------------------------------------
{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1.43427103409668,0,0.434271034096678,0,0,0,0,0,0,0
,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0.434271034096678,0,0.434271034096678,0,0.434271034096
678,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0.434271034096678,0.434271034096678,0
.434271034096678,0.434271034096678,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0.434271034096678,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0.4
34271034096678,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,1.43427103409668,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0.434271034096678,0,0,0,0,0}
(1 row)
```

5.10.5. ST_Flow_direction

This function computes the flow direction in an area based on the digital elevation model (DEM) data in the area.

Syntax

```
float8[] ST_flow_direction(raster rast, Box box);
```

Parameters

Parameter	Description
rast	The raster object, which currently must be a digital elevation model (DEM) that has only one band.
box	The world space of the area to be analyzed.

Description

This function computes the flow direction in an area based on the DEM data in the area by using the D8 flow direction algorithm.

The raster object must be a DEM value that has only one band.

Example

```
select st_flow_direction(rast, '((-202286.94,2232375.16),(-202135.0,2232225))'::box) from t
_overflow where id =2;
                              st_flow_direction
--------------------------------------------------------------------------------
 {2,2,2,4,4,8,2,2,2,4,4,8,1,1,2,4,8,4,128,128,1,2,4,8,2,2,1,4,4,4,1,1,1,1,4,16}
(1 row)
```

5.11. Attribute query and update

5.11.1. ST_Name

This function returns the name of a raster object. If no name is specified, the function returns null.

Syntax

```
text ST_Name(raster rast);
```

Parameters

Parameter	Description
rast	The raster object.

Examples

```
select ST_Name(rast) from rat where id=1;
________________________________
image1
```

5.11.2. ST_SetName

This function sets the name of a raster object.

Syntax

```
raster ST_SetName(raster rast, cstring name);
```

Parameters

Parameter	Description
rast	The raster object.
name	The name of the raster object.

Examples

```
update rat set rast = ST_SetName(rast,'image2') where id = 2;
----------------------------------------------------------------
(1 row)
```

5.11.3. ST_MetaData

This function returns the metadata of a raster object in JSON format.

Syntax

```
text ST_MetaData(raster raster_obj);
text ST_MetaData(raster raster_obj,
                 text key);
text ST_MetaData(raster raster_obj,
                 integer band,
                 text key);
```

Parameters

Parameter	Description
raster_obj	The raster object.
band	The sequence number of the band, which starts from 0.
key	The name of the metadata item that you want to query. If you set this parameter to `all`, all metadata items are retutrned in JSON format.

Examples

```
select ST_MetaData(raster_obj) from raster_table;
-- meta data with a name
SELECT ST_MetaData(raster_obj, 'swh#scale_factor')
FROM raster_table;
      st_metadata
-----------------------
 0.0001488117874873806
-- all meta data
SELECT ST_MetaData(raster_obj, 'all')
FROM raster_table;
      st_metadata
-----------------------
{"AREA_OR_POINT":"Area"}
-- meta data with a name of a band
SELECT ST_MetaData(raster_obj, 0, 'NETCDF_DIM_time')
FROM raster_table;
 st_metadata
-------------
 1043112
 -- all meta data of a band
SELECT ST_MetaData(raster_obj, 'all')
FROM raster_table;
                                                                st_metadata

-----------------------------------------------------------------------------------------------
---------------------------------------------
{"add_offset":"4.907141431495487","long_name":"Significant height of combined wind waves an
d swell","missing_value":"-32767","NETCDF_DI.
M_time":"1043112","NETCDF_VARNAME":"swh","scale_factor":"0.0001488117874873806","units":"m"
,"_FillValue":"-32767"}
```

5.11.4. ST_Width

This topic describes the ST_Width function, which obtains the width of a raster. For more information about how to obtain the width of a chunk, see the ST_ChuckWidth function.

Syntax

```
integer ST_Width(raster raster_obj);
integer ST_Width(raster raster_obj, integer pyramid);
```

Parameters

Parameter	Description
raster_obj	The raster whose width you want to obtain.
pyramid	The pyramid level at which the raster resides. Valid values start from 0.

Example

```
select ST_Width(raster_obj) from raster_table;
-----------------------------------------------
10060
```

5.11.5. ST_Height

This topic describes the ST_Height function, which obtains the height of a raster.

Syntax

```
integer ST_Height(raster raster_obj);
integer ST_Height(raster raster_obj, integer pyramid)
```

Parameters

Parameter	Description
raster_obj	The raster whose height you want to obtain.
pyramid	The pyramid level at which the raster resides. Valid values start from 0.

Example

```
select ST_Height(raster_obj) from raster_table;
```

5.11.6. ST_NumBands

This function returns the number of bands in a raster object.

Syntax

```
integer ST_NumBands(raster raster_obj);
```

Parameters

Parameter	Description
raster_obj	The raster object.

Examples

```
select ST_NumBands(raster_obj) from raster_table;
-------------------------------------
3
```

5.11.7. ST_Value

This function returns the value of a cell based on the specified band, row number, and column number.

Syntax

```
float8 ST_Value(raster rast, integer band, integer colsn, integer rowsn, boolean exclude_no
data_value);
```

Parameters

Parameter	Description
rast	The raster object.
band	The sequence number of the band.
colsn	The column number of the cell.
rowsn	The row number of the cell.
exclude_nodata_value	Specifies whether to exclude the NoData value. Default value: true.

Description

This function returns the value of a cell based on the specified band, row number, and column number. The band sequence number starts from 0. The default band sequence number, row number, and column number are all 0.

Examples

```
select st_value(rast, 1, 3, 4) from t_pixel where id=2;
 st_value
----------
       88
(1 row)
```

5.11.8. ST_RasterID

This function returns the universally unique identifier (UUID) of a raster object.

Syntax

```
text ST_RasterID(raster raster_obj);
```

Parameters

Parameter	Description
raster_obj	The raster object.

Examples

```
select ST_RasterID(raster_obj) from raster_table;
-----------------------------------
4e692ed0-74e2-42a3-a10d-c28d4ae31982
```

5.11.9. ST_CellDepth

This function returns the pixel depth of a raster object. The pixel depth can be 0, 1, 2, 4, 8, 16, 32, or 64, where 0 indicates that the pixel depth is unknown.

Syntax

```
integer ST_CellDepth(raster raster_obj);
```

Parameters

Parameter	Description
raster_obj	The raster object.

Examples

```
select ST_CellDepth(raster_obj) from raster_table;
__________________________________
8
```

5.11.10. ST_CellType

This function returns the pixel type of a raster object. The pixel type can be 8BSI, 8BUI, 16BSI, 16BUI, 32BSI, 32BUI, 32BF, or 64BF.

Syntax

```
text ST_CellType(raster raster_obj);
```

Parameters

Parameter	Description
raster_obj	The raster object.

Examples

```
select st_celltype(raster_obj) from raster_table;
____________________________________
8BUI
```

5.11.11. ST_InterleavingType

This function returns the interleaving type of a raster object. The interleaving type can be BSQ, BIL, or BIP.

Syntax

```
text ST_InterleavingType(raster raster_obj);
```

Parameters

Parameter	Description
raster_obj	The raster object.

Examples

```
select ST_InterleavingType(raster_obj) from raster_table;
____________________________________
BSQ
```

5.11.12. ST_TopPyramidLevel

This function returns the highest pyramid level of a raster object.

Syntax

```
integer TopPyramidLevel(raster raster_obj);
```

Parameters

Parameter	Description
raster_obj	The raster object.

Examples

```
select ST_TopPyramidLevel(raster_obj) from raster_table;
____________________________________
6
```

5.11.13. ST_Extent

This topic describes the ST_Extent function, which obtains the coordinate range of a raster in the following format: `'((maxX,maxY),(minX,minY))'`.

Syntax

```
BOX ST_Extent(raster raster_obj,CoorSpatialOption csOption = 'WorldFirst')
BOX ST_Extent(raster raster_obj, integer pyramid, CoorSpatialOption csOption = 'WorldFirst'
)
```

Parameters

Parameter	Description
raster_obj	The raster whose coordinate range you want to obtain.
csOption	The type of coordinate to return.
pyramid	The pyramid level at which the raster resides. Valid values start from 0.

Description

This function obtains the coordinate range of a raster based on the value of the csOption parameter:

- Raster: This function returns pixel coordinates.
- World: This function returns world coordinates.
- WorldFirst: This function preferentially returns world coordinates. If the raster is georeferenced, this function returns world coordinates. Otherwise, this function returns pixel coordinates.

Example

```
select ST_Extent(raster_obj, 'Raster'::CoorSpatialOption) from raster_table;
_________________________________
((255, 255), (0, 0))
```

5.11.14. ST_ConvexHull

This topic describes the ST_ConvexHull function, which obtains the minimum convex geometry that encloses all geometries within a raster.

Syntax

```
geometry ST_ConvexHull(raster source);
geometry ST_ConvexHull(raster source,
                       integer pyramid);
```

Parameters

Parameter	Description
source	The raster whose minimum convex geometry you want to obtain.
pyramid	The pyramid level at which the raster resides. Valid values start from 0. Default value: 0.

Description

This function obtains the minimum convex geometry that encloses all geometries within a raster.

Example

```
SELECT st_astext(st_convexhull(rast_object))
FROM raster_table;
----------------------------------------------------
 POLYGON((-180 90,180 90,180 -90,-180 -90,-180 90))
--Specify the pyramid level at which the raster resides.
SELECT st_astext(st_convexhull(rast_object,  1))
FROM raster_table;
----------------------------------------------------
 POLYGON((-180 90,180 90,180 -90,-180 -90,-180 90))
```

5.11.15. ST_Envelope

This topic describes the ST_Envelope function, which obtains the bounding box of a raster.

Syntax

```
geometry ST_Envelope(raster source);
geometry ST_Envelope(raster source,
                     integer pyramid);
```

Parameters

Parameter	Description
source	The raster whose bounding box you want to obtain.
pyramid	The pyramid level at which the raster resides. Valid values start from 0. Default value: 0.

Description

This function obtains the bounding box of a raster.

Example

```
SELECT st_astext(st_envelope(rast_object))
FROM raster_table;
-----------------------------------------------------
 POLYGON((-180 90,180 90,180 -90,-180 -90,-180 90))
--Specify the pyramid level at which the raster resides.
SELECT st_astext(st_envelope(rast_object,  1))
FROM raster_table;
-----------------------------------------------------
 POLYGON((-180 90,180 90,180 -90,-180 -90,-180 90))
```

5.11.16. ST_Srid

This function returns the spatial reference system identifier (SRID) of a raster object. The SRID and its definition are stored in the spatial_ref_sys table.

Syntax

```
integer ST_Srid(raster raster_obj);
```

Parameters

Parameter	Description
raster_obj	The raster object.

Examples

```
select ST_Srid(raster_obj) from raster_table where id=1;
____________________________________
4326
```

5.11.17. ST_SetSrid

This function sets the spatial reference system identifier (SRID) of a raster object. The SRID and its definition are stored in the spatial_ref_sys table.

Syntax

```
raster ST_SetSrid(raster rast, integer srid);
```

Parameters

Parameter	Description
rast	The raster object.
srid	The SRID of the raster object.

Description

If the raster object is stored in external mode, you must make sure that the raster object is geographically referenced and the specified SRID can be found in the spatial_ref_sys table. Otherwise, the information about the raster object cannot be updated.

Examples

```
update rast set rast=ST_SetSrid(rast,4326) where id=1;
____________________________________
(1 row)
```

5.11.18. ST_ScaleX

Queries the pixel width of a raster object on the X scale in the spatial reference system.

Prerequisites

The raster object has a valid spatial reference system identifier (SRID).

Syntax

```
float8 ST_ScaleX(raster raster_obj)
```

Parameters

Parameter	Description
raster_obj	The name of the raster object.

Examples

```
select st_scalex(rast), st_scaley(rast)
from raster_table;
 st_scalex | st_scaley
-----------+-----------
       1.3 |       0.3
```

5.11.19. ST_ScaleY

Queries the pixel width of a raster object on the Y scale in the spatial reference system.

Prerequisites

The raster object has a valid spatial reference system identifier (SRID).

Syntax

```
float8 ST_ScaleY(raster raster_obj)
```

Parameters

Parameter	Description
raster_obj	The name of the raster object.

Examples

```
select st_scalex(rast), st_scaley(rast)
from raster_table;
 st_scalex | st_scaley
-----------+-----------
       1.3 |       0.3
```

5.11.20. ST_SetScale

Configures the pixel widths of a raster object on the X and Y scales in the spatial reference system.

Syntax

```
raster ST_SetScale(raster raster_obj,  float8 scaleX, float8 scaleY)
raster ST_SetScale(raster raster_obj,  float8 scaleXY)
```

Parameters

Parameter	Description
raster_obj	The name of the raster object.
scaleX	The pixel width of the raster object on the X scale in the spatial reference system.
scaleY	The pixel width of the raster object on the Y scale in the spatial reference system.
scaleXY	The pixel width of the raster object on the X and Y scales in the spatial reference system. This parameter specifies the same pixel width on the X and Y scales.

Examples

```
UPDATE raster_table
SET rast = ST_SetScale(rast, 1.3, 0.6)
WHERE id = 2;
select st_scalex(rast), st_scaley(rast)
from raster_table
WHERE id = 2;
 st_scalex | st_scaley
-----------+-----------
       1.3 |       0.6
```

5.11.21. ST_SkewX

Queries the skew of a raster object on the X scale in the spatial reference system.

Prerequisites

The raster object has a valid spatial reference system identifier (SRID).

Syntax

```
float8 ST_SkewX(raster raster_obj)
```

Parameters

Parameter	Description
raster_obj	The name of the raster object.

Examples

```
select st_skewx(rast), st_skewy(rast)
from raster_table;
 st_skewx | st_skewy
----------+----------
      0.3 |      0.2
```

5.11.22. ST_SkewY

Queries the skew of a raster object on the Y scale in the spatial reference system.

Prerequisites

The raster object has a valid spatial reference system identifier (SRID).

Syntax

```
float8 ST_SkewY(raster raster_obj)
```

Parameters

Parameter	Description
raster_obj	The name of the raster object.

Examples

```
select st_skewx(rast), st_skewy(rast)
from raster_table;
 st_skewx | st_skewy
----------+----------
      0.3 |      0.2
```

5.11.23. ST_SetSkew

Configures the skews of a raster object on the X and Y scales in the spatial reference system.

Syntax

```
raster ST_SetSkew(raster raster_obj,  float8 skewX, float8 skewY)
raster ST_SetSkew(raster raster_obj,  float8 skewXY)
```

Parameters

Parameter	Description
raster_obj	The name of the raster object.
skewX	The skew of the raster object on the X scale.

Parameter	Description
skewY	The skew of the raster object on the Y scale.
skewXY	The skew of the raster object on the X and Y scales. This parameter specifies the same skew on the X and Y scales.

Examples

```
UPDATE raster_table
SET rast = ST_SetSkew(rast, 0.3, 0.6)
WHERE id = 2;
select st_skewx(rast), st_skewy(rast)
from raster_table
WHERE id = 2;
 st_skewx | st_skewy
----------+----------
      0.3 |      0.6
```

5.11.24. ST_UpperLeftX

Queries the upper-left X coordinate of a raster object in the spatial reference system.

Prerequisites

The raster object has a valid spatial reference system identifier (SRID).

Syntax

```
float8 ST_UpperLeftX(raster raster_obj)
```

Parameters

Parameter	Description
raster_obj	The name of the raster object.

Examples

```
select st_upperleftx(rast), st_upperlefty(rast)
from raster_table;
 st_upperleftx | st_upperlefty
---------------+---------------
        440720 |       3751320
```

5.11.25. ST_UpperLeftY

Queries the upper-left Y coordinate of a raster object in the spatial reference system.

Prerequisites

The raster object has a valid spatial reference system identifier (SRID).

Syntax

```
float8 ST_UpperLeftY(raster raster_obj)
```

Parameters

Parameter	Description
raster_obj	The name of the raster object.

Examples

```
select st_upperleftx(rast), st_upperlefty(rast)
from raster_table;
 st_upperleftx | st_upperlefty
---------------+---------------
        440720 |       3751320
```

5.11.26. ST_SetUpperLeft

Configures the upper-left X and Y coordinates of a raster object in the spatial reference system.

Syntax

```
raster ST_SetUpperLeft(raster raster_obj,  float8 upperleftX, float8 upperleftY)
raster ST_SetUpperLeft(raster raster_obj,  float8 upperleftXY)
```

Parameters

Parameter	Description
raster_obj	The name of the raster object.
upperleftX	The upper-left X coordinate of the raster object in the spatial reference system.
upperleftY	The upper-left Y coordinate of the raster object in the spatial reference system.
upperleftXY	The upper-left X and Y coordinates of the raster object in the spatial reference system. This parameter specifies the same upper-left X and Y coordinates.

Examples

```
UPDATE raster_table
SET rast = ST_SetUpperleft(rast, 120, 30)
WHERE id = 2;
select st_upperleftx(rast), st_upperlefty(rast)
from raster_table
WHERE id = 2;
 st_upperleftx | st_upperlefty
---------------+---------------
           120  |            30
```

5.11.27. ST_PixelWidth

Queries the pixel width of a raster object in the spatial reference system.

Syntax

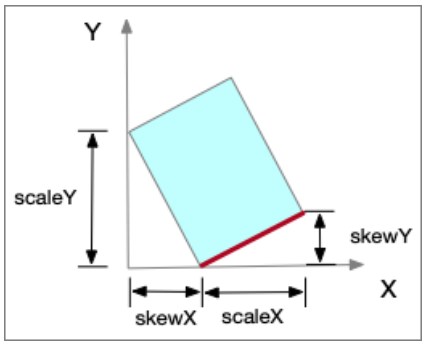
```
float8 ST_PixelWidth(raster raster_obj)
```

Parameters

Parameter	Description
raster_obj	The name of the raster object.

Description

The red part in the following figure shows the pixel width of the raster object. If the skew of the raster object is 0, the pixel width is equal to the X scale.



Examples

```
select st_pixelwidth(rast), st_pixelheight(rast)
from raster_table;
 st_pixelwidth | st_pixelheight
---------------+----------------
             60 |             60
```

5.11.28. ST_PixelHeight

Queries the pixel height of a raster object in the spatial reference system.

Syntax

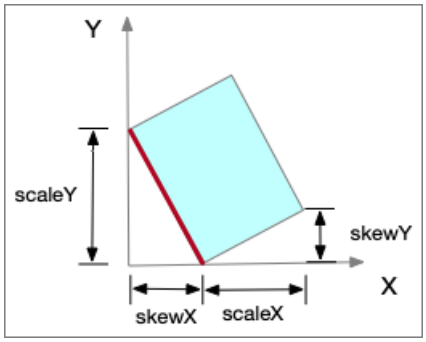
```
float8 ST_PixelHeight(raster raster_obj)
```

Parameters

Parameter	Description
raster_obj	The name of the raster object.

Description

The red part in the following figure shows the pixel height of the raster object. If the skew of the raster object is 0, the pixel height is equal to the Y scale.



Examples

```
select st_pixelwidth(rast), st_pixelheight(rast)
from raster_table;
 st_pixelwidth | st_pixelheight
---------------+----------------
            60 |             60
```

5.11.29. ST_Georeference

This function returns the geographic reference information about a raster object. Affine parameters in the text format of "A,B,C,D,E,F" are returned.

Syntax

```
text ST_Georeference(raster raster_obj);
```

Parameters

Parameter	Description
raster_obj	The raster object.

Examples

```
select ST_Georeference(raster_obj) from raster_table where id=1;
----------------------------------
2.500000000000000,0.000000000000000,38604686.750000000000000,0.000000000000000,-2.500000000
000000,4573895.750000000000000
```

5.11.30. ST_IsGeoreferenced

This function returns true if a raster object is geographically referenced. The return value is t or f in Boolean format.

Syntax

```
boolean ST_IsGeoreferenced(raster raster_obj);
```

Parameters

Parameter	Description
raster_obj	The raster object.

Description

In the returned result, t indicates true and f indicates false.

Examples

```
select ST_IsGeoreferenced(raster_obj) from raster_table where id=1;
----------------------------------
t
```

5.11.31. ST_UnGeoreference

This function deletes the geographic reference information about a raster object.

Syntax

```
raster ST_UnGeoreference(raster rast);
```

Parameters

Parameter	Description
rast	The raster object.

Examples

```
update rast set rast=ST_UnGeoreference(rast) where id=1;
____________________________________
(1 row)
```

5.11.32. ST_SetGeoreference

This function sets the geographic reference information about a raster object.

Syntax

```
raster ST_SetGeoreference(raster rast, integer srid, integer aop, double A,double B, double
C, double D, double E, double F);
```

Parameters

Parameter	Description
rast	The raster object.
srid	The spatial reference system identifier (SRID) of the raster object.
aop	The spatial reference point. Valid values: • 1: the center cell • 2: the upper-left cell
A, B, C, D, E, and F	The six parameters of an affine transformation. • x = A × Column number + B × Row number + C • y = D × Column number + E × Row number + F

Description

If the raster object is stored in external mode, you need to ensure that the raster object is geographically referenced and the specified SRID can be found in the spatial_ref_sys table. Otherwise, the information about the raster object cannot be updated.

Examples

```
update rast set rast=ST_SetGeoreference(rast,4326,1,8.4163,0,124,0,-8.4163,36.2) where id=1
;
____________________________________
(1 row)
```

5.11.33. ST_RPCGeoreference

Obtains the related information about the rational polynomial coefficients (RPCs) of a raster object. If the specified raster object contains RPC information, a JSON string is returned. If the specified raster object does not contain RPC information, null is returned.

Syntax

```
text ST_RPCGeoreference(raster raster_obj)
```

Parameters

Parameter	Description
raster_obj	The raster object.

Description

The ST_RPCGeoreference function returns a JSON string that contains RPC parameters. This string describes the reference information about the RPC space. The following table describes the supported parameters.

Parameter	Description	Type
lineOff	The line offset.	float
sampOff	The sampling offset.	float
latOff	The latitude offset.	float
longOff	The longitude offset.	float
heightOff	The height offset.	float
lineScale	The line ratio.	float
sampScale	The sampling ratio.	float
latScale	The latitude ratio.	float
longScale	The longitude ratio.	float
heightScale	The height ratio.	float
lineDenCoeff	The number of coefficients for a line denominator. Valid values: 1 to 20.	float array
lineNumCoeff	The number of coefficients for a line numerator. Valid values: 1 to 20.	float array
sampNumCoeff	The number of coefficients for a sample numerator. Valid values: 1 to 20.	float array
sampDenCoeff	The number of coefficients for a sample denominator. Valid values: 1 to 20.	float array

Parameter	Description	Type
errBias	The bias errors. The root-mean-square (RMS) bias errors in meters. These errors are related to all points per horizontal axis in an image. Default value: `-1.0`.	float
errRandom	The random errors. The RMS random errors in meters. These errors are related to each point per horizontal axis in an image. Default value: `-1.0`.	float

Example:

```
SELECT ST_RPCGeoreference(raster_obj) FROM raster_table where id=1;
_________________________________
 {"lineOff":12800.0,"sampOff":4008.0,"latOff":55.0203,"longOff":27.0478,"heightOff":179.0,"
lineScale":12800.0,"sampScale":4008.0,"latScale":0.1238,"longScale":0.0685,"heightScale":30
0.0,"errBias":-1.0,"errRandom":-1.0,"lineDenCoeff":[1.0,-0.00050066513,-0.0014578309,0.0006
037474.
.4,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0]","lineNumCoeff":[-0.0021
04832,-0.01642616,-1.027459,0.0041820025,-0.0019027952,0.000016143133,0.00047863558,-0.0002
1278669,0.0069588307,-0.0000022605722,-2.2259552e-7,-3.7469372e-7,0.00046486457,-1.8012888e
-8,0.0.
.000051407583,0.00075661479,-5.4524409e-7,1.3940799e-7,-0.000018281596,2.4215581e-9],"sampD
enCoeff":[1.0,-0.00050066513,-0.0014578309,0.00060374744,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.
0,0.0,0.0,0.0,0.0,0.0,0.0,0.0],"sampNumCoeff":[1.0,-0.00050066513,-0.0014578309,0.000603747
44,0.0.
.,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]}
```

5.11.34. ST_SetRPCGeoreference

Specifies rational polynomial coefficient (RPC) parameters for a raster object.

Syntax

```
raster ST_SetRPCGeoreference(raster raster_obj, text rpc_json)
```

Parameters

Parameter	Description
raster_obj	The raster object.
rpc_json	The RPC description in the JSON format.

Description

The following table describes the supported RPC parameters.

Parameter	Description	Type
lineOff	The line offset.	float
sampOff	The sampling offset.	float
latOff	The latitude offset.	float
longOff	The longitude offset.	float
heightOff	The height offset.	float
lineScale	The line ratio.	float
sampScale	The sampling ratio.	float
latScale	The latitude ratio.	float
longScale	The longitude ratio.	float
heightScale	The height ratio.	float
lineDenCoeff	The number of coefficients for a line denominator. Valid values: 1 to 20.	float array
lineNumCoeff	The number of coefficients for a line numerator. Valid values: 1 to 20.	float array
sampNumCoeff	The number of coefficients for a sample numerator. Valid values: 1 to 20.	float array
sampDenCoeff	The number of coefficients for a sample denominator. Valid values: 1 to 20.	float array
errBias	The bias errors. The root-mean-square (RMS) bias errors in meters. These errors are related to all points per horizontal axis in an image. Default value: `-1.0`.	float
errRandom	The random errors. The RMS random errors in meters. These errors are related to each point per horizontal axis in an image. Default value: `-1.0`.	float

Example:

```
SELECT ST_RPCGeoreference(ST_setRPCGeoreference(raster_obj, '{
    "lineOff": 12800.00,
    "sampOff": 4008.00,
    "latOff": 55.02030000,
    "longOff": 27.04780000,
    "heightOff": 179.000,
    "lineScale": 12800.00,
    "sampScale": 4008.00,
    "latScale": 0.12380000,
```

```
    "longScale": 0.06850000,
    "heightScale": 300.000,
    "lineNumCoeff": [
        -2.104832000000000E-03,
        -1.642616000000000E-02,
        -1.027459000000000E+00,
        4.182002500000000E-03,
        -1.902795200000000E-03,
        1.614313300000000E-05,
        4.786355800000000E-04,
        -2.127866900000000E-04,
        6.958830700000000E-03,
        -2.260572200000000E-06,
        -2.225955200000000E-07,
        -3.746937200000000E-07,
        4.648645700000000E-04,
        -1.801288800000000E-08,
        5.140758300000000E-06,
        7.566147900000000E-04,
        -5.452440900000000E-07,
        1.394079900000000E-07,
        -1.828159600000000E-05,
        2.421558100000000E-09
    ],
    "lineDenCoeff": [
        1.000000000000000E+00,
        -5.006651300000000E-04,
        -1.457830900000000E-03,
        6.037474400000000E-04,
        0.000000000000000E+00,
        0.000000000000000E+00,
        0.000000000000000E+00,
        0.000000000000000E+00,
        0.000000000000000E+00,
        0.000000000000000E+00,
        0.000000000000000E+00,
        0.000000000000000E+00,
        0.000000000000000E+00,
        0.000000000000000E+00,
        0.000000000000000E+00,
        0.000000000000000E+00,
        0.000000000000000E+00,
        0.000000000000000E+00,
        0.000000000000000E+00
    ],
    "sampNumCoeff": [
        1.000000000000000E+00,
        -5.006651300000000E-04,
        -1.457830900000000E-03,
        6.037474400000000E-04,
        0.000000000000000E+00,
        0.000000000000000E+00,
        0.000000000000000E+00,
        0.000000000000000E+00,
```

```
        0.000000000000000E+00,
        0.000000000000000E+00,
        0.000000000000000E+00,
        0.000000000000000E+00,
        0.000000000000000E+00,
        0.000000000000000E+00,
        0.000000000000000E+00,
        0.000000000000000E+00,
        0.000000000000000E+00,
        0.000000000000000E+00,
        0.000000000000000E+00,
        0.000000000000000E+00,
        0.000000000000000E+00,
        0.000000000000000E+00
    ],
    "sampDenCoeff": [
        1.000000000000000E+00,
        -5.006651300000000E-04,
        -1.457830900000000E-03,
        6.037474400000000E-04,
        0.000000000000000E+00,
        0.000000000000000E+00,
        0.000000000000000E+00,
        0.000000000000000E+00,
        0.000000000000000E+00,
        0.000000000000000E+00,
        0.000000000000000E+00,
        0.000000000000000E+00,
        0.000000000000000E+00,
        0.000000000000000E+00,
        0.000000000000000E+00,
        0.000000000000000E+00,
        0.000000000000000E+00,
        0.000000000000000E+00,
        0.000000000000000E+00,
        0.000000000000000E+00,
        0.000000000000000E+00,
        0.000000000000000E+00,
        0.000000000000000E+00,
        0.000000000000000E+00,
        0.000000000000000E+00
    ]
}'))
FROM raster_table
WHERE id = 1;
-----------------------------
 {"lineOff":12800.0,"sampOff":4008.0,"latOff":55.0203,"longOff":27.0478,"heightOff":179.0,"
lineScale":12800.0,"sampScale":4008.0,"latScale":0.1238,"longScale":0.0685,"heightScale":30
0.0,"errBias":-1.0,"errRandom":-1.0,"lineDenCoeff":[1.0,-0.00050066513,-0.0014578309,0.0006
037474.
.4,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0],"lineNumCoeff":[-0.0021
04832,-0.01642616,-1.027459,0.0041820025,-0.0019027952,0.000016143133,0.00047863558,-0.0002
1278669,0.0069588307,-0.0000022605722,-2.2259552e-7,-3.7469372e-7,0.00046486457,-1.8012888e
-8,0.0.
.000051407583,0.00075661479,-5.4524409e-7,1.3940799e-7,-0.000018281596,2.4215581e-9],"sampD
enCoeff":[1.0,-0.00050066513,-0.0014578309,0.00060374744,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.
0,0.0,0.0,0.0,0.0,0.0,0.0,0.0],"sampNumCoeff":[1.0,-0.00050066513,-0.0014578309,0.000603747
44,0.0.
.,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0]}
```

5.11.35. ST_NoData

This function returns the predefined NoData value for a band of a raster object. If the NoData value is not defined, the function returns null.

Syntax

```
float8 ST_NoData(raster raster_obj, integer band);
```

Parameters

Parameter	Description
raster_obj	The raster object.
band	The band sequence number, starting from 0.

Examples

```
select ST_NoData(raster_obj, 0) from raster_table where id=1;
__________________________________
0.000
```

5.11.36. ST_SetNoData

This function sets the NoData value for a specified band of a raster object.

Syntax

```
raster ST_SetNoData(raster rast, integer band_sn, double nodata_value);
```

Parameters

Parameter	Description
rast	The raster object.
band	The sequence number of the band, which starts from 0. A value of -1 indicates all bands.
nodata_value	The NoData value of the band.

Examples

```
update rast set rast=ST_SetNoData(rast,0, 999.999);
__________________________________
(1 row)
```

5.11.37. ST_ColorTable

This function returns the color table of a band of a raster object in JSON format.

Syntax

```
text ST_ColorTable(raster raster_obj, integer band);
```

Parameters

Parameter	Description
raster_obj	The raster object.
band	The sequence number of the band, which starts from 0.

Description

The following code shows color tables in JSON format.

- Four color components:

```
'{"compsCount":4,
    "entries":[
        {"value":0,"c1":0,"c2":0,"c3":0,"c4":255},
        {"value":1,"c1":0,"c2":0,"c3":85,"c4":255},
        {"value":2,"c1":0,"c2":0,"c3":170,"c4":255}
    ]
}'
```

- Three color components:

```
'{"compsCount":3,
    "entries":[
        {"value":0,"c1":0,"c2":0,"c3":0},
        {"value":1,"c1":0,"c2":0,"c3":85},
        {"value":2,"c1":0,"c2":0,"c3":170}
    ]
}'
```

If the band does not have a color table, the function returns null.

Examples

```
select ST_ColorTable(raster_obj,0) from raster_table where id = 1;
____________________________________
'{"compsCount":3,
    "entries":
    [
        {"value":0,"c1":0,"c2":0,"c3":0},
        {"value":1,"c1":0,"c2":0,"c3":85},
        {"value":2,"c1":0,"c2":0,"c3":170}
    ]
}'
```

5.11.38. ST_SetColorTable

This function sets the color table in JSON format for a specified band of a raster object.

Syntax

```
raster ST_SetColorTable(raster rast, integer band_sn, cstring clb);
```

Parameters

Parameter	Description
rast	The raster object.
band_sn	The sequence number of the band, which starts from 0.
clb	The color table in JSON format. For more information, see the description of the ST_ColorTable function.

Examples

```
update rast set rast=ST_SetColorTable(rast,0,
'{"compsCount":4,
    "entries":[
        {"value":0,"c1":0,"c2":0,"c3":0,"c4":255},
        {"value":1,"c1":0,"c2":0,"c3":85,"c4":255},
        {"value":2,"c1":0,"c2":0,"c3":170,"c4":255}
    ]
}');
_________________________________
(1 row)
```

5.11.39. ST_Statistics

This function returns the statistics about a band of a raster object in JSON format. If the band does not have statistics, the function returns null.

Syntax

```
text ST_Statistics(raster raster_obj, integer band);
```

Parameters

Parameter	Description
raster_obj	The raster object.
band	The sequence number of the band, which starts from 0.

Examples

```
select ST_Statistics(raster_obj, 0) from raster_table where id=1;
_________________________________
'{    "min" : 0.00, "max" : 255.00,"mean" : 125.00,"std" : 23.123,"approx" : false}'
```

5.11.40. ST_SetStatistics

This function sets the statistics about a specified band of a raster object.

Syntax

```
raster ST_SetStatistics(raster rast, integer band, double min, double max, double mean, dou
ble std,cstring samplingParams);
```

Parameters

Parameter	Description
rast	The raster object.
band	The sequence number of the band, which starts from 0.
min,max,mean,std	The statistics.
samplingParams	The sampling parameters, which can be *{"approx":false}* or *{"approx":true}*.

Examples

```
update rast set rast=ST_SetStatistics(rast,0,0.0, 255.0, 125.0, 23.6, '{"approx":false}');
_________________________________
(1 row)
```

5.11.41. ST_SummaryStats

This topic describes the ST_SummaryStats function. This function is used to calculate the statistics of a specified set of bands that are contained in a raster object.

Syntax

```
raster ST_SummaryStats(raster raster_obj)
raster ST_SummaryStats(raster raster_obj, cstring statsOption)
raster ST_SummaryStats(raster raster_obj,
                       cstring bands,
                       cstring statsOption)
```

Parameters

Parameter	Description
raster_obj	The name of the raster object.
bands	The sequence numbers of the bands. The value of this parameter can be an array of integers or an integer range. The integers start from 0. Examples: `'0'`, `'1-3'`, and `'1,2,3'`.
statsOptions	The JSON string to specify the statistics that you want to calculate.

The following table describes the parameter that is contained in the value of the statsOptions parameter.

Parameter	Description	Type	Format	Default value	Remarks
approx	Specifies whether to use the sampling method to calculate statistics.	boolean	None	true	• true: specifies to use the sampling method to calculate statistics. The statistics may be inaccurate. • false: specifies to calculate all statistics.

Examples:

Calculate the statistics for a specified set of bands that are contained in a raster object.

```
UPDATE raster_obj SET raster_obj=ST_SummaryStats(raster_obj) WHERE id = 1;
UPDATE rast SET rast=ST_SummaryStats(rast,'0-2','{"approx":false}') WHERE id = 1;
UPDATE rast SET rast=ST_SummaryStats(rast,'{"approx":false}') WHERE id = 1;
```

5.11.42. ST_ColorInterp

This function returns the color interpretation type of a band of a raster object.

Syntax

```
text ST_ColorInterp(raster raster_obj, integer band);
```

Parameters

Parameter	Description
raster_obj	The raster object.
band	The sequence number of the band, which starts from 0.

The following table describes the values of the interp parameter that is returned.

Value	Description
Undefined	The color interpretation type is not defined.
GrayIndex	The index of the associated gray value table.
RGBIndex	The index of the RGB color table.
RGBAIndex	The index of the RGBA color table.
RedBand	The red band in the RGB color model.
GreenBand	The green band in the RGB color model.
BlueBand	The blue band in the RGB color model.
AlphaBand	The alpha band in the RGBA color model.
HueBand	The hue band in the HSL color model.
SaturationBand	The saturation band in the HSL color model.
LightnessBand	The lightness band in the HSL color model.
CyanBand	The cyan band in the CMYK color model.
MagentaBand	The magenta band in the CMYK color model.
YellowBand	The yellow band in the CMYK color model.
BlackBand	The black band in the CMYK color model.
YCbCr_YBand	The Y band in the YCbCr color model.
YCbCr_CbBand	The Cb band in the YCbCr color model.
YCbCr_CrBand	The Cr band in the YCbCr color model.

Example

```
select ST_ColorInterp(raster_obj,0) from raster_table where id = 1;
____________________________________
RedBand
```

5.11.43. ST_SetColorInterp

This function sets the color interpretation type for a specified band of a raster object.

Syntax

```
raster ST_SetColorInterp(raster rast, integer band_sn, ColorInterp interp);
```

Parameters

Parameter	Description
rast	The raster object.
band_sn	The sequence number of the band, which starts from 0.
interp	The color interpretation type.

Description

The following table describes the values of the interp parameter.

Value	Description
Undefined	The color interpretation type is not defined.
GrayIndex	The index of the associated gray color table.
RGBIndex	The index of the associated RGB color table.
RGBAIndex	The index of the associated RGBA color table.
CMYKIndex	The index of the associated CMYK color table.
HSLIndex	The index of the associated HSL color table.
RedBand	The red band in the RGB color model.
GreenBand	The green band in the RGB color model.
BlueBand	The blue band in the RGB color model.
AlphaBand	The alpha band in the RGBA color model.
HueBand	The hue band in the HSL color model.
SaturationBand	The saturation band in the HSL color model.

Value	Description
LightnessBand	The lightness band in the HSL color model.
CyanBand	The cyan band in the CMYK color model.
MagentaBand	The magenta band in the CMYK color model.
YellowBand	The yellow band in the CMYK color model.
BlackBand	The black band in the CMYK color model.
YBand	The Y band in the YCbCr color model.
CbBand	The Cb band in the YCbCr color model.
CrBand	The Cr band in the YCbCr color model.

Examples

```
update rast set rast=ST_SetColorInterp(rast,0, 'CI_Cyan');
_________________________________
(1 row)
```

5.11.44. ST_Histogram

This function returns the histogram of a band of a raster object in text format. If the band does not have a histogram, the function returns null.

Syntax

```
text ST_Histogram(raster raster_obj, integer band);
```

Parameters

Parameter	Description
raster_obj	The raster object.
band	The band sequence number, starting from 0.

Examples

```
select ST_Histogram(raster_obj, 0) from raster_table where id=1;
____________________________________
{
 "approximate":false,
 "histsCounts":
 [2,1,1,0,8,17,47,101,193,345,443,640,877,1189,1560,1847,2087,2560,2816,3193,3567,3840,4101
,4415,4498,3876,3235,2458,1800,1598,1087,731,638,426,264,198,147,126,104,104,80,84,86,71,80
,62,74,85,72,80,70,88,69,68,62,58,63,51,53,55,54,56,55,63,47,39,49,59,66,62,64,73,66,72,67,
84,86,79,91,92,117,138,136,142,157,225,287,285,382,449,567,628,750,855,1021,1142,1242,1410,
1504,1590,1786,1870,2044,2099,2277,2373,2451,2585,2646,2882,2878,3091,3396,3620,3911,4124,4
304,4700,4893,5314,5446,5657,5765,5649,5749,5753,5601,5335,5161,4943,4592,4445,4207,4083,40
90,4270,4465,4514,4844,5204,5331,5597,5777,5838,6004,6316,6095,5762,5567,5465,4923,4677,422
0,3843,3401,3041,2571,2345,1972,1725,1376,1140,1008,841,716,548,442,373,308,212,133,78,68,3
1,24,12,2,2,1],
 "binFunction":
 {
  "type":"unknown",
  "binRange":
  {
   "minValue":29.0,
   "maxValue":208.0,
   "outRange":"include",
   "binValues":
   [29.0,30.0,31.0,32.0,33.0,34.0,35.0,36.0,37.0,38.0,39.0,40.0,41.0,42.0,43.0,44.0,45.0,46
.0,47.0,48.0,49.0,50.0,51.0,52.0,53.0,54.0,55.0,56.0,57.0,58.0,59.0,60.0,61.0,62.0,63.0,64.
0,65.0,66.0,67.0,68.0,69.0,70.0,71.0,72.0,73.0,74.0,75.0,76.0,77.0,78.0,79.0,80.0,81.0,82.0
,83.0,84.0,85.0,86.0,87.0,88.0,89.0,90.0,91.0,92.0,93.0,94.0,95.0,96.0,97.0,98.0,99.0,100.0
,101.0,102.0,103.0,104.0,105.0,106.0,107.0,108.0,109.0,110.0,111.0,112.0,113.0,114.0,115.0,
116.0,117.0,118.0,119.0,120.0,121.0,122.0,123.0,124.0,125.0,126.0,127.0,128.0,129.0,130.0,1
31.0,132.0,133.0,134.0,135.0,136.0,137.0,138.0,139.0,140.0,141.0,142.0,143.0,144.0,145.0,14
6.0,147.0,148.0,149.0,150.0,151.0,152.0,153.0,154.0,155.0,156.0,157.0,158.0,159.0,160.0,161
.0,162.0,163.0,164.0,165.0,166.0,167.0,168.0,169.0,170.0,171.0,172.0,173.0,174.0,175.0,176.
0,177.0,178.0,179.0,180.0,181.0,182.0,183.0,184.0,185.0,186.0,187.0,188.0,189.0,190.0,191.0
,192.0,193.0,194.0,195.0,196.0,197.0,198.0,199.0,200.0,201.0,202.0,203.0,204.0,205.0,206.0,
207.0]
  }
 }
}
```

5.11.45. ST_SetHistogram

This function sets the histogram in JSON format for a specified band of a raster object.

Syntax

```
raster ST_SetHistogram(raster rast, integer band, cstring histogram);
```

Parameters

Parameter	Description
rast	A raster object.
band	The sequence number of the band, which starts from 0.
histogram	The histogram in JSON format.

Description

The following table describes the parameters of the histogram in JSON format.

Parameter	Description	Type	Note
approximate	Specifies whether to enable data sampling.	BOOLEAN	None
histsCounts	The array that displays the statistics.	INTEGER[]	None
binFunction/type	The type of the bin function.	STRING	• linear • logarithm • explicit
binFunction/binTable/binValues	The bin values of the bin table.	NUMBER[]	This parameter takes effect only when the type of the bin function is explicit.
binFunction/binRange/minValue	The minimum value of the bin range.	NUMBER	This parameter takes effect only when the type of the bin function is logarithm or linear.
binFunction/binRange/maxValue	The maximum value of the bin range.	NUMBER	This parameter takes effect only when the type of the bin function is logarithm or linear.
binFunction/binRange/outRange	The value beyond the bin range.	NUMBER	This parameter takes effect only when the type of the bin function is logarithm or linear.
binFunction/binRange/binValues	The bin values of the bin range.	NUMBER[]	This parameter takes effect only when the type of the bin function is logarithm or linear.

```
Example 1:
{
  "approximate":false,
  "histsCounts":
  [2,1,1,0,8,17,47,101,193,345,443,640,877,1189,1560,1847,2087,2560,2816,3193,3567,3840,410
1,4415,4498,3876,3235,2458,1800,1598,1087,731,638,426,264,198,147,126,104,104,80,84,86,71,8
0,62,74,85,72,80,70,88,69,68,62,58,63,51,53,55,54,56,55,63,47,39,49,59,66,62,64,73,66,72,67
,84,86,79,91,92,117,138,136,142,157,225,287,285,382,449,567,628,750,855,1021,1142,1242,1410
,1504,1590,1786,1870,2044,2099,2277,2373,2451,2585,2646,2882,2878,3091,3396,3620,3911,4124,
4304,4700,4893,5314,5446,5657,5765,5649,5749,5753,5601,5335,5161,4943,4592,4445,4207,4083,4
090,4270,4465,4514,4844,5204,5331,5597,5777,5838,6004,6316,6095,5762,5567,5465,4923,4677,42
20,3843,3401,3041,2571,2345,1972,1725,1376,1140,1008,841,716,548,442,373,308,212,133,78,68,
31,24,12,2,2,1],
  "binFunction":
  {
    "type":"unknown",
    "binRange":
    {
      "minValue":29.0,
      "maxValue":208.0,
      "outRange":"include",
      "binValues":
      [29.0,30.0,31.0,32.0,33.0,34.0,35.0,36.0,37.0,38.0,39.0,40.0,41.0,42.0,43.0,44.0,45.0
,46.0,47.0,48.0,49.0,50.0,51.0,52.0,53.0,54.0,55.0,56.0,57.0,58.0,59.0,60.0,61.0,62.0,63.0,
64.0,65.0,66.0,67.0,68.0,69.0,70.0,71.0,72.0,73.0,74.0,75.0,76.0,77.0,78.0,79.0,80.0,81.0,8
2.0,83.0,84.0,85.0,86.0,87.0,88.0,89.0,90.0,91.0,92.0,93.0,94.0,95.0,96.0,97.0,98.0,99.0,10
0.0,101.0,102.0,103.0,104.0,105.0,106.0,107.0,108.0,109.0,110.0,111.0,112.0,113.0,114.0,115
.0,116.0,117.0,118.0,119.0,120.0,121.0,122.0,123.0,124.0,125.0,126.0,127.0,128.0,129.0,130.
0,131.0,132.0,133.0,134.0,135.0,136.0,137.0,138.0,139.0,140.0,141.0,142.0,143.0,144.0,145.0
,146.0,147.0,148.0,149.0,150.0,151.0,152.0,153.0,154.0,155.0,156.0,157.0,158.0,159.0,160.0,
161.0,162.0,163.0,164.0,165.0,166.0,167.0,168.0,169.0,170.0,171.0,172.0,173.0,174.0,175.0,1
76.0,177.0,178.0,179.0,180.0,181.0,182.0,183.0,184.0,185.0,186.0,187.0,188.0,189.0,190.0,19
1.0,192.0,193.0,194.0,195.0,196.0,197.0,198.0,199.0,200.0,201.0,202.0,203.0,204.0,205.0,206
.0,207.0]
    }
  }
}
Example 2:
{
  "approximate" : true,
  "histsCounts" :[1,2,3,4,5],
  "binFunction" :
  {
    "type" : "explicit",
    "binTable" :
    {
      "binValues" : [1.0,2.0,3.0,4.0,5.0]
    }
  }
}
```

Example

```
UPDATE rat SET raster = st_sethistogram(raster, 0, '{"approximate":true,"histsCounts":[1,2,
3,4,5],"binFunction":{"type":"explicit","binTable":{"binValues":[1.0,2.0,3.0,4.0,5.0]}}}')
where id =1;
--------------------------------
(1 row)
```

5.11.46. ST_BuildHistogram

This function computes the histogram of a band set of a raster object.

Syntax

```
raster ST_BuildHistogram(raster raster_obj);
```

Parameters

Parameter	Description
raster_obj	The raster object.

Examples

```
UPDATE raster_table SET raster_obj = st_buildhistogram(raster_obj) WHERE id = 1;
--------------------------------
(1 row)
```

5.11.47. ST_StatsQuantile

Calculates the quantiles of a raster object.

Syntax

```
raster ST_StatsQuantile(raster raster_obj)
```

Parameters

Parameter	Description
raster_obj	The name of the raster object.

Description

This function calculates quantiles based on bands and records the quantiles in the metadata of the raster object.

Examples

```
UPDATE rat_quantile
SET raster = ST_StatsQuantile(raster)
WHERE id = 1;
```

5.11.48. ST_Quantile

Queries the pixel values of the quantiles for a raster object.

Prerequisites

The quantiles of the raster object are calculated by using the ST_StatsQuantile function.

Syntax

```
setof record ST_Quantile(raster raster_obj,
                   float8[] quantiles default NULL,
                   cstring bands default '',
                   boolean exclude_nodata_value default true,
                   out integer band,
                   out float8 quantile,
                   out float8 value)
```

Parameters

Parameter	Description
raster_obj	The name of the raster object.
quantiles	The quantiles whose pixel values you want to calculate. Valid values: 0.25, 0.5, and 0.75. You can specify one or more quantiles.
bands	The serial numbers of the bands based on which the pixel values of the quantiles are calculated. Supported formats are `'0-2'` and `'1,2,3'`. Serial numbers start from 0. Default value: empty string (`''`). The default value specifies all bands.
exclude_nodata_value	Specifies whether to include NoData values during calculation.
band	Specifies to return the serial numbers of the bands.
quantile	Specifies to return the quantiles.
value	Specifies to return the pixel values.

Examples

```
-- Calculate the pixel value of the 0.25 quantile based on all bands.
SELECT  (ST_Quantile(rast, ARRAY[0.25], '0-2', true)). * FROM rat_quantile WHERE id = 1;
 band | quantile | value
------+----------+-------
    0 |     0.25 |    11
    1 |     0.25 |    10
    2 |     0.25 |    50
(3 rows)
-- Calculate the pixel values of the 0.25, 0.5, and 0.75 quantiles based on band 0.
SELECT  (ST_Quantile(rast, NULL, '0', true)). * FROM rat_quantile WHERE id = 1;
 band | quantile | value
------+----------+-------
    0 |     0.25 |    11
    0 |      0.5 |    11
    0 |     0.75 |    65
(3 rows)
```

5.11.49. ST_MD5Sum

Queries the MD5 hash string of a raster object.

Syntax

```
text ST_MD5Sum(raster raster_obj)
```

Parameters

Parameter	Description
raster_obj	The name of the raster object.

Description

The system searches the metadata of the raster object to obtain the MD5 hash string. If the MD5 hash string cannot be found, the system returns a NULL value. Then, the system calculates the MD5 hash string based on the Alibaba Cloud Object Storage Service (OSS) path where the raster object is stored.

The following table describes the parameters that you need to configure.

Parameter	Type	Description
ganos.raster.calculate_md5	boolean	Specifies whether to calculate the MD5 hash string and saves it to the metadata when the raster object is stored. Valid values: true | false. Default value: false. Example: `Set ganos.raster.calculate_md5 = true;`

Parameter	Type	Description
ganos.raster.md5sum_chunk_size	integer	Specifies the size of data that can be cached for each read operation when the system calculates the MD5 hash string. Unit: MB. Valid values: 1 to 256. Default value: 10. Example: `Set ganos.raster.md5sum_chunk_size = 20;`

Examples

```
SELECT ST_MD5SUM(rast)
FROM raster_table
WHERE id = 1;
              st_md5sum
----------------------------------
 21f41fd983d3139c75b04bff2b7bf5c9
```

5.11.50. ST_SetMD5Sum

Configures the MD5 hash string of a raster object.

Syntax

```
text ST_SetMD5Sum(raster raster_obj, text md5sum)
```

Parameters

Parameter	Description
raster_obj	The name of the raster object.
md5sum	The MD5 hash string of the raster object.

Description

This function records the MD5 hash string in the metadata of the raster object. The MD5 hash string must be 32 characters in length and can only contain digits and lowercase letters.

The following table describes the parameters that you need to configure.

Parameter	Type	Description
ganos.raster.calculate_md5	boolean	Specifies whether to calculate the MD5 hash string and saves it to the metadata when the raster object is stored. Valid values: true | false. Default value: false. Example: `Set ganos.raster.calculate_md5 = true;`

Examples

```
SELECT ST_MD5SUM(ST_SetMD5Sum(rast,'21f41fd983d3139c75b04bff2b7bf5c9'))
FROM raster_table
WHERE id = 1;
              st_md5sum
-----------------------------------
 21f41fd983d3139c75b04bff2b7bf5c9
```

5.11.51. ST_XMin

This function returns the minimum X-coordinate value of the specified raster object.

Syntax

```
float8 ST_XMin(raster raster_obj)
float8 ST_XMin(raster raster_obj, integer pyramid)
```

Parameters

Parameter	Description
raster_obj	The name of the raster object.
pyramid	The pyramid level. The value starts from 0.

Description

If the raster object has been georeferenced, this function returns the minimum X-coordinate value of the raster object. Otherwise, this function returns 0.

Examples

```
select ST_XMin(rast)
from raster_table;
 st_xmin
----------
      120
```

5.11.52. ST_YMin

This function returns the minimum Y-coordinate value of the specified raster object.

Syntax

```
float8 ST_YMin(raster raster_obj)
float8 ST_YMin(raster raster_obj, integer pyramid)
```

Parameters

Parameter	Description
raster_obj	The name of the raster object.
pyramid	The pyramid level. The value starts from 0.

Description

If the raster object has been georeferenced, this function returns the minimum Y-coordinate value of the raster object. Otherwise, this function returns 0.

Examples

```
select ST_YMin(rast)
from raster_table;
 st_ymin
----------
      30
```

5.11.53. ST_XMax

This function returns the maximum X-coordinate value of the specified raster object.

Syntax

```
float8 ST_XMax(raster raster_obj)
float8 ST_XMax(raster raster_obj, integer pyramid)
```

Parameters

Parameter	Description
raster_obj	The name of the raster object.
pyramid	The pyramid level. The value starts from 0.

Description

If the raster object has been georeferenced, this function returns the maximum X-coordinate value of the raster object. Otherwise, this function returns -1.

Examples

```
select ST_XMax(rast)
from raster_table;
 st_xmax
----------
      121
```

5.11.54. ST_YMax

This topic describes the ST_YMax function, which obtains the maximum Y coordinate of a raster.

Syntax

```
float8 ST_YMax(raster raster_obj)
float8 ST_YMax(raster raster_obj, integer pyramid)
```

Parameters

Parameter	Description
raster_obj	The raster whose maximum Y coordinate you want to obtain.
pyramid	The pyramid level at which the raster resides. Valid values start from 0.

Description

If the raster is georeferenced, this function returns the maximum Y coordinate of the raster. Otherwise, this function returns a value that is equal to the height of the raster minus 1.

Example

```
select st_ymax(rast)
from raster_table;
 st_ymax
----------
      31
```

5.11.55. ST_ChunkHeight

This function returns the tile height of the specified raster object.

Syntax

```
integer ST_ChunkHeight(raster raster_obj)
```

Parameters

Parameter	Description
raster_obj	The name of the raster object.

Description

This function returns the tile height of the specified raster object. The tile height is measured in pixels.

Examples

```
select ST_ChunkHeight(rast)
from raster_table;
 st_chunkheight
----------
      256
```

5.11.56. ST_ChunkWidth

This function returns the tile width of the specified raster object.

Syntax

```
integer ST_ChunkWidth(raster raster_obj)
```

Parameters

Parameter	Description
raster_obj	The name of the raster object.

Description

This function returns the tile width of the specified raster object. The width is measured in pixels.

Examples

```
select ST_ChunkWidth(rast)
from raster_table;
 st_chunkwidth
----------
      256
```

5.11.57. ST_ChunkBands

This function returns the number of bands for a tile of the specified raster object.

Syntax

```
integer ST_ChunkBands(raster raster_obj)
```

Parameters

Parameter	Description
raster_obj	The name of the raster object.

Examples

```
select ST_ChunkBands(rast)
from raster_table;
 st_chunkbands
----------
       3
```

5.11.58. ST_MetaItems

This topic describes the ST_MetaItems function, which returns the customized metadata items of a raster.

Syntax

```
text[] ST_metaItems(raster raster_obj);
text[] ST_metaItems(raster raster_obj,
                    integer band);
```

Parameters

Parameter	Description
raster_obj	The raster whose customized metadata items you want to obtain.
band	The sequence number of the band whose customized metadata items you want to obtain. A raster has one or more bands. Valid values start from 0.

Examples

```
-- meta data items of a raster
SELECT ST_MetaItems(raster_obj)
FROM raster_table;
                                                                                st_metaitems

-----------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------
 {latitude#long_name,latitude#units,longitude#long_name,longitude#units,NC_GLOBAL#Conventio
ns,NC_GLOBAL#history,NETCDF_DIM_EXTRA,NETCDF_DIM_time_DEF,NETCDF_DIM_time_VALUES}
-- meta data items of a band
SELECT ST_MetaItems(raster_obj, 0)
FROM raster_table;
                                        st_metaitems
-----------------------------------------------------------------------------------------------
{add_offset,long_name,missing_value,NETCDF_DIM_time,NETCDF_VARNAME,scale_factor,units,_Fill
Value}
```

5.11.59. ST_SetMetaData

This topic describes the ST_SetMetaData function, which specifies a metadata item of a raster or band.

Syntax

```
raster ST_SetMetaData(raster raster_obj,
                      text key,
                      text value);
raster ST_MetaData(raster raster_obj,
                   integer band,
                   text key,
                   text value);
```

Parameters

Parameter	Description
raster_obj	The raster whose metadata item you want to specify.
band	The sequence number of the band whose metadata item you want to specify. Valid values start from 0.
key	The name of the metadata item that you want to specify.
value	The value of the metadata item.

Description

If you specify an empty value (`''`) for a metadata item, the metadata item is deleted.

Examples

```
SELECT ST_MetaData(ST_SetMetaData(rast, 'NETCDF_DIM_time', '12345'), 'NETCDF_DIM_time')
FROM raster_table
st_metadata
------------
12345
SELECT ST_MetaData(ST_SetMetaData(rast, 0, 'NETCDF_DIM_time', '12345'), 0, 'NETCDF_DIM_time
')
FROM raster_table
st_metadata
------------
12345
```

5.11.60. ST_BeginDateTime

This topic describes the ST_BeginDateTime function, which obtains the start time of a raster.

Syntax

```
timestamp ST_BeginDateTime(raster raster_obj);
```

Parameters

Parameter	Description
raster_obj	The raster whose start time you want to obtain.

Examples

```
SELECT ST_beginDateTime(raster_obj)
FROM raster_table;
    st_begindatetime
-------------------------
Wed Jan 01 00:00:00 2020
```

5.11.61. ST_EndDateTime

This topic describes the ST_EndDateTime function, which obtains the end time of a raster.

Syntax

```
timestamp ST_EndDateTime(raster raster_obj);
```

Parameters

Parameter	Description
raster_obj	The raster whose end time you want to obtain.

Examples

```
SELECT ST_endDateTime(raster_obj)
FROM raster_table;
     st_enddatetime
-------------------------
Thu Jan 02 00:00:00 2020
```

5.11.62. ST_SetBeginDateTime

This topic describes the ST_SetBeginDateTime function, which specifies the start time of a raster.

Syntax

```
raster ST_SetBeginDateTime(raster raster_obj,timestamp time);
```

Parameters

Parameter	Description
raster_obj	The raster whose start time you want to specify.

Parameter	Description
time	The start time of the raster. We recommend that you specify the time in the `yyyy-MM-dd HH:mm:ss` format. Example: `2020-08-30 18:00:00`.

Examples

```
SELECT ST_beginDateTime(ST_setBeginDateTime(raster_obj, '2020-01-01'))
FROM raster_table;
    st_begindatetime
------------------------
Wed Jan 01 00:00:00 2020
```

5.11.63. ST_SetEndDateTime

This topic describes the ST_SetEndDateTime function, which specifies the end time of a raster.

Syntax

```
raster ST_SetEndDateTime(raster raster_obj, timestamp time);
```

Parameters

Parameter	Description
raster_obj	The raster whose end time you want to specify.
time	The end time of the raster. We recommend that you specify the time in the `yyyy-MM-dd HH:mm:ss` format. Example: `2020-08-30 18:00:00`.

Examples

```
SELECT ST_endDateTime(ST_setEndDateTime(raster_obj, '2020-01-01'))
FROM raster_table;
    st_enddatetime
------------------------
Wed Jan 01 00:00:00 2020
```

5.11.64. ST_DateTime

This topic describes the ST_DateTime function, which obtains the time information of a raster or band.

Syntax

```
text ST_DateTime(raster raster_obj);
timetamp ST_DateTime(raster raster_obj,integer band);
```

Parameters

Parameter	Description
raster_obj	The raster whose time information you want to obtain.
band	The sequence number of the band whose time information you want to obtain. Valid values start from 0.

Description

This function obtains the time information of a raster or band. If you specify the band parameter, this function returns the time information of the specified band. Otherwise, this function returns the time information of all bands of the raster by using a JSON-formatted string.

Examples

```
SELECT ST_DateTime(raster_obj)
FROM raster_table;

st_datetime

-----------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------
---------------------------------------
 {"0":"Mon Dec 31 00:00:00 2018","1":"Mon Dec 31 01:00:00 2018","2":"Mon Dec 31 02:00:00 20
18","3":"Mon Dec 31 03:00:00 2018","4":"Mon Dec 31 04:00:00 2018","5":"Mon Dec 31 05:00:00
2018","6":"Mon Dec 31 06:00:00 2018",".
.7":"Mon Dec 31 07:00:00 2018","8":"Mon Dec 31 08:00:00 2018","9":"Mon Dec 31 09:00:00 2018
","10":"Mon Dec 31 10:00:00 2018","11":"Mon Dec 31 11:00:00 2018","12":"Mon Dec 31 12:00:00
2018","13":"Mon Dec 31 13:00:00 2018".
.,"14":"Mon Dec 31 14:00:00 2018","15":"Mon Dec 31 15:00:00 2018","16":"Mon Dec 31 16:00:00
2018","17":"Mon Dec 31 17:00:00 2018","18":"Mon Dec 31 18:00:00 2018","19":"Mon Dec 31 19:0
0:00 2018","20":"Mon Dec 31 20:00:00 .
.2018","21":"Mon Dec 31 21:00:00 2018","22":"Mon Dec 31 22:00:00 2018","23":"Mon Dec 31 23:
00:00 2018"}
SELECT ST_DateTime(raster_obj, 0)
FROM raster_table;
         datetime
---------------------------
"Mon Dec 31 00:00:00 2018"
```

5.11.65. ST_SetDateTime

This topic describes the ST_SetDateTime function, which specifies the start and end time of a raster or the time information of a band.

Syntax

```
raster ST_setDateTime(raster raster_obj,
                      timestamp start,
                      timestamp end);
raster ST_setDateTime(raster raster_obj,
                      integer band,
                      timestamp time);
```

Parameters

Parameter	Description
raster_obj	The raster whose start and end time you want to specify.
band	The sequence number of the band whose start and end time you want to specify. Valid values start from 0.
time	The time information of the specified band. We recommend that you specify the time in the `yyyy-MM-dd HH:mm:ss` format. Example: `2020-08-30 18:00:00`.
start	The start time that you want to specify. We recommend that you specify the time in the `yyyy-MM-dd HH:mm:ss` format.
end	The end time that you want to specify. We recommend that you specify the time in the `yyyy-MM-dd HH:mm:ss` format.

Examples

```
select ST_DateTime(ST_setDateTime(raster_obj, 0, '2020-01-02'::timestamp), 0)
FROM raster_table
       st_datetime
--------------------------
 Thu Jan 02 00:00:00 2020
```

5.11.66. ST_BuildPercentiles

This function is used to calculate the percentiles of the specified bands of a raster.

Syntax

```
raster ST_BuildPercentiles(raster raster_obj,
                           cstring sampleOption default '{}')
raster ST_BuildPercentiles(raster raster_obj,
                           cstring bands,
                           cstring sampleOption default '{}',
                           cstring parallelOption default '{}')
```

Parameters

Parameter	Description
raster_obj	The raster that you want to query.
sampleOption	A JSON string that consists of sampling options. For more information, see the "sampleOption" table in this topic.
bands	The IDs of the bands. Example: `'0,1-3,6,8'`. Valid band IDs start from `0`. If you set this parameter to `''`, all bands of the raster are specified.
parallelOption	A JSON string that consists of parallel operation options. For more information, see the "parallelOption" table of this topic.

The following table describes the parameters in the sampleOption parameter.

Parameter	Type	Default value	Description
approx	boolean	false	Specifies whether to use the sampling method. Valid values: true and false. **Note** If you set this parameter to true, the sampling method is used. In this case, the return result may be inaccurate.
factor	integer	4	The sampling factor that is used when the sampling method is used. Valid values: any positive integer. The sampling factor specifies the number of pixels that are used as a sampling unit. **Note** This parameter is valid only when you set the approx parameter to true.
exclusive_nodata	boolean	true	Specifies whether to include NoData values. Valid values: true and false.

The following table describes the parameters in the parallelOptions parameter.

Parameter	Description	Type	Default value	Remarks
parallel	The degree of parallelism that is allowed.	integer	ganos.parallel.degree	Valid values: 1 to 64.

Examples

```
-- Use the default parameter settings.
UPDATE raster_table SET raster_obj = ST_BuildPercentiles(raster_obj) WHERE id = 1;
-- Specify the bands and the sampling method.
UPDATE raster_table SET raster_obj = ST_BuildPercentiles(rast,'0-2','{"approx":false}') WHE
RE id = 1;
-- Specify the sampling method.
UPDATE raster_table SET raster_obj = ST_BuildPercentiles(rast,'{"approx":false}') WHERE id
= 1;
-- Specify the bands, sampling rate, and degree of parallelism.
UPDATE raster_table SET raster_obj = ST_BuildPercentiles(raster_obj,'0-2','{"approx":true,
"factor":5}', '{"parallel":4}') WHERE id = 1;
```

5.11.67. ST_ComputeStatistics

This function is used to collect all statistics about a raster. The statistics include the histogram and percentiles.

Syntax

```
raster ST_ComputeStatistics(raster raster_obj,
                            cstring sampleOption default '{}')
raster ST_ComputeStatistics(raster raster_obj,
                            cstring bands,
                            cstring sampleOption default '{}',
                            cstring parallelOption default '{}')
```

Parameters

Parameter	Description
raster_obj	The raster that you want to query.
sampleOption	A JSON string that consists of sampling options. For more information, see the "sampleOption" table in this topic.
bands	The IDs of the bands. Example: `'0,1-3,6,8'`. Valid band IDs start from `0`. If you set this parameter to `''`, all bands of the raster are specified.
parallelOption	A JSON string that consists of parallel operation options. For more information, see the "parallelOption" table of this topic.

The following table describes the parameters in the sampleOption parameter.

Parameter	Type	Default value	Description

Parameter	Type	Default value	Description
approx	boolean	false	Specifies whether to use the sampling method. Valid values: true and false. **Note** If you set this parameter to true, the sampling method is used. In this case, the return result may be inaccurate.
factor	integer	4	The sampling factor that is used when the sampling method is used. Valid values: any positive integer. The sampling factor specifies the number of pixels that are used as a sampling unit. **Note** This parameter is valid only when you set the approx parameter to true.
exclusive_nodata	boolean	true	Specifies whether to include NoData values. Valid values: true and false.

The following table describes the parameters in the parallelOptions parameter.

Parameter	Description	Type	Default value	Remarks
parallel	The degree of parallelism that is allowed.	integer	ganos.parallel.degree	Valid values: 1 to 64.

Examples

```
-- Use the default parameter settings.
UPDATE raster_table SET raster_obj = ST_ComputeStatistics(raster_obj) WHERE id = 1;
-- Specify the bands and the sampling method.
UPDATE raster_table SET raster_obj = ST_ComputeStatistics(rast,'0-2','{"approx":false}') WH
ERE id = 1;
-- Specify the sampling method.
UPDATE raster_table SET raster_obj = ST_ComputeStatistics(rast,'{"approx":false}') WHERE id
= 1;
-- Specify the bands, sampling rate, and degree of parallelism.
UPDATE raster_table SET raster_obj = ST_ComputeStatistics(raster_obj,'0-2','{"approx":true,
"factor":5}', '{"parallel":4}') WHERE id = 1;
```

5.11.68. ST_Percentile

This topic describes how to obtain the percentage of bands for imagery.

Syntax

```
float8 ST_Percentile(raster raster_obj,
                     integer band,
                     integer percentage)
```

Parameters

Parameter	Description
raster_obj	The raster object.
band	The specified band SNs that start from `0` in a format similar to `'0,1-3,6,8'`. Double quotation marks (`''`) indicate all bands.
percentage	The percentage of the specified bands for the raster object. You can use ST_BuildPercentiles or ST_ComputeStatistics to obtain the percentage.

Examples

```
SELECT st_percentile(rast_obj, 0, 10)
FROM raster_table
WHERE id = 1;
--------------------
29.7
```

You can also use ST_Percentile to implement the ST_Quantile feature.

```
SELECT st_percentile(rast_obj, 0, 25), st_percentile(rast_obj, 0, 50)
FROM raster_table
WHERE id = 1;
```

5.12. Algebra and Analysis Functions

5.12.1. ST_Reclassify

The ST_Reclassify function returns a raster object. The returned object has the same spatial reference and resolution as the original image. The number of bands is specified by the relassexpr parameter.

Syntax

```
raster ST_Reclassify(raster raster_obj,
                 cstring reclassexpr default NULL
                 cstring storageOption default '')
```

Parameters

Parameter	Description
raster_obj	The raster object to be reclassified.

Parameter	Description
reclassexpr	The JSON string is used to indicate a numerical interval for classification.
storageOption	The storage option that is represented by a JSON string in the returned result.

The reclassexpr parameter indicates a JSON string array. Each child JSON object indicates a band operation parameter. The following table describes the parameters.

Parameter	Description	Type	Default value	Notes
band	The band sequence number.	integer	None	The band sequence number starts from 0.
remap	The parameter that is used for classification.	object	-	-
nodata	Specifies whether to use nodata.	boolean	false	• If you set this parameter to true, the pixel value is nodata and the classification result is also nodata. • If you set this parameter to false, non-nodata values are calculated as general numeric values.
nodataValue	The nodata value.	float8	0	The new nodata value.

The remap parameter specifies how to map original pixel values to new pixel values.

- The key indicates the original pixel value range and consists of one or more numeric values that are separated with commas (,). You can specify an open or closed range at the start and end.
  - A left parenthesis "(" indicates greater than.
  - A right parenthesis ")" indicates less than.
  - A right bracket "]" indicates less than or equal to.
  - A left bracket "[" indicates greater than or equal to.

  The default value is `()`.
- The value indicates the result of the mapping from original to new pixel values and consists of one or more numeric values that are separated with commas (,).
- Three mapping methods are available:
  - Range-to-range mapping: The original and new pixel ranges have the same number of numeric values. Examples: "300,400,500":"80,90,100" and "[300,400,500]":"80,90,100".
  - Range-to-value mapping: The original pixel range has one more numeric value than the new pixel range. Example: "(300,400,500]":"80,90".
  - Value-to-value mapping: The original and new pixel ranges both have only one numeric value. Example: "10":"1".

- If a pixel value does not belong to a mapping range, the pixel value is considered as a nodata value.
- Multiple ranges cannot contain the same pixel value.
- Examples
  - Example 1

    The following example shows how to reclassify band 0:

```
if 0<old_value<=100
    new_value = 20
else if 100<old_value<=200
    new_value = 50
else
    new_value = 0
```

```
[
   {
      "band":0,
      "remap":{
            "(0,100,200]":"20,50"
      }
   }
]
```

  - Example 2

    Multiple segmented values are supported. Pixel values beyond the range are set to nodata values.

```
[
   {
      "band":0,
      "remap":{
            "(0,100,200]":"20,50",
            "(300,400,500]":"80,90,100"
      }
   }
]
```

- Example 3

  The following example shows how to reclassify band 0:

```
if 0<old_value<=100
    new_value = 20
else if 100<old_value<=200
    new_value = 50
else
    new_value = 999
```

  The following example shows how to reclassify band 1:

```
if 400<old_value<=600
    new_value = 20+(old_value-400)/200 * (90-20)
else if 600<old_value<=800
    new_value = 90+(old_value-600)/200 * (130-90)
else
    new_value = 0
```

```
[
   {
      "band":0,
      "remap":{
         "(0,100,200]":"20,50"
      },
      "nodata":true,
      "nodataValue":999
   },
   {
      "band":1,
      "remap":{
        "(400,600,800]":"20,90,130"
      },
      "nodata":false,
      "nodataValue":0
   }
]
```

The following table describes the parameters of storageOption.

Parameter	Description	Type	Default value	Notes
chunking	Specifies whether to store data as chunks.	boolean	Same as the original raster	-
chunkdim	The dimension information about the chunk.	string	Same as the original raster	This parameter only takes effect when the chunking parameter is set to true.

Parameter	Description	Type	Default value	Notes
chunktable	The name of the chunk table.	string	''	If the '' value is passed, a temporary chunk table that has a random name is generated to store data. This temporary table is valid in only the current session. If you need to retain an accessible clipping object, you must specify the name of the chunk table.
compression	The type of the compression algorithm.	string	Same as the original raster	Only none, jpeg, zlib, png, lzo, and lz4 are supported.
quality	The compression quality.	integer	Same as the original raster	This parameter only takes effect for the jpeg compression algorithm.
interleaving	The interleaving method.	string	Same as the original raster	The value must be one of the following types: • bip: band interleaved by pixel (BIP) • bil: band interleaved by line (BIL) • bsq: band sequential (BSQ)
endian	The endian.	string	Same as the original raster	The value must be one of the following types: • NDR: little endian • XDR: big endian
celltype	The pixel type.	string	Same as the original raster	-

Examples

```
-- Create a permanent table.
CREATE TABLE rast_reclassify_result(id integer, rast raster);
-- Create a temporary table.
CREATE TEMP TABLE rast_reclassify_result_temp(id integer, rast raster);
-- Store the result in a temporary table.
INSERT INTO rast_reclassify_result_temp(id, rast)
select 1, ST_Reclassify(rast, '[{"band":0,"remap":{"(0,100,200]":"20,50"}}]')
from reclass_table
```

5.12.2. ST_MapAlgebra

This topic describes the ST_MapAlgebra function. This function uses algebraic expressions in compliance with Algebra Computing Language to transform multiple original raster objects into one raster object.

Syntax

```
raster ST_MapAlgebra(raster[] rasters ,
        cstring algebraExpr  default NULL,
        cstring storageoption default '')
```

Parameters

Parameter	Description
rasters	The original raster objects that you want to transform.
algebraExpr	The JSON string that specifies the algebraic expressions that you want to use.
storageOption	The JSON string that specifies how to store the new raster object.

Note The system only requires that the lengths and widths of the original raster objects are in the same unit of measurement. It does not check their spatial reference systems or resolutions. If the lengths and widths of the original raster objects are in different units of measurement, you can use the ST_Tranform, ST_Resize, and ST_Clip functions to convert them into the same unit.

Each child JSON object in the JSON string specified by the algebraExpr parameter represents a field dictated to an algebraic expression. The following table describes these fields.

Field	Description	Type	Default value	Setting notes
algebraExpr	The algebraic expression that you want to use.	String	N/A	N/A.
nodata	Specifies whether nodata values are valid.	Boolean	false	• If you set this field to true, nodata values are valid and pixels with nodata values are not transformed. • If you set this field to false, nodata values are invalid and pixels with nodata values are transformed.
nodataValue	The nodata value to return for the new raster object.	float8	0	N/A.

The algebraic expression specified by the algebraExpr field contains the following keywords:

- [r, b]
  - r: indicates the ID of the new raster object in the array specified by the rasters parameter. Format: 0-n-1.
  - b: indicates the band number of the new raster object. Format: 0-n-1.
- x

  The sequence number of the column where the specified pixel resides.
- y

  The sequence number of the row where the specified pixel resides.

The following table lists the operations that you can incorporate in an algebraic expression.

<table>
<tr><th>Type</th><th>Operator or function</th><th>Remarks</th></tr>
<tr><td>Operators</td><td>• +
• -
• *
• /
• % (remainder)
• ** (power)</td><td>N/A.</td></tr>
<tr><td>Bitwise operations</td><td>• <<
• >>
• &
• |
• ^</td><td>N/A.</td></tr>
<tr><td>Logical operations</td><td>• <
• >
• ==
• !=
• <=
• >=
• &&
• ||
• !</td><td>N/A.</td></tr>
</table>

Type	Operator or function	Remarks
Operations on functions	• abs • sqrt • exp • log • ln • sin • cos • tan • sinh • cosh • tanh • arcsin • arccos • arctan • ceil • floor • round	You can specify only one operation on a function.
Statistical functions	• min • max • sum • mean • majority • minority • std • median • range • variety	You must specify two or more statistical functions.

- Example 1

  Transform the original raster object into a new raster object that only has one band. The return result is as follows: raster[0]band[0] + raster[1]band[0] * raster[1]band[1].

```
[
    {
        "expr":"([0,0] + [1,0] * [1,1]) ",
        "nodata": true,
        "nodataValue":999
    }
]
```

- Example 2

  Compute the variance of three bands.

```
[
    {
        "expr":"(std([0,0],[0,1],[0,2]))",
        "nodata": true,
        "nodataValue":999
    }
]
```

- Example 3

  Transform the original raster objects into a new raster object that has three bands. Each band is transformed by using a unique expression.

```
[
    {
        "expr":"(min([0,0],[0,1],[0,2]))",
        "nodata": true,
        "nodataValue":999
    },
    {
        "expr":"(max([0,0],[0,1],[0,2]))",
        "nodata": true,
        "nodataValue":999
    },
    {
        "expr":"(mean([0,0],[0,1],[0,2]))",
        "nodata": true,
        "nodataValue":999
    }
]
```

The following table describes fields in the storageOption parameter.

Field	Description	Type	Default value	Setting notes
chunking	Specifies whether to store the new raster object as chunks.	Boolean	Same as the original raster object	N/A.
chunkdim	The dimensions used to store the new raster object as chunks.	String	Same as the original raster object	This field only takes effect when the chunking field is set to true.

Field	Description	Type	Default value	Setting notes
chunktable	The name of the chunk table.	String	Null string (''')	By default, a temporary chunk table with a random name is generated to store data. This temporary chunk table is only valid in the current session. To save the new raster object permanently, you must specify you want to create a permanent chunk table in the chunktable field.
compression	The format used for image compression.	String	Same as the original raster object	Six compression formats are supported: None, JPEG, Zlib, PNG, LZO, and LZ4.
quality	The image quality of the new raster object.	Integer	Same as the original raster object	This field only takes effect in JPEG format.
interleaving	The interleaving type of the new raster object.	String	Same as the original raster object	Valid values: • bip: band interleaved by pixel (BIP) • bil: band interleaved by line (BIL) • bsq: band sequential (BSQ)
endian	The endian format of the new raster object.	String	Same as the original raster object	Valid values: • NDR: specifies little endian format. • XDR: specifies big endian format.

Examples

```
-- Create a permanent chunk table.
CREATE TABLE rast_mapalgebra_result(id integer, rast raster);
-- Insert data into a chunk table.
WITH foo AS (
  SELECT 1 AS rid, rast AS rast from t1 WHERE id = 1
UNION ALL
  SELECT 2 AS rid, rast AS rast from t2 WHERE id = 2
)
INSERT INTO rast_mapalgebra_result
SELECT 1, ST_MapAlgebra(
    ARRAY(SELECT rast FROM foo ORDER BY rid),
    '[{"expr":"([0,0] + 0.5 * [1,0] - ([1,1])","nodata": true, "nodataValue":999}]',
    '{"chunktable":"algebra_rbt"}'
);
```

5.13. Raster Image Processing

5.13.1. ST_SubRaster

This topic describes the ST_SubRaster function. This function is used to convert the specified pyramid level or bands of an original raster object into a new raster object.

Syntax

```
raster ST_SubRaster(raster raster_obj ,
                    integer pyramidLevel default 0,
                    cstring bands default '',
                    cstring storageOption  default '',
                    cstring options default '{}')
```

Parameters

Parameter	Description
raster_obj	The name of the original raster object.
pyramidLevel	The pyramid level that you want to convert.
bands	The bands that you want to convert. The value of this parameter can be an array of integers or an integer range. The integers start from 0. Examples: `'0-2'` and `'1,2,3'`. Default value: `''`. The default value specifies to generate a new raster object based on all bands.
storageOption	The JSON string that specifies how to store the new raster object. For more information, see the storageOption parameter.
options	The JSON string that specifies how to generate the new raster object. For more information, see the options parameter.

The following table describes the parameters that are contained in the value of the storageOption parameter.

Parameter	Description	Type	Default value	Remarks
chunking	Specifies whether to store the new raster object as chunks.	boolean	Same as the original raster object.	None.
chunkdim	The dimensions that are used to store the new raster object as chunks.	string	Same as the original raster object.	This parameter is valid only if the `chunking` parameter is set to true.
chunktable	The name of the chunk table that is used to store the new raster object.	string	`''`	If `''` or the value NULL is specified, a temporary chunk table with a random name is generated to store data. This temporary chunk table is valid only in the current session. To create a permanent chunk table for the new raster object, you must specify a name for the chunk table.
compression	The format that is used to compress the raster object.	string	Same as the original raster object.	Valid values: • NONE • JPEG • ZLIB • PNG • LZO • LZ4
quality	The image quality of the new raster object after compression.	integer	Same as the original raster object.	Valid values: 1 to 99. This parameter is valid only if the **compression** parameter is set to JPEG.
interleaving	The interleaving type of the new raster object.	string	Same as the original raster object.	Valid values: • bip: band interleaved by pixel (BIP) • bil: band interleaved by line (BIL) • bsq: band sequential (BSQ)

Parameter	Description	Type	Default value	Remarks
endian	The endian format of the new raster object.	string	Same as the original raster object.	Valid values: • NDR: little endian • XDR: big endian
celltype	The pixel type of the new raster object.	string	Same as the original raster object.	Valid value: • 1bb: 1-bit Boolean • 2bui: 2-bit unsigned integer • 4bui: 4-bit unsigned integer • 8bsi: 8-bit signed integer • 8bui: 8-bit unsigned integer • 16bsi: 16-bit signed integer • 16bui: 16-bit unsigned integer • 32bsi: 32-bit signed integer • 32bui: 32-bit unsigned integer • 64bsi: 64-bit signed integer • 64bui: 64-bit unsigned integer • 32bf: 32-bit floating number • 64bf: 64-bit floating number

The following table describes the parameters that are contained in the value of the options parameter.

Parameter	Description	Type	Default value	Remarks
parallel	The degree of parallelism that is allowed.	integer	ganos.parallel.degree	Valid values: 1 to 64.

Parameter	Description	Type	Default value	Remarks
stretch	The method that is used to stretch pixel values.	string	none	Valid values: • none: This function does not stretch pixel values. If a pixel value is beyond the value range that is supported by the pixel type, this function replaces the pixel value with the specified extremum. This extremum can be either the maximum or minimum value, depending on the pixel value. • stats: This function stretches pixel values based on the statistics of the original raster. The statistics of the original raster object must be obtained. Before you invoke this function, you can use the ST_SummaryStats function to obtain the statistics. • data_type: This function stretches pixel values based on the extrema that are specified for the pixel type.

Examples:

Convert the specified band into a new raster object.

```
SELECT ST_SubRaster(rast, 1, '0-2', '{"chunktable":"chunk_table", "chunking":true}')
FROM raster_sub
WHERE id=1;
```

Convert the pixel type and stretch pixel values in parallel.

```
SELECT ST_SubRaster(rast, 1, '0-2', '{"chunktable":"chunk_table", "chunking":true, "celltyp
e": "8BUI"}', '{"stretch": "data_type", "parallel": 4}')
FROM raster_sub
WHERE id=1;
```

5.13.2. ST_Transform

This topic describes the ST_Transform function. This function transforms the underlying coordinates of a raster object from one known spatial reference system to another.

Syntax

```
raster ST_Transform(raster rast,
           integer outSrid,
           cstring processexpr default '',
           cstring storageOption default '')
```

Parameters

Parameter	Description
rast	The original raster object whose coordinates you want to transform.
outSrid	The spatial reference system of the new raster object. The value of this parameter must be a valid SIRD that can be queried from the spatial_ref_sys table.
processExpr	The JSON string that specifies how to resample pixels and process nodata values.
storageOption	The JSON string that specifies how to store the new raster object.

Each child JSON object in the JSON string specified by the processExpr parameter represents a field. The following table describes these fields.

Field	Description	Type	Default value	Setting notes
resample	The method used to resample pixels.	Text	Near	Valid values: Near | Average | Cubic | Bilinear.
nodata	Specifies whether nodata values from the original raster object are valid.	Boolean	false	• If you set this field to true, nodata values are valid and pixels with nodata values are not resampled. • If you set this field to false, nodata values are invalid and pixels with nodata values are resampled.

Field	Description	Type	Default value	Setting notes
nodataValue	The new nodata value specified based on the bands in the new raster object.	float8 float8[]	NULL	You can specify one or more nodata values in the nodataValue field. • If you specify one nodata value, this nodata value is used for all bands in the new raster object. • If you specify more than one nodata value, the number of specified nodata values must be the same as the number of bands in the new raster object.

Note Exercise caution when specifying the nodata and nodataValue fields. If the original raster object does not have pixels with nodata values, we recommend that you set the nodata field to false and do not specify the nodataValue field. Otherwise, image artifacts may occur.

The following table describes fields in the storageOption parameter.

Field	Description	Type	Default value	Setting notes
chunking	Specifies whether to store the new raster object as chunks.	Boolean	Same as the original raster object	N/A.
chunkdim	The dimensions used to store the new raster object as chunks.	String	Same as the original raster object	This field only takes effect when the chunking field is set to true.
chunktable	The name of the chunk table.	String	Null string (''')	By default, a temporary chunk table with a random name is generated to store data. This temporary chunk table is only valid in the current session. To save the new raster object permanently, you must specify you want to create a permanent chunk table in the chunktable field.

Field	Description	Type	Default value	Setting notes
compression	The format used for image compression.	String	Same as the original raster object	Six compression formats are supported: None, JPEG, Zlib, PNG, LZO, and LZ4.
quality	The image quality of the new raster object.	Integer	Same as the original raster object	This field only takes effect in JPEG format.
interleaving	The interleaving type of the new raster object.	String	Same as the original raster object	Valid values: • bip: band interleaved by pixel (BIP) • bil: band interleaved by line (BIL) • bsq: band sequential (BSQ)
endian	The endian format of the new raster object.	String	Same as the original raster object	Valid values: • NDR: specifies little endian format. • XDR: specifies big endian format.

Note If you set the chunktable field to NULL or a null string (''), a temporary chunk table with a random name is generated in the current session to store the new raster object. The temporary chunk table is only valid in the current session, and is deleted immediately after the current session ends. To save the new raster object permanently, you must specify you want to create a permanent chunk table in the chunktable field.

Examples

```
CREATE TABLE if not exists datasource_table(id integer, rast raster);
INSERT INTO datasource_table values(1, ST_ImportFrom('rbt','$(RAST_DATA_DIR)/512_512_1_bsq_
8u_geo.tif', '{}'));
-----------------------------------------------------
-- Method 1: Specify the chunktable field to create a permanent chunk table for storing the
new raster object.
-----------------------------------------------------
CREATE TABLE rat_transform_result(id integer, rast raster);
-- If you do not specify the nodata field:
INSERT INTO rat_transform_result(id, rast)
select 10, ST_Transform(rast,32652, '{"resample":"Near","nodata":false}','{"chunking":true,
"chunkdim":"(256,256,1)","compression":"none","interleaving":"bsq","chunktable":"result_rbt
"}')
from datasource_table
where id =1;
-- If you specify one nodata value in the nodataValue field and pixels with nodata values a
re resampled:
INSERT INTO rat_transform_result(id, rast)
select 11, ST_Transform(rast,32652, '{"resample":"Near","nodata":true,"nodatavalue":255}','
{"chunking":true,"chunkdim":"(256,256,1)","compression":"none","interleaving":"bsq","chunkt
able":"result_rbt"}')
from datasource_table
where id =1;
-- If you specify more than one nodata value in the nodataValue field:
INSERT INTO rat_transform_result(id, rast)
select 12, ST_Transform(rast,32652, '{"resample":"Near","nodata":false,"nodatavalue":[255,2
55,255]}','{"chunking":true,"chunkdim":"(256,256,1)","compression":"none","interleaving":"b
sq","chunktable":"result_rbt"}')
from datasource_table
where id =1;
-----------------------------------------------------
-- Method 2: Do not specify the chunktable field. The new raster object is saved to a tempo
rary chunk table with a random name and can only be used for nested computations in the cur
rent session.
-----------------------------------------------------
CREATE TEMP TABLE rat_transform_result_temp(id integer, rast raster);
INSERT INTO rast_clip_result_temp(id, rast)
select 1, ST_Transform(rast,32652,'{"resample":"Near","nodata":false, "nodataValue":[255,25
5,255]}','{"chunking":true,"chunkdim":"(256,256,1)","compression":"none","interleaving":"bs
q"}')
from datasource_table
where id =1;
```

5.13.3. ST_Rescale

This topic describes the ST_Rescale function. This function rescales the pixel range of a raster object but leaves the geospatial data range unchanged.

Syntax

```
raster ST_Rescale(raster rast,
           f8 scale_x,
           f8 scale_y,
             cstring processexpr default '',
             cstring storageOption default '');
raster ST_Rescale(raster raster,
           f8 scale_xy,
             cstring processexpr default '',
             cstring storageOption default '')
```

Parameters

Parameter	Description
rast	The original raster object whose pixel range you want to rescale.
scale_x	The proportional ratio to rescale the original raster object in the x direction.
scale_y	The proportional ratio to rescale the original raster object in the y direction.
scale_xy	The proportional ratio to rescale the original raster object in both the x and y directions.
processExpr	The JSON string that specifies how to resample pixels and process nodata values.
storageOption	The JSON string that specifies how to store the new raster object.

Each child JSON object in the JSON string specified by the processExpr parameter represents a field. The following table describes these fields.

Field	Description	Type	Default value	Setting notes
resample	The method used to resample pixels.	text	Near	Valid values: Near | Average | Cubic | Bilinear.
nodata	Specifies whether nodata values from the original raster object are valid.	Boolean	false	• If you set this field to true, nodata values are valid and pixels with nodata values are not resampled. • If you set this field to false, nodata values are invalid and pixels with nodata values are resampled.

Field	Description	Type	Default value	Setting notes
nodataValue	The new nodata value specified based on the bands in the new raster object.	float8 float8[]	NULL	You can specify one or more nodata values in the nodataValue field. • If you specify one nodata value, this nodata value is used for all bands in the new raster object. • If you specify more than one nodata value, the number of specified nodata values must be the same as the number of bands in the new raster object.

Note Exercise caution when you specify the nodata and nodataValue fields. If the original raster object does not have pixels with nodata values, we recommend that you set the nodata field to false and do not specify the nodataValue field. Otherwise, image artifacts may occur.

The following table describes fields in the storageOption parameter.

Field	Description	Type	Default value	Setting notes
chunking	Specifies whether to store the new raster object as chunks.	Boolean	Same as the original raster object	N/A.
chunkdim	The dimensions used to store the new raster object as chunks.	String	Same as the original raster object	This field only takes effect when the chunking field is set to true.
chunktable	The name of the chunk table.	String	Null string (''')	By default, a temporary chunk table with a random name is generated to store data. This temporary chunk table is only valid in the current session. To save the new raster object permanently, you must specify you want to create a permanent chunk table in the chunktable field.

Field	Description	Type	Default value	Setting notes
compression	The format used for image compression.	String	Same as the original raster object	Six compression formats are supported: None, JPEG, Zlib, PNG, LZO, and LZ4.
quality	The image quality of the new raster object.	Integer	Same as the original raster object	This field only takes effect in JPEG format.
interleaving	The interleaving type of the new raster object.	String	Same as the original raster object	Valid values: • bip: band interleaved by pixel (BIP) • bil: band interleaved by line (BIL) • bsq: band sequential (BSQ)
endian	The endian format of the new raster object.	String	Same as the original raster object	Valid values: • NDR: specifies little endian format. • XDR: specifies big endian format.

Note If you set the chunktable field to NULL or a null string (''), a temporary chunk table with a random name is generated in the current session to store the new raster object. The temporary chunk table is only valid in the current session, and is deleted immediately after the current session ends. To save the new raster object permanently, you must specify you want to create a permanent chunk table in the chunktable field.

Examples

```
CREATE TABLE if not exists datasource_table(id integer, rast raster);
INSERT INTO datasource_table values(1, ST_ImportFrom('rbt','$(RAST_DATA_DIR)/512_512_1_bsq_
8u_geo.tif', '{}'));
-----------------------------------------------------
-- Method 1: Specify the chunktable field to create a permanent chunk table to store the ne
w raster object.
-----------------------------------------------------
CREATE TABLE rat_rescale_result(id integer, rast raster);
-- If you do not specify the nodata field:
INSERT INTO rat_rescale_result(id, rast)
select 10, ST_Rescale(rast,1024,1024, '{"resample":"Near","nodata":false}','{"chunking":tru
e,"chunkdim":"(256,256,1)","compression":"none","interleaving":"bsq","chunktable":"result_r
bt"}')
from datasource_table
where id =1;
-- If you specify one nodata value in the nodataValue field and pixels with nodata values a
re resampled:
INSERT INTO rat_rescale_result(id, rast)
select 11, ST_Rescale(rast,1024,1024, '{"resample":"Near","nodata":true,"nodatavalue":255}'
,'{"chunking":true,"chunkdim":"(256,256,1)","compression":"none","interleaving":"bsq","chun
ktable":"result_rbt"}')
from datasource_table
where id =1;
-- If you specify more than one nodata value in the nodataValue field:
INSERT INTO rat_rescale_result(id, rast)
select 12, ST_Rescale(rast,1024,1024, '{"resample":"Near","nodata":false,"nodatavalue":[255
,255,255]}','{"chunking":true,"chunkdim":"(256,256,1)","compression":"none","interleaving":
"bsq","chunktable":"result_rbt"}')
from datasource_table
where id =1;
-----------------------------------------------------
-- Method 2: Do not specify the chunktable field. The new raster object is saved to a tempo
rary chunk table with a random name and can only be used for nested computations in the cur
rent session.
-----------------------------------------------------
CREATE TEMP TABLE rat_rescale_result_temp(id integer, rast raster);
INSERT INTO rat_rescale_result_temp(id, rast)
select 1, ST_Rescale(rast,1024,1024,'{"resample":"Near","nodata":false, "nodataValue":[255,
255,255]}','{"chunking":true,"chunkdim":"(256,256,1)","compression":"none","interleaving":"
bsq"}')
from datasource_table
where id =1;
```

5.13.4. ST_Resize

This topic describes the ST_Resize function. This function resizes the pixel range of a raster object but leaves the geospatial data range unchanged.

Syntax

```
raster ST_Resize(raster rast,
           integer outWidth,
           integer outHeight,
            cstring processexpr default '',
            cstring storageOption default '')
```

Parameters

Parameter	Description
rast	The original raster object whose pixel range you want to resize.
outWidth	The width of the new raster object in pixels.
outHeight	The height of the new raster object in pixels.
processExpr	The JSON string that specifies how to resample pixels and process nodata values.
storageOption	The JSON string that specifies how to store the new raster object.

Each child JSON object in the JSON string specified by the processExpr parameter represents a field. The following table describes these fields.

Field	Description	Type	Default value	Setting notes
resample	The method used to resample pixels.	text	Near	Valid values: Near | Average | Cubic | Bilinear.
nodata	Specifies whether nodata values from the original raster object are valid.	Boolean	false	• If you set this field to true, nodata values are valid and pixels with nodata values are not resampled. • If you set this field to false, nodata values are invalid and pixels with nodata values are resampled.

Field	Description	Type	Default value	Setting notes
nodataValue	The new nodata value specified based on the bands in the new raster object.	float8 float8[]	NULL	You can specify one or more nodata values in the nodataValue field. • If you specify one nodata value, this nodata value is used for all bands in the new raster object. • If you specify more than one nodata value, the number of specified nodata values must be the same as the number of bands in the new raster object.

Note Exercise caution when you specify the nodata and nodataValue fields. If the original raster object does not have pixels with nodata values, we recommend that you set the nodata field to false and do not specify the nodataValue field. Otherwise, image artifacts may occur.

The following table describes fields in the storageOption parameter.

Field	Description	Type	Default value	Setting notes
chunking	Specifies whether to store the new raster object as chunks.	Boolean	Same as the original raster object	N/A.
chunkdim	The dimensions used to store the new raster object as chunks.	String	Same as the original raster object	This field only takes effect when the chunking field is set to true.
chunktable	The name of the chunk table.	String	Null string (''')	By default, a temporary chunk table with a random name is generated to store data. This temporary chunk table is only valid in the current session. To save the new raster object permanently, you must specify you want to create a permanent chunk table in the chunktable field.

Field	Description	Type	Default value	Setting notes
compression	The format used for image compression.	String	Same as the original raster object	Six compression formats are supported: None, JPEG, Zlib, PNG, LZO, and LZ4.
quality	The image quality of the new raster object.	Integer	Same as the original raster object	This field only takes effect in JPEG format.
interleaving	The interleaving type of the new raster object.	String	Same as the original raster object	Valid values: • bip: band interleaved by pixel (BIP) • bil: band interleaved by line (BIL) • bsq: band sequential (BSQ)
endian	The endian format of the new raster object.	String	Same as the original raster object	Valid values: • NDR: specifies little endian format. • XDR: specifies big endian format.

Note If you set the chunktable field to NULL or a null string (''), a temporary chunk table with a random name is generated in the current session to store the new raster object. The temporary chunk table is only valid in the current session, and is deleted immediately after the current session ends. To save the new raster object permanently, you must specify you want to create a permanent chunk table in the chunktable field.

Examples

```
CREATE TABLE if not exists datasource_table(id integer, rast raster);
INSERT INTO datasource_table values(1, ST_ImportFrom('rbt','$(RAST_DATA_DIR)/512_512_1_bsq_
8u_geo.tif', '{}'));
-----------------------------------------------------
-- Method 1: Specify the chunktable field to create a permanent chunk table to store the ne
w raster object.
-----------------------------------------------------
CREATE TABLE rat_resize_result(id integer, rast raster);
-- If you do not specify the nodata field:
INSERT INTO rat_resize_result(id, rast)
select 10, ST_Resize(rast,1024,1024, '{"resample":"Near","nodata":false}','{"chunking":true
,"chunkdim":"(256,256,1)","compression":"none","interleaving":"bsq","chunktable":"result_rb
t"}')
from datasource_table
where id =1;
-- If you specify one nodata value in the nodataValue field and pixels with nodata values a
re resampled:
INSERT INTO rat_resize_result(id, rast)
select 11, ST_Resize(rast,1024,1024, '{"resample":"Near","nodata":true,"nodatavalue":255}',
'{"chunking":true,"chunkdim":"(256,256,1)","compression":"none","interleaving":"bsq","chunk
table":"result_rbt"}')
from datasource_table
where id =1;
-- If you specify more than one nodata value in the nodataValue field:
INSERT INTO rat_resize_result(id, rast)
select 12, ST_Resize(rast,1024,1024, '{"resample":"Near","nodata":false,"nodatavalue":[255,
255,255]}','{"chunking":true,"chunkdim":"(256,256,1)","compression":"none","interleaving":"
bsq","chunktable":"result_rbt"}')
from datasource_table
where id =1;
-----------------------------------------------------
-- Method 2: Do not specify the chunktable field. The new raster object is saved to a tempo
rary chunk table with a random name and can only be used for nested computations in the cur
rent session.
-----------------------------------------------------
CREATE TEMP TABLE rat_resize_result_temp(id integer, rast raster);
INSERT INTO rat_resize_result_temp(id, rast)
select 1, ST_Resize(rast,1024,1024,'{"resample":"Near","nodata":false, "nodataValue":[255,2
55,255]}','{"chunking":true,"chunkdim":"(256,256,1)","compression":"none","interleaving":"b
sq"}')
from datasource_table
where id =1;
```

5.13.5. ST_RPCRectify

Rectifies a raster data file based on the related rational polynomial coefficients (RPC) parameters and returns the updated raster data file.

Syntax

```
raster ST_RPCRectify(raster rast,
 raster dem default NULL,
                    cstring rectifyOption default '{}',
                    cstring storageOption default '{}',
   cstring parallelOption default '{}')
```

Parameters

Parameter	Description
rast	The raster object to be rectified.
dem	A digital elevation model (DEM) that is referenced when this function rectifies the raster object. Default value: null.
rectifyOption	The options to be rectified. The value is a JSON-formatted string.
storageOption	The storage options for the output. The value is a JSON-formatted string.
parallelOption	The parallel options. The value is a JSON-formatted string.

Description

Note The raster object must contain RPC information.

rectifyOption is an array of strings in the JSON format. The following table describes the parameters of each JSON object.

Parameter	Description	Type	Default value	Description
resample	The method that is used to resample pixels.	string	'Near'	Valid values: • Near • Average • Cubic • Bilinear
outSrid	The spatial reference information of the new raster object.	integer	The value is the same as the value for the original raster object.	None

Parameter	Description	Type	Default value	Description
nodataValue	The new nodata value that is specified based on the bands of the new raster object.	float8 float8[]	NULL	You can specify a value or an array in the nodataValue parameter. • If you specify a value, the value is used for all bands in the new raster object. • If you specify an array [0, 0, 0], the number of array elements must be the same as the number bands in the new raster object. • If you do not specify this parameter, the system sets this parameter to the value of the nodataValue parameter for the original raster object. If the original nodataValue parameter is not specified, the system sets this parameter to 0.
RPC_DEM_MISSING_VALUE	The height value when the dem parameter is set to the new nodata value or the raster data is not included in the valid values of the dem parameter.	float8	0	None

The following table describes the parameters of the storageOption parameter.

Parameter	Description	Type	Default value	Description
chunking	Specifies whether to store the data as chunks.	boolean	The value is the same as the value for the original raster object.	None

Parameter	Description	Type	Default value	Description
chunkdim	The dimensions that are used to chunk data.	string	The value is the same as the value for the original raster object.	This parameter takes effect only when you set the chunking parameter to true.
chunktable	The name of the chunk table.	string	`''`	If you specify the value `''`, the system create a temporary chunk table that has a random name in the current session. This chunk table is used to store the new raster object. The temporary chunk table is available only in the current session. After the session ends, the temporary chunk table is also deleted. If you want to save the new raster object, you must specify a name for the chunk table. If the specified parallel method does not support chunk tables that are created by anonymous users, you must create a chunk table in advance.
compression	The format into which you want to compress the data of the raster object.	string	The value is the same as the value for the original raster object.	Valid values: • NONE • JPEG • ZLIB • PNG • LZO • LZ4 • JP2K
quality	The quality of compression.	integer	The value is the same as the value for the original raster object.	This parameter is available only for JPEG and compression algorithms.

Parameter	Description	Type	Default value	Description
interleaving	The method that is used to interleave the data of the raster object.	string	The value is the same as the value for the original raster object.	Valid values: • bip: band interleaved by pixel (BIP) • bil: band interleaved by line (BIL) • bsq: band sequential (BSQ)
endian	The endian format of the new raster object.	string	The value is the same as the value for the original raster object.	Valid values: • NDR: little endian • XDR: big endian

Examples:

```
CREATE TABLE if not exists raster_table(id integer, rast raster);
-- The RPC raster data.
INSERT INTO raster_table values(1, ST_ImportFrom('t_chunk','<RAST_DATA_DIR>/rpc.tif', '{}')
);
-- The DEM data.
INSERT INTO raster_table values(2, ST_ImportFrom('t_chunk','<RAST_DATA_DIR>/rpc_dem.tif', '
{}'));
CREATE TABLE raster_result(id integer, rast raster);
-- The dem parameter is not specified. The system samples data by using the Nearest Neighbo
r Search (NNS) method.
INSERT INTO raster_result(id, rast)
select 10, ST_RPCRectify(rast,
                       NULL,
                         '{"resample":"Near"}',
                         '{"chunking":true,"chunkdim":"(256,256,1)","interleaving":"bsq","c
hunktable":"t_chunk"}')
from raster_table
where id =1;
-- The dem parameter is specified. The system samples data by using the NNS method.
INSERT INTO raster_result(id, rast)
select 11, ST_RPCRectify(rast,
                       (SELECT rast FROM raster_table WHERE id = 2),
                         '{"resample":"Near", "outSrid": 32635}',
                         '{"chunking":true,"chunkdim":"(256,256,1)","interleaving":"bsq","c
hunktable":"t_chunk"}',
                         '{"parallel": 8}')
from raster_table
where id =1;
```

5.13.6. ST_LinearStretch

This function is used to linearly stretch a raster and return a new raster.

Syntax

- Syntax 1

```
raster ST_LinearStretch(raster raster_obj ,
                        integer pyramidLevel default 0,
                        cstring bands default '',
                        integer minRatio default 0,
                        integer maxRatio default 100,
                        cstring storageOption  default '',
                        cstring parallelOption default '{}')
```

- Syntax 2

```
raster ST_LinearStretch(raster raster_obj ,
                        float8[] minValues,
                        float8[] maxValues,
                        integer pyramidLevel default 0,
                        cstring bands default '',
                        cstring storageOption  default '',
                        cstring parallelOption default '{}')
```

Parameters

Parameter	Description
raster_obj	The raster that you want to query.
pyramidLevel	The pyramid level that you want to stretch. Valid values start from 0. Default value: 0.
bands	The IDs of the bands that you want to stretch. Examples: `'0-2'` and `'1,2,3'`. Valid band IDs start from `0`. Default value: `''`. The default value specifies all bands of the raster.
storageOption	A JSON string that consists of storage options. For more information, see the "storageOption" table of this topic.
parallelOption	A JSON string that consists of parallel operation options. For more information, see the " parallelOption" table of this topic.
minRatio	The minimum percentile based on which the raster is stretched. Valid values: 0 to 100.
maxRatio	The maximum percentile based which the raster is stretched. Valid values: 0 to 100. The value of this parameter must be greater than the value of the minRatio parameter.
minValues	An array that consists of minimum values. The number of minimum values must be the same as the number of bands that you want to stretch.
maxValues	An array that consists of maximum values. The number of maximum values must be the same as the number of bands that you want to stretch.

The following table describes the parameters in storageOption parameter.

Parameter	Description	Type	Default value	Remarks
chunking	Specifies whether to store the data of the new raster as chunks.	boolean	Same as the value for the original raster.	None.

Parameter	Description	Type	Default value	Remarks
chunkdim	The dimensions based on which the system chunks the data of the new raster.	string	Same as the value for the original raster.	This parameter is valid only when you set the `chunking` parameter to true.
chunktable	The name of the chunk table that stores the data of the new raster.	string	`''`	If you specify `''` or the value NULL, a temporary chunk table with a random name is generated to store the data. The temporary chunk table is valid only in the current session. To create a permanent chunk table for the new raster, you must specify a name for the chunk table.
compression	The format in which the system compresses the data of the new raster.	string	Same as the value for the original raster.	Valid values: • NONE • JPEG • ZLIB • PNG • LZO • LZ4
quality	The image quality of the new raster after compression.	integer	Same as the value for the original raster.	Valid values: 1 to 99. This parameter is valid only when you set the **compression** parameter to JPEG.
interleaving	The method that is used to interleave the data of the new raster.	string	Same as the value for the original raster.	Valid values: • bip: band interleaved by pixel (BIP) • bil: band interleaved by line (BIL) • bsq: band sequential (BSQ)
endian	The endian format of the new raster.	string	Same as the value for the original raster.	Valid values: • NDR: little endian • XDR: big endian

Parameter	Description	Type	Default value	Remarks
celltype	The pixel type of the new raster.	string	Same as the value for the original raster.	Valid values: • 1bb: 1-bit Boolean value • 2bui: 2-bit unsigned integer • 4bui: 4-bit unsigned integer • 8bsi: 8-bit signed integer • 8bui: 8-bit unsigned integer • 16bsi: 16-bit signed integer • 16bui: 16-bit unsigned integer • 32bsi: 32-bit signed integer • 32bui: 32-bit unsigned integer • 64bsi: 64-bit signed integer • 64bui: 64-bit unsigned integer • 32bf: 32-bit floating number • 64bf: 64-bit floating number

The following table describes the parameters in the parallelOptions parameter.

Parameter	Description	Type	Default value	Remarks
parallel	The degree of parallelism that is allowed.	integer	ganos.parallel.degree	Valid values: 1 to 64.

Description

This function linearly stretches a raster and returns a new raster.

- Method 1: Use the ST_BuildPercentiles function or the ST_ComputeStatistics function to calculate percentiles. Then, stretch the original raster based on the percentiles.

  For example, the pixel values in the histogram are within the range of 33 to 206, and the minimum percentile and the maximum percentile are set to 2%. This means that 2% of the pixel values below the minimum percentile are within the range of 33 to 45, and 2% of the pixel values above the maximum percentile are within the range of 198 to 206. In this case, the pixel values within the range of 33 to 45 are converted into 0, the pixel values within the range of 198 to 206 are converted into 255, and all the other pixel values in the histogram are redistributed within the range of 0 to 255.

- Method 2: Stretch the original raster based on the minimum extent that is specified by the minValues parameter and the maximum extent that is specified by the maxValues parameter.

  For example, in an 8-bit data set, a minimum value can be 35, and a maximum value can be 206. This function distributes the pixel values of the original raster within the range of 0 to 255 across the histogram. This way, the brightness and contrast of the image are improved, and the elements in the image are easier to distinguish.

Examples

Example 1: Linearly stretch a raster whose minimum percentage and maximum percentage are set to 2%.

```
INSERT INTO raster_table
SELECT 100, ST_LinearStretch(rast,
                             1,
                             '0-2',
                             2,  -- min ratio
                             98, -- max ratio
                             '{"chunktable":"chunk_table", "chunking":true}',
                             '{"parallel":4}')
FROM raster_table
WHERE id=1;
```

Example 2: Stretch a raster to the range of 100 to 200 on all specified bands.

```
INSERT INTO raster_table
SELECT 200, ST_LinearStretch(rast,
                             ARRAY[100,100, 100]::float8[],  -- min values
                             ARRAY[200,200, 200]::float8[],  -- max values
                             0,
                             '0-2',
                             '{"chunktable":"chunk_table", "chunking":true}',
                             '{"parallel":4}')
FROM raster_table
WHERE id=1;
```

5.14. Operators

5.14.1. Equal to operator (=)

This topic describes the equal to operator (=). This operator determines whether the universally unique identifier (UUID) of the first specified raster object is equal to the UUID of the second.

Syntax

```
bool  Operator =(raster rast1, raster rast2);
```

Parameters

Parameter	Description
rast1	The raster object 1.
rast2	The raster object 2.

Note This operator only compares the UUIDs of two specified raster objects. This operator does not compare their spatial extents or pixel types. This operator is used only for operations such as union and B-tree.

Examples

```
SELECT a.rast = b.rast
FROM tbl_a a, tbl_b b
WHERE a.id = b.id
```

5.14.2. Greater than operator (>)

This topic describes the greater than operator (>). This operator determines whether the universally unique identifier (UUID) of the first specified raster object is greater than the UUID of the second.

Syntax

```
bool  Operator >(raster rast1, raster rast2);
```

Parameters

Parameter	Description
rast1	The raster object 1.
rast2	The raster object 2.

Note This operator only compares the UUIDs of two specified raster objects. This operator does not compare their spatial extents or pixel types. This operator is used only for operations such as union and B-tree.

Examples

```
SELECT a.rast > b.rast
FROM tbl_a a, tbl_b b
WHERE a.id = b.id
```

5.14.3. Less than operator (<)

This topic describes the less than operator (<). This operator determines whether the universally unique identifier (UUID) of the first specified raster object is less than the UUID of the second.

Syntax

```
bool  Operator <(raster rast1, raster rast2);
```

Parameters

Parameter	Description
rast1	The raster object 1.
rast2	The raster object 2.

Note This operator only compares the UUIDs of two specified raster objects. This operator does not compare their spatial extents or pixel types. This operator is used only for operations such as union and B-tree.

Examples

```
SELECT a.rast < b.rast
FROM tbl_a a, tbl_b b
WHERE a.id = b.id
```

5.14.4. Greater than or equal to operator (>=)

This topic describes the greater than or equal to operator (>=). This operator determines whether the universally unique identifier (UUID) of the first specified raster object is greater than or equal to the UUID of the second.

Syntax

```
bool  Operator >=(raster rast1, raster rast2);
```

Parameters

Parameter	Description
rast1	The raster object 1.
rast2	The raster object 2.

Note This operator only compares the UUIDs of two specified raster objects. This operator does not compare their spatial extents or pixel types. This operator is used only for operations such as union and B-tree.

Examples

```
SELECT a.rast >= b.rast
FROM tbl_a a, tbl_b b
WHERE a.id = b.id
```

5.14.5. Less than or equal to operator (<=)

This topic describes the less than or equal to operator (<=). This operator determines whether the universally unique identifier (UUID) of the first specified raster object is less than or equal to the UUID of the second.

Syntax

```
bool  Operator <=(raster rast1, raster rast2);
```

Parameters

Parameter	Description
rast1	The raster object 1.
rast2	The raster object 2.

Note This operator only compares the UUIDs of two specified raster objects. This operator does not compare their spatial extents or pixel types. This operator is used only for operations such as union and B-tree.

Examples

```
SELECT a.rast <= b.rast
FROM tbl_a a, tbl_b b
WHERE a.id = b.id
```

5.15. Functions for Identifying Spatial Relationships

5.15.1. ST_Intersects

This topic describes the ST_Intersects function. This function identifies the spatial relationship between two raster objects or between a raster object and a geometric object to determine whether the first specified object intersects with the second.

Syntax

```
bool  ST_Intersects(raster rast1, raster rast2);
bool  ST_Intersects(raster rast, geometry geom);
bool  ST_Intersects(geometry geom, raster rast);
```

Parameters

Parameter	Description
rast1	The raster object 1.
rast2	The raster object 2.
rast	A raster object.
geom	A geometric object.

Example

```
SELECT a.id
FROM tbl_a a, tbl_b b
WHERE ST_Intersects(a.rast, b.rast)
```

5.15.2. ST_Contains

This topic describes the ST_Contains function. This function identifies the spatial relationship between two raster objects or between a raster object and a geometric object to determine whether the first specified object completely contains the second.

Syntax

```
bool  ST_Contains(raster rast1, raster rast2);
bool  ST_Contains(raster rast, geometry geom);
bool  ST_Contains(geometry geom, raster rast);
```

Parameters

Parameter	Description
rast1	The raster object 1.
rast2	The raster object 2.
rast	A raster object.
geom	A geometric object.

Example

```
SELECT a.id
FROM tbl_a a, tbl_b b
WHERE ST_Contains(a.rast, b.rast)
```

5.15.3. ST_ContainsProperly

This topic describes the ST_ContainsProperly function. This function identifies the spatial relationship between two raster objects or between a raster object and a geometric object to determine whether the first specified object properly contains the second.

Syntax

```
bool  ST_ContainsProperly(raster rast1, raster rast2);
bool  ST_ContainsProperly(raster rast, geometry geom);
bool  ST_ContainsProperly(geometry geom, raster rast);
```

Parameters

Parameter	Description
rast1	The raster object 1.
rast2	The raster object 2.
rast	A raster object.
geom	A geometric object.

Example

```
SELECT a.id
FROM tbl_a a, tbl_b b
WHERE ST_ContainsProperly(a.rast, b.rast)
```

5.15.4. ST_Covers

This topic describes the ST_Covers function. This function identifies the spatial relationship between two raster objects or between a raster object and a geometric object to determine whether the first specified object covers the second.

Syntax

```
bool  ST_Covers(raster rast1, raster rast2);
bool  ST_Covers(raster rast, geometry geom);
bool  ST_Covers(geometry geom, raster rast);
```

Parameters

Parameter	Description
rast1	The raster object 1.
rast2	The raster object 2.
rast	A raster object.
geom	A geometric object.

Example

```
SELECT a.id
FROM tbl_a a, tbl_b b
WHERE ST_Covers(a.rast, b.rast)
```

5.15.5. ST_CoveredBy

This topic describes the ST_CoveredBy function. This function identifies the spatial relationship between two raster objects or between a raster object and a geometric object to determine whether the first specified object is covered by the second.

Syntax

```
bool  ST_CoveredBy(raster rast1, raster rast2);
bool  ST_CoveredBy(raster rast, geometry geom);
bool  ST_CoveredBy(geometry geom, raster rast);
```

Parameters

Parameter	Description
rast1	The raster object 1.
rast2	The raster object 2.
rast	A raster object.
geom	A geometric object.

Example

```
SELECT a.id
FROM tbl_a a, tbl_b b
WHERE ST_CoveredBy(a.rast, b.rast)
```

5.15.6. ST_Disjoint

This topic describes the ST_Disjoint function. This function identifies the spatial relationship between two raster objects or between a raster object and a geometric object to determine whether the first specified object does not touch or intersect the second.

Syntax

```
bool  ST_Disjoint(raster rast1, raster rast2);
bool  ST_Disjoint(raster rast, geometry geom);
bool  ST_Disjoint(geometry geom, raster rast);
```

Parameters

Parameter	Description
rast1	The raster object 1.
rast2	The raster object 2.
rast	A raster object.
geom	A geometric object.

Example

```
SELECT a.id
FROM tbl_a a, tbl_b b
WHERE ST_Disjoint(a.rast, b.rast)
```

5.15.7. ST_overlaps

This topic describes the ST_overlaps function. This function identifies the spatial relationship between two raster objects or between a raster object and a geometric object to determine whether the first specified object overlaps with the second.

Syntax

```
bool  ST_overlaps(raster rast1, raster rast2);
bool  ST_overlaps(raster rast, geometry geom);
bool  ST_overlaps(geometry geom, raster rast);
```

Parameters

Parameter	Description
rast1	The raster object 1.
rast2	The raster object 2.
rast	A raster object.

Parameter	Description
geom	A geometric object.

Example

```
SELECT a.id
FROM tbl_a a, tbl_b b
WHERE ST_Contains(a.rast, b.rast)
```

5.15.8. ST_Touches

This topic describes the ST_Touches function. This function identifies the spatial relationship between two raster objects or between a raster object and a geometric object to determine whether the first specified object touches the second.

Syntax

```
bool  ST_Touches(raster rast1, raster rast2);
bool  ST_Touches(raster rast, geometry geom);
bool  ST_Touches(geometry geom, raster rast);
```

Parameters

Parameter	Description
rast1	The raster object 1.
rast2	The raster object 2.
rast	A raster object.
geom	A geometric object.

Example

```
SELECT a.id
FROM tbl_a a, tbl_b b
WHERE ST_Touches(a.rast, b.rast)
```

5.15.9. ST_Within

This topic describes the ST_Within function. This function identifies the spatial relationship between two raster objects or between a raster object and a geometric object to determine whether the first specified object is completely within the second.

Syntax

```
bool  ST_Within(raster rast1, raster rast2);
bool  ST_Within(raster rast, geometry geom);
bool  ST_Within(geometry geom, raster rast);
```

Parameters

Parameter	Description
rast1	The raster object 1.
rast2	The raster object 2.
rast	A raster object.
geom	A geometric object.

Example

```
SELECT a.id
FROM tbl_a a, tbl_b b
WHERE ST_Within(a.rast, b.rast)
```

5.16. Auxiliary functions

5.16.1. ST_CreateChunkTable

This topic describes the ST_CreateChunkTable function. This function is used to create a database chunk table.

Syntax

```
boolean ST_CreateChunkTable(cstring tableName)
```

Parameters

Parameter	Description
tableName	The name of the database chunk table. The name must meet the following requirements: • The name must start with a letter and end with a letter or digit. • The name can contain lowercase letters, digits, and underscores (_). • The name must be 2 to 64 characters in length. • The name must be unique in the current database.

Examples:

```
SELECT ST_CreateChunkTable('t_chunk');
____________________________________
t
```

5.16.2. ST_CheckGPU

This function checks whether a GPU is available in the current environment.

Syntax

```
text ST_CheckGPU();
```

Description

This function checks whether a GPU can be identified in the current runtime environment of ApsaraDB for RDS.

Examples

```
select st_checkgpu();
-----------------------------------------------------------------------------------------
 [GPU prop]multiProcessorCount=20; sharedMemPerBlock=49152; maxThreadsPerBlock=1024
(1 row)
```

5.16.3. ST_AKId

Queries the AccessKey ID that is used to access a raster object stored in Alibaba Cloud Object Storage Service (OSS). This function can be used with the function that is used to modify multiple AccessKey IDs at a time.

Prerequisites

The raster object is stored in OSS.

Syntax

```
text ST_AKId(raster raster_obj)
```

Parameters

Parameter	Description
raster_obj	The name of the raster object.

Examples

```
SELECT ST_AKId(rast) from raster_table;
     st_akid
------------------
 OSS_ACCESSKEY_ID
```

5.16.4. ST_SetAccessKey

Configures the logon information that is used to access a raster object stored in Alibaba Cloud Object Storage Service (OSS).

Prerequisites

The raster object is stored in OSS.

Syntax

```
raster ST_SetAccessKey(raster raster_obj, text id, text key, bool valid default true)
```

Parameters

Parameter	Description
raster_obj	The name of the raster object.
id	The AccessKey ID that is used to log on to OSS. If you set this parameter to NULL, the previous AccessKey ID is used.
key	The AccessKey secret that is used to log on to OSS. If you set this parameter to NULL, the previous AccessKey secret is used.
valid	Specifies whether to verify the validity of the logon information.

Examples

```
UPDATE raster_table
SET rast = ST_SetAccessKey(rast, 'OSS_ACCESSKEY_ID', 'OSS_ACCESSKEY_SECRET');
SELECT ST_AKId(rast) from raster_table;
     st_akid
------------------
 OSS_ACCESSKEY_ID
```

5.16.5. ST_SetAKId

Sets the AccessKey ID that is used to access a raster object stored in Alibaba Cloud Object Storage Service (OSS).

Prerequisites

The raster object is stored in OSS.

Syntax

```
raster ST_SetAKId(raster raster_obj, text id, bool valid default true)
```

Parameters

Parameter	Description
raster_obj	The name of the raster object.
id	The AccessKey ID that is used to log on to OSS. If you set this parameter to NULL, the previous AccessKey ID is used.
valid	Specifies whether to verify the validity of the logon information.

Examples

```
UPDATE raster_table
SET rast = ST_SetAKId(rast, 'OSS_ACCESSKEY_ID');
SELECT ST_AKId(rast) from raster_table;
     st_akid
------------------
 OSS_ACCESSKEY_ID
```

5.16.6. ST_SetAKSecret

Sets the AccessKey secret that is used to access a raster object stored in Alibaba Cloud Object Storage Service (OSS).

Prerequisites

The raster object is stored in OSS.

Syntax

```
raster ST_SetAKSecret(raster raster_obj, text key, bool valid default true)
```

Parameters

Parameter	Description
raster_obj	The name of the raster object.
key	The AccessKey secret that is used to log on to OSS. If you set this parameter to NULL, the previous AccessKey secret is used.
valid	Specifies whether to verify the validity of the logon information.

Examples

```
UPDATE raster_table
SET rast = ST_SetAKSecret(rast, 'OSS_ACCESSKEY_SECRET');
```

5.16.7. ST_RasterDrivers

This topic describes the ST_RasterDrivers function. This function is used to query all drivers that are supported by raster data sources in Ganos.

Syntax

```
setof record ST_RasterDrivers(out idx integer,
                     out short_name text,
                    out long_name text,
                    out can_read boolean,
                      out can_export boolean,
                    out can_asfile boolean,
                      out create_options text);
```

Parameters

Parameter	Description
idx	The ID of the driver.
short_name	The abbreviated name of the driver.
long_name	The full name of the driver.
can_read	Valid values: • true: The file formats that are supported by the driver can be used in the ST_CreateRast and ST_ImportFrom functions. For more information, see ST_CreateRast and ST_ImportFrom. • false: The file formats that are supported by the driver cannot be used in the ST_CreateRast or ST_ImportFrom function. If you use these file formats in the ST_CreateRast or ST_ImportFrom function, errors are reported. For more information, see ST_CreateRast and ST_ImportFrom.
can_export	Valid values: • true: The file formats that are supported by the driver can be used in the ST_ExportTo function. For more information, see ST_ExportTo. • false: The file formats that are supported by the driver cannot be used in the ST_ExportTo function. If you use these file formats in the ST_ExportTo function, errors are reported. For more information, see ST_ExportTo.

Parameter	Description
can_asfile	Valid values: • true: The file formats that are supported by the driver can be used in the ST_AsDatasetFile function. For more information, see ST_AsDatasetFile. • false: The file formats that are supported by the driver cannot be used in the ST_AsDatasetFile function. If you use these file formats in the ST_AsDatasetFile function, errors are reported. For more information, see ST_AsDatasetFile.
create_options	The options that are included in the create_option parameter in the ST_ExportTo and ST_AsDatasetFile functions. For more information, see ST_ExportTo and ST_AsDatasetFile.

Examples

```
-- Query information about the netCDF driver.
SELECT * FROM
st_rasterdrivers()
where short_name='netCDF';
 idx | short_name |         long_name          | can_read | can_export | can_asfile |
create_options
-----+------------+----------------------------+----------+------------+-----------
 36 | netCDF     | Network Common Data Format | t        | t          | t
| <CreationOptionList>   <Option name='FORMAT' type='string-select'
default='NC'>     <Value>NC</Value>     <Value>NC2</Value>
    <Value>NC4</Value>     <Value>NC4C</Value>   </Option>   <Option
name='COMPRESS' type='string-select' default='NONE'>     <Value>NONE</Value>
<Value>DEFLATE</Value>   </Option>   <Option name='ZLEVEL' ty
pe='int' description='DEFLATE compression level 1-9' default='1'/>   <Option
name='WRITE_BOTTOMUP' type='boolean' default='YES'>   </Option>   <Option
name='WRITE_GDAL_TAGS' type='boolean' default='YES'>   </Opt
ion>   <Option name='WRITE_LONLAT' type='string-select'>     <Value>YES</Value>
<Value>NO</Value>     <Value>IF_NEEDED</Value>   </Option>   <Option
name='TYPE_LONLAT' type='string-select'>     <Value>float<
/Value>     <Value>double</Value>   </Option>   <Option name='PIXELTYPE'
type='string-select' description='only used in Create()'>
<Value>DEFAULT</Value>       <Value>SIGNEDBYTE</Value>   </Option>   <Opti
on name='CHUNKING' type='boolean' default='YES' description='define chunking
when creating netcdf4 file'/>   <Option name='MULTIPLE_LAYERS' type='string-
select' description='Behaviour regarding multiple vector l
ayer creation' default='NO'>       <Value>NO</Value>
<Value>SEPARATE_FILES</Value>       <Value>SEPARATE_GROUPS</Value>   </Option>
<Option name='CONFIG_FILE' type='string' description='Path to a XML con
figuration file (or content inlined)'/></CreationOptionList>
-- Query all drivers that are supported by the ST_ImportFrom and ST_CreateRast functions.
select short_name from
st_rasterdrivers()
where can_read =true;
-- Query all drivers that are supported by the ST_ExportTo function.
select short_name from
st_rasterdrivers()
where can_export =true;
-- Query all drivers that are supported by the ST_AsDatasetFile function.
select short_name from
st_rasterdrivers()
where can_asfile =true;
```

5.17. Variables

5.17.1. ganos.parallel.transaction

This topic describes the ganos.parallel.transaction variable. This variable specifies whether Ganos can commit and roll parallel transactions back along with the main transaction while parallel operations are run.

Data type

String

Value

- **transaction_commit**: specifies that Ganos can commit or roll parallel transactions back along with the main transaction. This is the default value.
- **fast_commit**: specifies that Ganos cannot commit or roll parallel transactions back along with the main transaction.

Examples:

```
SET ganos.parallel.transaction  = transaction_commit;
```

5.17.2. ganos.parallel.degree

This topic describes the ganos.parallel.degree variable. If you do not specify the degree of parallelism, Ganos uses the value of this variable as the default degree of parallelism. You can set the default parallelism by using the ganos.parallel.degree variable.

Data type

Integer

Value

Valid values: 1 to 64. Default value: 1.

Examples:

```
SET ganos.parallel.degree  = 4;
```

5.17.3. ganos.raster.calculate_md5

This topic describes the ganos.raster.calculate_md5 variable. This variable specifies whether PolarDB calculates the MD5 hash string and saves the string to the metadata when a raster object is imported.

Data type

Boolean

Value

- **true**: specifies that PolarDB calculates the MD5 hash string and saves the string to the metadata when a raster object is imported.
- **false**: specifies that PolarDB does not calculate the MD5 hash string when a raster object is imported. This is the default value.

Example

```
set ganos.raster.calculate_md5 = true;
```

5.17.4. ganos.raster.md5sum_chunk_size

This topic describes the ganos.raster.md5sum_chunk_size variable. This variable specifies the amount of data that can be cached for each read operation when PolarDB calculates the MD5 hash string.

Data type

Integer

Value

Valid values: 10 to 100. Unit: MB. Default value: 10.

Example

```
set ganos.raster.md5sum_chunk_size = 20;
```

5.17.5. ganos.raster.mosaic_must_same_nodata

This topic describes the ganos.raster.mosaic_must_same_nodata variable. This variable specifies whether the NoData values in the data source must be the same during the Mosaic operation.

Data type

Boolean

Value

- **true**: specifies that the NoData values in the data source must be the same. This is the default value.
- **false**: specifies that the NoData values in the data source can be different.

> **Note** PolarDB does not convert the NoData values during the mosaic operation. If you set this parameter to **false**, the semantics of the pixels may change after the mosaic operation.

Example

```
set ganos.raster.mosaic_must_same_nodata = false;
```

5.17.6. ganos.raster.memory_oss_file_max_size

This topic describes the ganos.raster.memory_oss_file_max_size parameter of the ST_ImportFrom function. This parameter specifies the maximum size of the OSS object that can be mapped into an in-memory file in Ganos.

Data type

integer

Value

Valid values: 1 to 2048. Default value: 100. Unit: MB.

Examples

```
SET ganos.raster.memory_oss_file_max_size = 100;
```

6.SpatialRef SQL reference

6.1. ST_SrEqual

This topic describes the ST_SrEqual function, which determines whether two spatial reference systems are the same.

Syntax

```
boolean ST_SrEqual(cstring sr1, cstring sr2, boolean strict default true);
```

Parameters

Parameter	Description
sr1	The string representing spatial reference system 1. It must be an OGC WKT or PROJ.4 string.
sr2	The string representing spatial reference system 2. It must be an OGC WKT or PROJ.4 string.
strict	Specifies whether to use the strict comparison method. The value true specifies to compare the names of the reference ellipsoids if the spatial reference systems are georeferenced. Default value: true.

Description

This function parses the semantics of two spatial reference systems to compare the parameter information of these systems. The compared parameter information includes the projection methods, the reference ellipsoids, and the long and short axes of the reference ellipsoids. If the spatial reference systems are the same, this function returns t. Otherwise, this function returns f.

Example:

```
-- Compare two text-based spatial reference systems.
select ST_srEqual('GEOGCS["WGS 84",DATUM["WGS_1984",SPHEROID["WGS 84",6378137,298.257223563
,AUTHORITY["EPSG","7030"]],AUTHORITY["EPSG","6326"]],PRIMEM["Greenwich",0,AUTHORITY["EPSG",
"8901"]],UNIT["degree",0.0174532925199433,AUTHORITY["EPSG","9122"]],AUTHORITY["EPSG","4326"
]]', '+proj=longlat +datum=WGS84 +no_defs');
st_srequal
------------
 t
 -- Search for the spatial reference system identifier (SRID) of a spatial reference system
from the spatial_ref_sys table.
 select srid from spatial_ref_sys where st_srequal(srtext::cstring, '+proj=longlat +ellps=G
RS80 +no_defs ') limit 1;
 srid
------
 3824
```

6.2. ST_SrReg

This topic describes the ST_SrReg function, which registers a spatial reference system.

Syntax

```
integer ST_SrReg(cstring sr);
integer ST_SrReg(cstring auth_name, integer auth_id, cstring sr);
```

Parameters

Parameter	Description
sr	The string representing the spatial reference system. It must be an OGC WKT or PROJ.4 string.
auth_name	The author who defines the spatial reference system. Example: EPSG.
auth_id	The spatial reference system identifier (SRID) of the spatial reference system.

Description

If the spatial reference system already exists, this function returns the SRID of the existing spatial reference system. If the spatial reference system does not exist, this function inserts a record into the spatial_ref_sys table and returns the SRID of the new spatial reference system.

Example:

```
-- Register a spatial reference system that already exists.
select 4490, ST_srReg('GEOGCS["China Geodetic Coordinate System 2000",DATUM["China_2000",SP
HEROID["CGCS2000",6378137,298.257222101,AUTHORITY["EPSG","1024"]],AUTHORITY["EPSG","1043"]]
,PRIMEM["Greenwich",0,AUTHORITY["EPSG","8901"]],UNIT["degree",0.0174532925199433,AUTHORITY[
"EPSG","9122"]],AUTHORITY["EPSG","4490"]]');
 st_srreg
----------
     4490
-- Register a new spatial reference system.
select ST_srReg('user_defined',100, 'GEOGCS["User Geodetic Coordinate System ",DATUM["China
_2000",SPHEROID["CGCS2000",6378137,298.257222101,AUTHORITY["EPSG","903"]],AUTHORITY["EPSG",
"1043"]],PRIMEM["Greenwich",0,AUTHORITY["EPSG","8901"]],UNIT["degree",0.0174532925199433,AU
THORITY["EPSG","9122"]],AUTHORITY["EPSG","4491"]]');
 st_srreg
----------
    10001
select ST_srReg('+proj=tmerc +lat_0=1 +lon_0=112 +k=1 +x_0=19500001 +y_0=0 +ellps=krass +to
wgs84=24.47,-130.89,-81.56,0,0,0.13,-0.22 +units=m +no_defs');
 st_srreg
----------
   10002
```

6.3. ST_SrFromEsriWkt

This function converts the specified string from the Esri well-known text (WKT) format to the Open Geospatial Consortium (OGC) WKT format.

Syntax

```
cstring ST_SrFromEsriWkt(cstring sr);
```

Parameters

Parameter	Description
sr1	The spatial reference string that uses the Esri WKT format.

Examples

```
SELECT ST_srFromEsriWkt('GEOGCS["China Geodetic Coordinate System 2000",DATUM["D_China_2000
",SPHEROID["CGCS2000",6378137,298.257222101]],PRIMEM["Greenwich",0],UNIT["Degree",0.0174532
92519943295]]');
st_srfromesriwkt

-----------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------
 GEOGCS["China Geodetic Coordinate System 2000",DATUM["China_2000",SPHEROID["CGCS2000",6378
137,298.257222101]],PRIMEM["Greenwich",0],UNIT["Degree",0.017453292519943295]]
```

7.Point cloud SQL reference

7.1. Constructors

7.1.1. ST_makePoint

This function constructs a pcpoint object.

Syntax

```
pcpoint ST_makePoint(integer pcid, float8[] vals);
```

Parameters

Parameter	Description
pcid	The ID of the schema, which comes from the point_cloud_formats table.
float8[]	The float8 array. The number of entries in the array is equal to the number of dimensions specified in the schema.

Examples

```
SELECT ST_makePoint(1, ARRAY[-127, 45, 124.0, 4.0]);
------------------------------------------------
010100000064CEFFFF94110000703000000400
```

7.1.2. ST_makePatch

This function constructs a pcpatch object.

Syntax

```
pcpatch ST_makePatch(integer pcid, float8[] vals);
```

Parameters

Parameter	Description
pcid	The ID of the schema, which comes from the pointcloud_formats table.
float8[]	The float8 array. The number of entries in the array is an integral multiple of the number of dimensions specified in the schema.

Examples

```
SELECT ST_asText(ST_MakePatch(1, ARRAY[-126.99,45.01,1,0, -126.98,45.02,2,0, -126.97,45.03,
3,0]));
--------------------------------------------------
{"pcid":1,"pts":[
 [-126.99,45.01,1,0],[-126.98,45.02,2,0],[-126.97,45.03,3,0]
]}
```

7.1.3. ST_Patch

This function constructs a pcpatch object from a pcpoint array.

Syntax

```
pcpatch ST_Patch(pcpoint[] pts);
```

Parameters

Parameter	Description
pts	The pcpoint array.

Examples

```
INSERT INTO patches (pa)
SELECT ST_Patch(pt) FROM points GROUP BY id/10;
```

7.2. Attribute functions

7.2.1. ST_asText

This function converts a pcpoint or pcpatch object into a JSON-formatted string.

Syntax

```
text ST_asText(pcpatch pp);
text ST_asText(pcpoint pt);
```

Parameters

Parameter	Description
pp	The pcpatch object.
pt	The pcpoint object.

Examples

```
SELECT ST_asText('010100000064CEFFFF94110000703000000400'::pcpoint);
--------------------------------
{"pcid":1,"pt":[-127,45,124,4]}
```

7.2.2. ST_pcID

This function returns the schema ID of a pcpoint or pcpatch object.

Syntax

```
integer ST_pcID(pcpoint pt);
integer ST_pcID(pcpatch pp);
```

Parameters

Parameter	Description
pt	The pcpoint object.
pp	The pcpatch object.

Examples

```
SELECT ST_pcID('010100000064CEFFFF94110000703000000400'::pcpoint);
--------------------------------
1
```

7.2.3. ST_get

This function returns the values of attribute dimensions in a pcpoint object.

Syntax

```
float8[] ST_get(pcpoint pc);
numeric ST_get(pcpoint pc, text dimname);
```

Parameters

Parameter	Description
pc	The pcpoint object.
dimname	The name of the specified attribute dimension.

Examples

```
SELECT ST_Get('010100000064CEFFFF94110000703000000400'::pcpoint, 'Intensity');
-------------------
4
SELECT ST_Get('010100000064CEFFFF94110000703000000400'::pcpoint);
--------------------
{-127,45,124,4}
```

7.2.4. ST_numPoints

This function returns the number of pcpoint objects in a pcpatch object.

Syntax

```
integer ST_numPoints(pcpatch pc);
```

Parameters

Parameter	Description
pc	The pcpatch object.

Examples

```
SELECT ST_NumPoints(pa) FROM patches LIMIT 1;
---------------------------
9
```

7.2.5. ST_summary

This function returns the summary of a pcpatch object.

Syntax

```
text ST_summary(pcpatch pc);
```

Parameters

Parameter	Description
pc	The pcpatch object.

Examples

```
SELECT ST_Summary(pa) FROM patches LIMIT 1;
-----------------------------------
{"pcid":1, "npts":9, "srid":4326, "compr":"dimensional","dims":[{"pos":0,"name":"X","size":
4,"type":"int32_t","compr":"sigbits","stats":{"min":-126.99,"max":-126.91,"avg":-126.95}},{
"pos":1,"name":"Y","size":4,"type":"int32_t","compr":"sigbits","stats":{"min":45.01,"max":4
5.09,"avg":45.05}},{"pos":2,"name":"Z","size":4,"type":"int32_t","compr":"sigbits","stats":
{"min":1,"max":9,"avg":5}},{"pos":3,"name":"Intensity","size":2,"type":"uint16_t","compr":"
rle","stats":{"min":0,"max":0,"avg":0}}]}
```

7.3. Object operations

7.3.1. ST_boundingDiagonalGeometry

This function returns the diagonal of the bounding box of a pcpatch object as a geometry object.

Syntax

```
geometry ST_boundingDiagonalGeometry(pcpatch pc);
```

Parameters

Parameter	Description
pc	The pcpatch object.

Description

The diagonal of a bounding box is a 2D linestring object of Ganos Geometry. This function can be used to create an index on a pcpatch object column.

Examples

```
SELECT ST_AsText(ST_BoundingDiagonalGeometry(pa)) FROM patches;
                  st_astext
-------------------------------------------------
LINESTRING Z (-126.99 45.01 1,-126.91 45.09 9)
LINESTRING Z (-126 46 100,-126 46 100)
LINESTRING Z (-126.2 45.8 80,-126.11 45.89 89)
LINESTRING Z (-126.4 45.6 60,-126.31 45.69 69)
LINESTRING Z (-126.3 45.7 70,-126.21 45.79 79)
LINESTRING Z (-126.8 45.2 20,-126.71 45.29 29)
LINESTRING Z (-126.5 45.5 50,-126.41 45.59 59)
LINESTRING Z (-126.6 45.4 40,-126.51 45.49 49)
LINESTRING Z (-126.9 45.1 10,-126.81 45.19 19)
LINESTRING Z (-126.7 45.3 30,-126.61 45.39 39)
LINESTRING Z (-126.1 45.9 90,-126.01 45.99 99)
CREATE INDEX ON patches USING GIST(ST_BoundingDiagonalGeometry(patch) gist_geometry_ops_nd)
;
```

7.3.2. ST_compress

This function compresses a pcpatch object based on the specified compression method.

Syntax

```
pcpatch ST_compress(pcpatch pc, text global_compression_schema default '', text compression
_config default '');
```

Parameters

Parameter	Description
pc	The pcpatch object.
global_compression_schema	The compression schema.
compression_config	The compression configuration item that specifies the compression algorithm for specific dimensions.

Description

Valid values of the global_compression_schema parameter include:

```
auto            -- determined by pcid
dimension
laz              -- no compression config supported
ght             -- is discarded
```

If the global_compression_schema parameter is set to dimension, valid values of the compression_config parameter include:

```
auto -- determined automatically, from values stats
zlib -- deflate compression
sigbits -- significant bits removal
rle -- run-length encoding
```

Examples

```
SELECT ST_asText(ST_Compress(ST_MakePatch(1, ARRAY[-126.99,45.01,1,0, -126.98,45.02,2,0, -1
26.97,45.03,3,0])));
--------------------------------------------------
{"pcid":1,"pts":[
 [-126.99,45.01,1,0],[-126.98,45.02,2,0],[-126.97,45.03,3,0]
]}
```

7.3.3. ST_explode

This function separates a pcpatch object into pcpoint objects in multiple rows.

Syntax

```
setof[pcpoint] ST_explode(pcpatch pc);
```

Parameters

Parameter	Description
pc	The pcpatch object.

Examples

```
SELECT ST_AsText(ST_Explode(pa)), id FROM patches WHERE id = 7;
-----------------------------------------
              st_astext                | id
-----------------------------------------+----
 {"pcid":1,"pt":[-126.5,45.5,50,5]}    |  7
 {"pcid":1,"pt":[-126.49,45.51,51,5]} |  7
 {"pcid":1,"pt":[-126.48,45.52,52,5]} |  7
 {"pcid":1,"pt":[-126.47,45.53,53,5]} |  7
 {"pcid":1,"pt":[-126.46,45.54,54,5]} |  7
 {"pcid":1,"pt":[-126.45,45.55,55,5]} |  7
 {"pcid":1,"pt":[-126.44,45.56,56,5]} |  7
 {"pcid":1,"pt":[-126.43,45.57,57,5]} |  7
 {"pcid":1,"pt":[-126.42,45.58,58,5]} |  7
 {"pcid":1,"pt":[-126.41,45.59,59,5]} |  7
```

7.3.4. ST_patchAvg

This function computes the average values of all the attribute dimensions for all the pcpoint objects in a pcpatch object and returns a new pcpoint object whose attribute dimensions are set to these average values.

Syntax

```
pcpoint ST_patchAvg(pcpatch pc);
```

Parameters

Parameter	Description
pc	The pcpatch object.

Examples

```
SELECT ST_AsText(ST_PatchAvg(pa)) FROM patches WHERE id = 7;
-------------------------------------
{"pcid":1,"pt":[-126.46,45.54,54.5,5]}
```

7.3.5. ST_envelopeGeometry

This function returns the bounding box of a pcpatch object as a geometry object.

Syntax

```
geometry ST_envelopeGeometry(pcpatch pc);
```

Parameters

Parameter	Description
pc	The pcpatch object.

Description

A bounding box is a 2D geometry object of Ganos Geometry.

Examples

```
SELECT ST_AsText(ST_EnvelopeGeometry(pa)) FROM patches LIMIT 1;
------------------------------
POLYGON((-126.99 45.01,-126.99 45.09,-126.91 45.09,-126.91 45.01,-126.99 45.01))
CREATE INDEX ON patches USING GIST(ST_EnvelopeGeometry(patch));
```

7.3.6. ST_filterGreaterThan

This function filters all pcpoint objects to obtain the pcpoint objects whose values of the specified attribute dimension are greater than a fixed value in a pcpatch object and returns a new pcpatch object that is composed of these pcpoint objects.

Syntax

```
pcpatch ST_filterGreaterThan(pcpatch pc, text dimname, float8 value);
```

Parameters

Parameter	Description
pc	The pcpatch object.
dimname	The name of the specified attribute dimension.
value	The fixed value of the attribute dimension.

Examples

```
SELECT ST_AsText(ST_FilterGreaterThan(pa, 'y', 45.57)) FROM patches WHERE id = 7;
-----------------------------------------------------------
 {"pcid":1,"pts":[[-126.42,45.58,58,5],[-126.41,45.59,59,5]]}
```

7.3.7. ST_filterLessThan

This function filters all pcpoint objects to obtain the pcpoint objects whose values of the specified attribute dimension are smaller than a fixed value in a pcpatch object and returns a new pcpatch object that is composed of these pcpoint objects.

Syntax

```
pcpatch ST_filterLessThan(pcpatch pc, text dimname, float8 value);
```

Parameters

Parameter	Description
pc	The pcpatch object.
dimname	The name of the specified attribute dimension.
value	The fixed value of the attribute dimension.

Examples

```
SELECT ST_AsText(ST_FilterLessThan(pa, 'y', 45.60)) FROM patches WHERE id = 7;
-----------------------------------------------------------
 {"pcid":1,"pts":[[-126.42,45.58,58,5],[-126.41,45.59,59,5]]}
```

7.3.8. ST_filterEquals

This function filters all pcpoint objects to obtain the pcpoint objects whose values of the specified attribute dimension are equal to a fixed value in a pcpatch object and returns a new pcpatch object that is composed of these pcpoint objects.

Syntax

```
pcpatch ST_filterEquals(pcpatch pc, text dimname, float8 value);
```

Parameters

Parameter	Description
pc	The pcpatch object.
dimname	The name of the specified attribute dimension.

Parameter	Description
value	The fixed value of the attribute dimension.

Examples

```
SELECT ST_AsText(ST_FilterEquals(pa, 'y', 45.57)) FROM patches WHERE id = 7;
------------------------------------------------------------
 {"pcid":1,"pts":[[-126.42,45.57,58,5],[-126.41,45.57,59,5]]}
```

7.3.9. ST_filterBetween

This function filters all pcpoint objects to obtain the pcpoint objects whose values of the specified attribute dimension are between two fixed values in a pcpatch object and returns a new pcpatch object that is composed of these pcpoint objects.

Syntax

```
pcpatch ST_filterBetween(pcpatch pc, text dimname, float8 minvalue, float8 maxvalue);
```

Parameters

Parameter	Description
pc	The pcpatch object.
dimname	The name of the specified attribute dimension.
minvalue	The minimum fixed value of the attribute dimension.
maxvalue	The maximum fixed value of the attribute dimension.

Description

The pcpoint objects whose values of the specified attribute dimension are equal to the minimum or maximum fixed value are not returned.

Examples

```
SELECT ST_AsText(ST_FilterBetween(pa, 'y', 45.57, 45.60)) FROM patches WHERE id = 7;
------------------------------------------------------------
 {"pcid":1,"pts":[[-126.42,45.58,58,5],[-126.41,45.59,59,5]]}
```

7.3.10. ST_isSorted

This function returns true if all the pcpoint objects in a pcpatch object are sorted based on the specified attribute dimensions.

Syntax

```
boolean ST_isSorted(pcpatch pc, text[] dimnames, boolean strict default true);
```

Parameters

Parameter	Description
pc	The pcpatch object.
dimnames	The array of attribute dimension names.
strict	Indicates whether each pcpoint object has unique values for the specified attribute dimensions.

Examples

```
SELECT ST_isSorted(pa, Array['a','c']) FROM patches;
------------------------------------------------------------
 f
(1 rows)
```

7.3.11. ST_patchMax

This function computes the maximum values of all the attribute dimensions for all the pcpoint objects in a pcpatch object and returns a new pcpoint object whose attribute dimensions are set to these maximum values.

Syntax

```
pcpoint ST_patchMax(pcpatch pc);
```

Parameters

Parameter	Description
pc	The pcpatch object.

Examples

```
SELECT ST_AsText(ST_PatchMax(pa)) FROM patches WHERE id = 7;
----------------------------------
{"pcid":1,"pt":[-126.41,45.59,59,5]}
```

7.3.12. ST_patchMin

This function computes the minimum values of all the attribute dimensions for all the pcpoint objects in a pcpatch object and returns a new pcpoint object whose attribute dimensions are set to these minimum values.

Syntax

```
numeric ST_patchMin(pcpatch pc);
```

Parameters

Parameter	Description
pc	The pcpatch object.

Examples

```
SELECT ST_AsText(ST_PatchMin(pa)) FROM patches WHERE id = 7;
---------------------------------
{"pcid":1,"pt":[-126.5,45.5,50,5]}
```

7.3.13. ST_patchAvg

This function computes the average value of an attribute dimension for all the pcpoint objects in a pcpatch object.

Syntax

```
numeric ST_patchAvg(pcpatch pc, text dimname);
```

Parameters

Parameter	Description
pc	The pcpatch object.
dimname	The name of the specified attribute dimension.

Examples

```
SELECT ST_PatchAvg(pa, 'intensity') FROM patches WHERE id = 7;
--------------------------
5.0000000000000000
```

7.3.14. ST_patchMax

This function computes the maximum value of an attribute dimension for all the pcpoint objects in a pcpatch object.

Syntax

```
numeric ST_patchMax(pcpatch pc, text dimname);
```

Parameters

Parameter	Description
pc	The pcpatch object.
dimname	The name of the specified attribute dimension.

Examples

```
SELECT ST_PatchMax(pa, 'intensity') FROM patches WHERE id = 7;
--------------------------
125.0000000000000000
```

7.3.15. ST_patchMin

This function computes the minimum value of an attribute dimension for all the pcpoint objects in a pcpatch object.

Syntax

```
numeric ST_patchMin(pcpatch pc, text dimname);
```

Parameters

Parameter	Description
pc	The pcpatch object.
dimname	The name of the specified attribute dimension.

Examples

```
SELECT ST_PatchMin(pa, 'intensity') FROM patches WHERE id = 7;
--------------------------
0.25122
```

7.3.16. ST_pointN

This function returns the pcpoint object with the specified sequence number in a pcpatch object.

Syntax

```
pcpoint ST_pointN(pcpatch pc, integer n);
```

Parameters

Parameter	Description
pc	The pcpatch object.
n	The sequence number of the pcpoint object, starting from 1. If n is a negative number, it specifies the pcpoint object with the sequence number of |n| starting from the end of the pcpatch object reversely. For example, a value of -2 indicates the last but one pcpoint object in the pcpatch object.

Examples

```
SELECT ST_asText(ST_pointN(pa, 4)) FROM patches WHERE id = 7;
----------------------------------
{"pcid":1,"pt":[-126.41,45.59,59,5]}
```

7.3.17. ST_sort

This function sorts all the pcpoint objects in a pcpatch object based on the specified attribute dimensions and returns a new pcpatch object that is composed of the sorted pcpoint objects.

Syntax

```
pcpatch ST_sort(pcpatch pc, text[] dimnames);
```

Parameters

Parameter	Description
pc	The pcpatch object.
dimnames	The array of attribute dimension names.

Examples

```
update patches set pa=ST_Sort(pa, Array['y','x']);
--------------------------------------
(1 rows)
```

7.3.18. ST_range

This function obtains n consecutive pcpoint objects that start from a sequence number from a pcpatch object and returns a new pcpatch object that is composed of these pcpoint objects.

Syntax

```
pcpatch ST_range(pcpatch pc, integer start, integer n);
```

Parameters

Parameter	Description
pc	The pcpatch object.
start	The sequence number of the start pcpoint object. The value starts from 1.
n	The number of pcpoint objects after the pcpoint object (included) specified by the start parameter.

Examples

```
update patches set pa=ST_range(pa, 2, 16);
---------------------------------------
(1 rows)
```

7.3.19. ST_setPcid

This function sets a new schema for a pcpatch object and returns a new pcpatch object.

Syntax

```
pcpatch ST_setPcid(pcpatch pc, integer pcid, float8 def default 0.0);
```

Parameters

Parameter	Description
pc	The pcpatch object.
pcid	The ID of the new schema, which comes from the pointcloud_formats table.
def	The specified value. Attribute dimensions that exist in the new schema but not in the old schema are set to this value. The default value is 0.0.

Description

Attribute dimensions that exist in the old schema but not in the new schema are discarded.

Examples

```
update patches set pa=ST_setPcid(pa, 2, 0.0);
---------------------------------------
(1 rows)
```

7.3.20. ST_transform

This function transforms a pcpatch object based on a new schema and returns a new pcpatch object.

Syntax

```
pcpatch ST_transform(pcpatch pc, integer pcid, float8 def default 0.0);
```

Parameters

Parameter	Description
pc	The pcpatch object.
pcid	The ID of the new schema, which comes from the pointcloud_formats table.
def	The specified value. Attribute dimensions that exist in the new schema but not in the old schema are set to this value. The default value is 0.0.

Description

Different from the ST_setPcid function, the ST_transform function returns a new pcpatch object based on the different dimension interpretations, scales, or offsets in the new schema.

Examples

```
update patches set pa=ST_transform(pa, 2, 0.0);
--------------------------------------
(1 rows)
```

7.3.21. ST_unCompress

This function returns an uncompressed version of the pcpatch object.

Syntax

```
pcpatch ST_unCompress(pcpatch pc);
```

Parameters

Parameter	Description
pc	The pcpatch object.

Description

All the pcpatch objects returned by functions are compressed based on the compression method specified in the schema before they are returned.

Examples

```
SELECT ST_Uncompress(pa) FROM patches WHERE ST_NumPoints(pa) = 1;
--------------------------------------
0101000000000000000001000000C8CEFFFFF8110000102700000A00
```

7.3.22. ST_union

This function merges an array of pcpatch entries into a pcpatch object.

Syntax

```
pcpatch ST_union(pcpatch[] pcs);
```

Parameters

Parameter	Description
pcs	The pcpatch array.

Examples

```
-- Compare npoints(sum(patches)) with sum(npoints(patches)).
SELECT ST_NumPoints(ST_Union(pa)) FROM patches;
SELECT Sum(ST_NumPoints(pa)) FROM patches;
100
```

7.4. OGC WKB operations

7.4.1. ST_asBinary

This function returns the well-known binary (WKB) value of a pcpoint object.

Syntax

```
bytea ST_asBinary(pcpoint pt);
```

Parameters

Parameter	Description
pt	The pcpoint object.

Examples

```
SELECT ST_AsBinary('010100000064CEFFFF94110000703000000400'::pcpoint);
---------------------------------------------
\x01010000800000000000c05fc0000000000080464000000000000005f40
```

7.4.2. ST_envelopeAsBinary

This function returns the well-known binary (WKB) value of the bounding box of a pcpatch object.

Syntax

```
bytea ST_envelopeAsBinary(pcpatch pc);
```

Parameters

Parameter	Description
pc	The pcpatch object.

Description

A bounding box is a 2D geometry object of Ganos Geometry.

Examples

```
SELECT ST_EnvelopeAsBinary(pa) FROM patches LIMIT 1;
---------------------------------------------
\x01030000000100000005000000090c2f5285cbf5fc0e17a
14ae4781464090c2f5285cbf5fc0ec51b81e858b46400ad7
a3703dba5fc0ec51b81e858b46400ad7a3703dba5fc0e17a
14ae4781464090c2f5285cbf5fc0e17a14ae47814640
```

7.4.3. ST_boundingDiagonalAsBinary

This function returns the well-known binary (WKB) value of the bounding box diagonal of a pcpatch object.

Syntax

```
bytea ST_boundingDiagonalAsBinary(pcpatch pc);
```

Parameters

Parameter	Description
pc	The pcpatch object.

Description

A bounding box is a 2D geometry object.

Examples

```
SELECT ST_BoundingDiagonalAsBinary( ST_Patch(ARRAY[ ST_MakePoint(1, ARRAY[0.,0.,0.,10.]), S
T_MakePoint(1, ARRAY[1.,1.,1.,10.]), ST_MakePoint(1, ARRAY[10.,10.,10.,10.])]));
-------------------------------------------------------
\x01020000a0e6100000020000000000000000000000000000000000000000000000000000000000000000000244
0000000000000244000000000000002440
```

7.5. Spatial relationship judgment

7.5.1. ST_intersects

This function returns true if the bounding boxes of two pcpatch objects or the bounding boxes of a pcpatch object and a geometry object intersect.

Syntax

```
boolean ST_intersects(pcpatch pp1, pcpatch pp2);
boolean ST_intersects(geometry g,  pcpatch pp1);
```

Parameters

Parameter	Description
pp1	Pcpatch object 1.
pp2	Pcpatch object 2.
g	The geometry object of Ganos Geometry.

Examples

```
-- Patch should intersect itself
SELECT ST_Intersects(
         '01010000000000000001000000C8CEFFFFF8110000102700000A00'::pcpatch,
         '01010000000000000001000000C8CEFFFFF8110000102700000A00'::pcpatch);
------------------------
t
SELECT ST_Intersects('SRID=4326;POINT(-126.451 45.552)'::geometry, pa) FROM patches WHERE i
d = 7;
------------------------
t
```

7.6. Spatial processing

7.6.1. ST_intersection

This function returns a new pcpatch object that indicates the intersection of a pcpatch object and the given geometry object.

Syntax

```
pcpatch ST_intersection(pcpatch pc, geometry geom);
```

Parameters

Parameter	Description
pc	The pcpatch object.
geom	The geometry object of Ganos Geometry.

Examples

```
SELECT ST_AsText(ST_Explode(ST_Intersection(
      pa,
      'SRID=4326;POLYGON((-126.451 45.552, -126.42 47.55, -126.40 45.552, -126.451 45.552))
'::geometry
)))
FROM patches WHERE id = 7;
             st_astext
--------------------------------------
 {"pcid":1,"pt":[-126.44,45.56,56,5]}
 {"pcid":1,"pt":[-126.43,45.57,57,5]}
 {"pcid":1,"pt":[-126.42,45.58,58,5]}
 {"pcid":1,"pt":[-126.41,45.59,59,5]}
```

8.Trajectory SQL reference

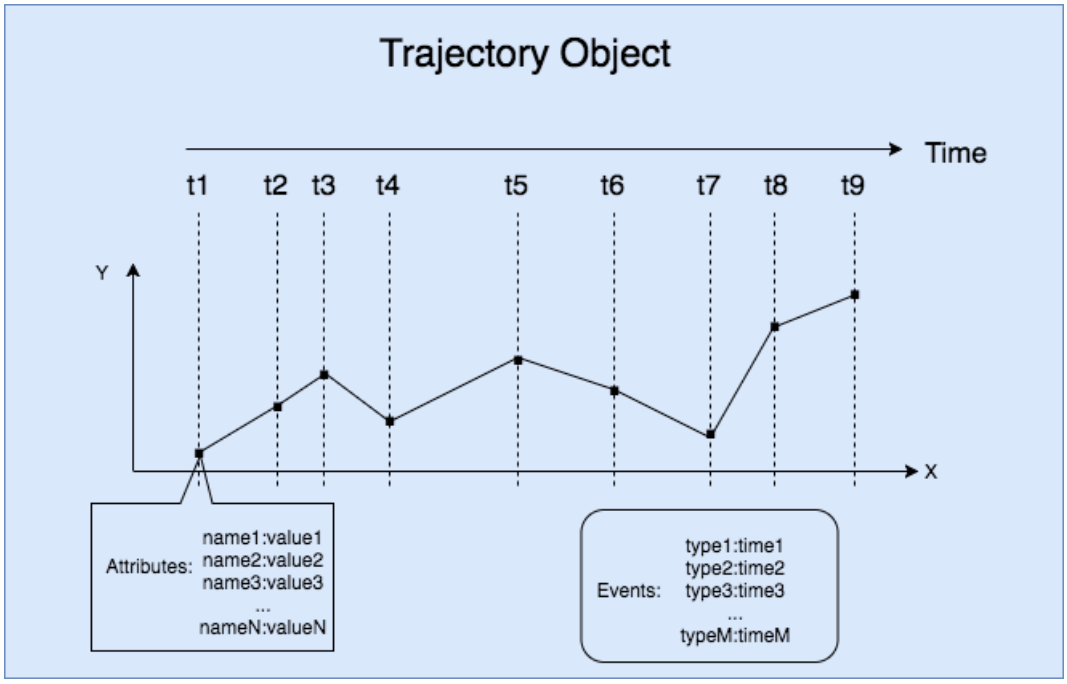
8.1. Terms

This topic introduces the basic concepts of Trajectory SQL.

Trajectory objects

Concept	Description
Trajectory Point	The spatio-temporal object that consists of the spatial location and attributes of a moving object at a specific point in time. The spatial location can be represented by 2D or 3D coordinates. The attributes can be a number of fields that use different data types.
Trajectory Object	The high-dimensional object that consists of trajectory points and events. Specifically, the high-dimensional object includes time, space, attributes, and events.
Trajectory Timeline	The time value sequence of a trajectory. The time values in the sequence must be consecutive.
Trajectory Spatial	The spatial geometry of a trajectory. In most cases, the spatial geometry is described by using the LineString data type.
Trajectory Leaf	The spatial location of a moving object at a specific point in time. The spatial location is a trajectory point.
Trajectory Attributes	The attributes that describe the trajectory of a moving object at different trajectory points. These attributes include velocity and direction.
Trajectory Attribute Field	The attribute field that describes the trajectory of a moving object. For example, these attribute fields include the velocity field. The number of values for an attribute field is the same as the number of trajectory points that is owned by the moving object.
Trajectory Field Value	The value of an attribute field at a specific point in time.
Trajectory Events	The events that occur along a trajectory. For example, these events include the refueling, breakdown, and car lock events of a moving car along the trajectory. An event consists of the event type ID and the event time.

The following figure shows the trajectory of a moving object.

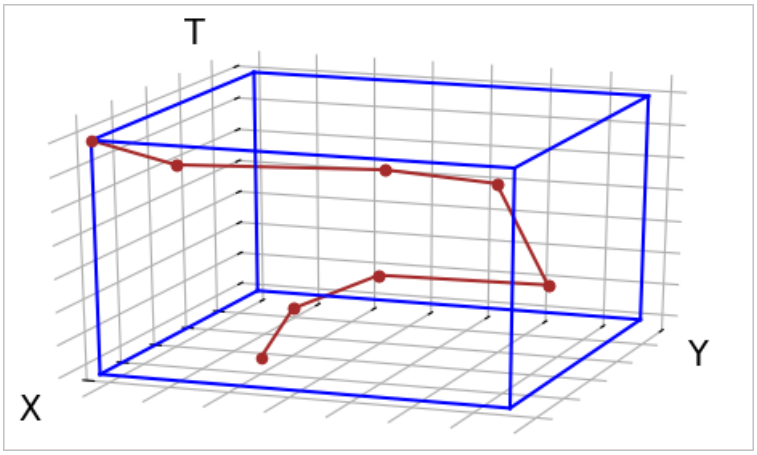


BoxNDF objects

Trajectory-related objects (trajectories and geometries) can be complex. Therefore, cubes that are easier to calculate than these trajectory-related objects may be used to describe queries or simplify computing operations. The BoxNDF data type is used to describe cubes.

A BoxNDF object is a multi-dimensional spatio-temporal cube that is expressed by the minimum and maximum values on the following four coordinate axes: x, y, z, and t. The z axis indicates space, and the t axis indicates time. Different cubes may include different axes. For example, some cubes include only the x and y axes, and some include all of the x, y, z, and t axes.

When you run a query, you can use a cube to specify the query scope. In addition, you can use a bounding box of the trajectory or geometry type to make the query easier. The following figure shows the trajectory and bounding box of an object that moves on the x, y, and t axes.



8.2. Constructors

8.2.1. Overview

Constructors include the function for constructing a trajectory object by using JSON-formatted strings or arrays and the function for appending trajectory points or a sub-trajectory to a trajectory object.

8.2.2. ST_append

This function appends trajectory points or a sub-trajectory to a trajectory object.

Syntax

```
trajectory ST_append(trajectory traj, geometry spatial, timestamp[] timespan, text str_attr
s_json);
trajectory ST_append(trajectory traj, trajectroy tail);
```

Parameters

Parameter	Description
traj	The original trajectory object.
spatial	The spatial geometry object of the appended trajectory object.
timespan	The time array of the appended trajectory object. It can be a timeline.
str_attrs_json	The attributes of the appended trajectory object. For more information, see ST_makeTrajectory.
tail	The appended trajectory object.

Examples

```
With traj AS (  Select ST_makeTrajectory('STPOINT', 'LINESTRING(1 1, 6 6, 9 8)'::geometry,
'[2010-01-01 11:30, 2010-01-01 15:00)'::tsrange,  '{"leafcount":3,"attributes":{"velocity":
{"type": "integer", "length": 2,"nullable" : true,"value": [120, 130, 140]}, "accuracy": {"
type": "float", "length": 4, "nullable" : false,"value": [120, 130, 140]}, "bearing": {"typ
e": "float", "length": 8, "nullable" : false,"value": [120, 130, 140]}, "acceleration": {"t
ype": "string", "length": 20, "nullable" : true,"value": ["120", "130", "140"]}, "active":
{"type": "timestamp", "nullable" : false,"value": ["Fri Jan 01 14:30:00 2010", "Fri Jan 01
15:00:00 2010", "Fri Jan 01 15:30:00 2010"]}}, "events": [{"1" : "Fri Jan 01 14:30:00 2010"
}, {"2" : "Fri Jan 01 15:00:00 2010"}, {"3" : "Fri Jan 01 15:30:00 2010"}]}') a,  ST_makeTr
ajectory('STPOINT', 'LINESTRING(7 7, 3 4, 1 5)'::geometry, '[2010-01-02 15:30, 2010-01-02 1
8:00)'::tsrange,  '{"leafcount":3,"attributes":{"velocity": {"type": "integer", "length": 2
,"nullable" : true,"value": [121, 131, 141]}, "accuracy": {"type": "float", "length": 4, "n
ullable" : false,"value": [121, 131, 141]}, "bearing": {"type": "float", "length": 8, "null
able" : false,"value": [121, 131, 141]}, "acceleration": {"type": "string", "length": 20, "
nullable" : true,"value": ["121", "131", "141"]}, "active": {"type": "timestamp", "nullable
" : false,"value": ["Fri Jan 02 14:30:00 2010", "Fri Jan 02 15:00:00 2010", "Fri Jan 02 15:
30:00 2010"]}}, "events": [{"1" : "Fri Jan 02 14:30:00 2010"}, {"2" : "Fri Jan 02 15:00:00
2010"}, {"3" : "Fri Jan 02 15:30:00 2010"}]}') b)Select ST_Append(a, b) from traj;
                st_append

-----------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------
-----------------------------------------------
 {"trajectory":{"version":1,"type":"STPOINT","leafcount":6,"start_time":"2010-01-01 11:30:0
0","end_time":"2010-01-02 18:00:00","spatial":"LINESTRING(1 1,6 6,9 8,7 7,3 4,1 5)","timeli
ne":["2010-01-01 11:30:00","2010-01-01 13:15:00","2010-01-01 15:00:00","2010-01-02 15:30:00
","2010-01-02 16:45:00","2010-01-02 18:00:00"],"attributes":{"leafcount":6,"velocity":{"typ
e":"integer","length":2,"nullable":true,"value":[120,130,140,121,131,141]},"accuracy":{"typ
e":"float","length":4,"nullable":false,"value":[120.0,130.0,140.0,121.0,131.0,141.0]},"bear
ing":{"type":"float","length":8,"nullable":false,"value":[120.0,130.0,140.0,121.0,131.0,141
.0]},"acceleration":{"type":"string","length":20,"nullable":true,"value":["120","130","140"
,"121","131","141"]},"active":{"type":"timestamp","length":8,"nullable":false,"value":["201
0-01-01 14:30:00","2010-01-01 15:00:00","2010-01-01 15:30:00","2010-01-02 14:30:00","2010-0
1-02 15:00:00","2010-01-02 15:30:00"]}},"events":[{"1":"2010-01-01 14:30:00"},{"2":"2010-01
-01 15:00:00"},{"3":"2010-01-01 15:30:00"},{"1":"2010-01-02 14:30:00"},{"2":"2010-01-02 15:
00:00"},{"3":"2010-01-02 15:30:00"}]}}
(1 row)
```

8.2.3. ST_makeTrajectory

This function constructs a trajectory object.

Syntax

```
trajectory  ST_makeTrajectory (leaftype type, geometry spatial, tsrange timespan, cstring a
ttrs_json);
trajectory  ST_makeTrajectory (leaftype type, geometry spatial, timestamp start, timestamp
end, cstring attrs_json);
trajectory  ST_makeTrajectory (leaftype type, geometry spatial, timestamp[] timeline, cstri
ngattrs_json);
trajectory ST_makeTrajectory (leaftype type, float8[] x, float8[] y, integer srid, timestam
p[] timeline, text[] attr_field_names, int4[] attr_int4, float8[] attr_float8, text[] attr_
cstring, anyarrayattr_any);
```

Parameters

Parameter	Description
type	The trajectory type. Only ST_POINT is supported.
spatial	The spatial geometry object based on a linestring or point object.
timespan	The time range of the trajectory, which contains the start time and end time.
start	The start time of the trajectory.
end	The end time of the trajectory.
timeline	The trajectory timeline. The number of time points must be the same as that of points in the linestring object.
attrs_json	The trajectory attributes and events, in JSON format. The value can be null.
attr_field_names	The array of the names of all attribute fields for the trajectory.
x	The x-axis (array) for the geometry object.
y	The y-axis (array) for the geometry object.
srid	The spatial reference system identifier (SRID) of the trajectory. This parameter is required.

1. The format of the attrs_json parameter is as follows:

```
{
  "leafcount": 3,
  "attributes": {
    "velocity": {
      "type": "integer",
      "length": 2,
      "nullable": true,
      "value": [
        120,
        null,
        140
      ]
    },
    "accuracy": {
```

```
    "accuracy": {
      "type": "float",
      "length": 4,
      "nullable": false,
      "value": [
        120,
        130,
        140
      ]
    },
    "bearing": {
      "type": "float",
      "length": 8,
      "nullable": false,
      "value": [
        120,
        130,
        140
      ]
    },
    "vesname": {
      "type": "string",
      "length": 20,
      "nullable": true,
      "value": [
        "dsff",
        "fgsd",
        null
      ]
    },
    "active": {
      "type": "timestamp",
      "nullable": false,
      "value": [
        "Fri Jan 01 14:30:00 2010",
        "Fri Jan 01 15:00:00 2010",
        "Fri Jan 01 15:30:00 2010"
      ]
    }
  },
  "events": [
    {
      "1": "Fri Jan 01 14:30:00 2010"
    },
    {
      "2": "Fri Jan 01 15:00:00 2010"
    },
    {
      "3": "Fri Jan 01 15:30:00 2010"
    }
  ]
}
```

leafcount specifies the number of trajectory points in the trajectory object. Its value must be consistent with the number of spatial points in the spatial geometry object. Its value is also the same as the number of values for each attribute field. All attribute fields must have the same number of values.

attributes specifies trajectory attributes, including the definitions and a sequence of values for all attribute fields. It must co-exist with leafcount. Attribute definitions and requirements are as follows:

- An attribute name can contain a maximum of **60** characters.
- type: the field type. Valid values: integer, float, string, timestamp, and bool.
- length: the field length. The value varies with the field type. integer: 1, 2, 4, or 8. float: 4 or 8. string: customizable. (If not specified, the default length is 64 and the maximum length is 253. The length value is the actual number of characters and excludes the end identifier.) timestamp: If not specified, the default length is 8. bool: If not specified, the default length is 1.
- nullable: indicates whether the field can be null. Valid values: true and false. Default value: true.
- value: the sequence of field values, listed in JSON array format. If a value is null, null is displayed.

events specifies trajectory events. Multiple events are listed in JSON array format. Each array element is expressed in key:value format, where key indicates the event type and value indicates the event time.

2. If only the timespan parameter or the start and end parameters are specified as time, such as in syntax 1 and syntax 2, this constructor can interpolate time points based on the number of spatial points to generate the trajectory timeline.
3. If all the four types of syntax do not meet your requirements, you can customize the parameters following the first six fixed parameters to create a custom ST_makeTrajectory constructor as follows:

```
CREATE OR REPLACE FUNCTION  _ST_MakeTrajectory(type leaftype, x float8[], y float8[] ,
srid integer, timespan timestamp[],
    attrs_name cstring[], attr1 float8[], attr2 float4[], attr3 timestamp[])
    RETURNS trajectory
    AS '$libdir/libpg-trajectoryxx','sqltr_traj_make_all_array'
    LANGUAGE 'c' IMMUTABLE Parallel SAFE;
```

Examples

```
-- (1) ST_MakeTrajectory with a time range
select ST_MakeTrajectory('STPOINT'::leaftype, st_geomfromtext('LINESTRING (114 35, 115 36,
116 37)', 4326), '[2010-01-01 14:30, 2010-01-01 15:30)'::tsrange, '{"leafcount":3,"attribut
es":{"velocity": {"type": "integer", "length": 2,"nullable" : true,"value": [120, 130, 140]
}, "accuracy": {"type": "float", "length": 4, "nullable" : false,"value": [120, 130, 140]},
"bearing": {"type": "float", "length": 8, "nullable" : false,"value": [120, 130, 140]}, "ve
sname": {"type": "string", "length": 20, "nullable" : true,"value": ["adsf", "sdf", "sdfff"
]}, "active": {"type": "timestamp", "nullable" : false,"value": ["Fri Jan 01 14:30:00 2010"
, "Fri Jan 01 15:00:00 2010", "Fri Jan 01 15:30:00 2010"]}}, "events": [{"1" : "Fri Jan 01
14:30:00 2010"}, {"2" : "Fri Jan 01 15:00:00 2010"}, {"3" : "Fri Jan 01 15:30:00 2010"}]}')
;                        st_maketrajectory
-----------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------
```

```
---------------------------------
 {"trajectory":{"version":1,"type":"STPOINT","leafcount":3,"start_time":"2010-01-01 14:30:0
0","end_time":"2010-01-01 15:30:00","spatial":"SRID=4326;LINESTRING(114 35,115 36,116 37)",
"timeline":["2010-01-01 14:30:00","2010-01-01 15:00:00","2010-01-01 15:30:00"],"attributes"
:{"leafcount":3,"velocity":{"type":"integer","length":2,"nullable":true,"value":[120,130,14
0]},"accuracy":{"type":"float","length":4,"nullable":false,"value":[120.0,130.0,140.0]},"be
aring":{"type":"float","length":8,"nullable":false,"value":[120.0,130.0,140.0]},"vesname":{
"type":"string","length":20,"nullable":true,"value":["adsf","sdf","sdfff"]},"active":{"type
":"timestamp","length":8,"nullable":false,"value":["2010-01-01 14:30:00","2010-01-01 15:00:
00","2010-01-01 15:30:00"]}},"events":[{"1":"2010-01-01 14:30:00"},{"2":"2010-01-01 15:00:0
0"},{"3":"2010-01-01 15:30:00"}]}}
(1 row)
-- (2) ST_MakeTrajectory with a start timestamp and an end timestamp
select ST_MakeTrajectory('STPOINT'::leaftype, st_geomfromtext('LINESTRING (114 35, 115 36,
116 37)', 4326), '2010-01-01 14:30'::timestamp, '2010-01-01 15:30'::timestamp, '{"leafcount
":3,"attributes":{"velocity": {"type": "integer", "length": 2,"nullable" : true,"value": [1
20, 130, 140]}, "accuracy": {"type": "float", "length": 4, "nullable" : false,"value": [120
, 130, 140]}, "bearing": {"type": "float", "length": 8, "nullable" : false,"value": [120, 1
30, 140]}, "vesname": {"type": "string", "length": 20, "nullable" : true,"value": ["adsf",
"sdf", "sdfff"]}, "active": {"type": "timestamp", "nullable" : false,"value": ["Fri Jan 01
14:30:00 2010", "Fri Jan 01 15:00:00 2010", "Fri Jan 01 15:30:00 2010"]}}, "events": [{"1"
: "Fri Jan 01 14:30:00 2010"}, {"2" : "Fri Jan 01 15:00:00 2010"}, {"3" : "Fri Jan 01 15:30
:00 2010"}]}');                        st_maketrajectory
-----------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------
---------------------------------
 {"trajectory":{"version":1,"type":"STPOINT","leafcount":3,"start_time":"2010-01-01 14:30:0
0","end_time":"2010-01-01 15:30:00","spatial":"SRID=4326;LINESTRING(114 35,115 36,116 37)",
"timeline":["2010-01-01 14:30:00","2010-01-01 15:00:00","2010-01-01 15:30:00"],"attributes"
:{"leafcount":3,"velocity":{"type":"integer","length":2,"nullable":true,"value":[120,130,14
0]},"accuracy":{"type":"float","length":4,"nullable":false,"value":[120.0,130.0,140.0]},"be
aring":{"type":"float","length":8,"nullable":false,"value":[120.0,130.0,140.0]},"vesname":{
"type":"string","length":20,"nullable":true,"value":["adsf","sdf","sdfff"]},"active":{"type
":"timestamp","length":8,"nullable":false,"value":["2010-01-01 14:30:00","2010-01-01 15:00:
00","2010-01-01 15:30:00"]}},"events":[{"1":"2010-01-01 14:30:00"},{"2":"2010-01-01 15:00:0
0"},{"3":"2010-01-01 15:30:00"}]}}
(1 row)
-- (3) ST_MakeTrajectory with a timestamp array
select ST_MakeTrajectory('STPOINT'::leaftype, st_geomfromtext('LINESTRING (114 35, 115 36,
116 37)', 4326), ARRAY['2010-01-01 14:30'::timestamp, '2010-01-01 15:00'::timestamp, '2010-
01-01 15:30'::timestamp], '{"leafcount":3,"attributes":{"velocity": {"type": "integer", "le
ngth": 2,"nullable" : true,"value": [120, 130, 140]}, "accuracy": {"type": "float", "length
": 4, "nullable" : false,"value": [120, 130, 140]}, "bearing": {"type": "float", "length":
8, "nullable" : false,"value": [120, 130, 140]}, "vesname": {"type": "string", "length": 20
, "nullable" : true,"value": ["adsf", "sdf", "sdfff"]}, "active": {"type": "timestamp", "nu
llable" : false,"value": ["Fri Jan 01 14:30:00 2010", "Fri Jan 01 15:00:00 2010", "Fri Jan
01 15:30:00 2010"]}}, "events": [{"1" : "Fri Jan 01 14:30:00 2010"}, {"2" : "Fri Jan 01 15:
00:00 2010"}, {"3" : "Fri Jan 01 15:30:00 2010"}]}');                        st_maketraject
ory
-----------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------
```

```
-----------------------------------------------------------------------------------------------
---------------------------------
 {"trajectory":{"version":1,"type":"STPOINT","leafcount":3,"start_time":"2010-01-01 14:30:0
0","end_time":"2010-01-01 15:30:00","spatial":"SRID=4326;LINESTRING(114 35,115 36,116 37)",
"timeline":["2010-01-01 14:30:00","2010-01-01 15:00:00","2010-01-01 15:30:00"],"attributes"
:{"leafcount":3,"velocity":{"type":"integer","length":2,"nullable":true,"value":[120,130,14
0]},"accuracy":{"type":"float","length":4,"nullable":false,"value":[120.0,130.0,140.0]},"be
aring":{"type":"float","length":8,"nullable":false,"value":[120.0,130.0,140.0]},"vesname":{
"type":"string","length":20,"nullable":true,"value":["adsf","sdf","sdfff"]},"active":{"type
":"timestamp","length":8,"nullable":false,"value":["2010-01-01 14:30:00","2010-01-01 15:00:
00","2010-01-01 15:30:00"]}},"events":[{"1":"2010-01-01 14:30:00"},{"2":"2010-01-01 15:00:0
0"},{"3":"2010-01-01 15:30:00"}]}}
(1 row)
-- (4) ST_MakeTrajectory with null attrs_json
select ST_MakeTrajectory('STPOINT'::leaftype, st_geomfromtext('LINESTRING (114 35, 115 36,
116 37)', 4326), '[2010-01-01 14:30, 2010-01-01 15:30)'::tsrange, null);
                        st_maketrajectory

-----------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------
-----------------------------------------------------------------
{"trajectory":{"leafsize":3,"starttime":"Fri Jan 01 14:30:00 2010","endtime":"Fri Jan 01 15
:30:00 2010","spatial":"LINESTRING(114 35,115 36,116 37)","timeline":["Fri Jan 01 14:30:00
2010","Fri Jan 01 15:00:00 2010","Fri Jan 01 15:30:00 2010"]}}
(1 row)
-- (5) ST_MakeTrajectory that constructs a trajectory object from points
select st_makeTrajectory('STPOINT'::leaftype, ARRAY[1::float8], ARRAY[2::float8], 4326, ARR
AY['2010-01-01 11:30'::timestamp], ARRAY['velocity'], ARRAY[1::int4], NULL, NULL, NULL::any
array);
                                                        st_maketrajectory

-----------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------
{"trajectory":{"version":1,"type":"STPOINT","leafcount":1,"start_time":"2010-01-01 11:30:00
","end_time":"2010-01-01 11:30:00","spatial":"SRID=4326;POINT(1 2)","timeline":["2010-01-01
11:30:00"],"attributes":{"leafcount":1,"velocity":{"type":"integer","length":4,"nullable":t
rue,"value":[1]}}}}
(1 row)
```

8.3. Editing and processing functions

8.3.1. ST_attrDeduplicate

This function removes trajectory points whose values for the specified attribute field are duplicate from a trajectory object and returns the trajectory object after deduplication. The start and end trajectory points of the trajectory object must be kept.

Syntax

```
trajectory ST_attrDeduplicate(trajectory traj, cstring attr_field_name);
```

Parameters

Parameter	Description
traj	The trajectory object.
attr_field_name	The name of the specified attribute field.

Examples

```
select st_attrDeduplicate(ST_makeTrajectory('STPOINT'::leaftype, 'LINESTRING(-179.48077 51.
72814,-179.46731 51.74634,-179.46502 51.74934,-179.46183 51.75378,-179.45943 51.75736,-179.
45560 51.76273,-179.44845 51.77186,-179.43419 51.78977,-179.41259 51.81643,-179.41001 51.81
941,-179.40751 51.82223,-179.40497 51.82505,-179.40242 51.82796,-179.39981 51.83095,-179.39
734 51.83398,-179.39499 51.83709)'::geometry, ARRAY['2017-01-15 09:06:39'::timestamp,'2017-
01-15 09:13:39','2017-01-15 09:14:48','2017-01-15 09:16:28','2017-01-15 09:17:48','2017-01-
15 09:19:48','2017-01-15 09:23:19','2017-01-15 09:30:28','2017-01-15 09:34:40','2017-01-15
09:36:59','2017-01-15 09:38:09','2017-01-15 09:39:18','2017-01-15 09:40:40','2017-01-15 09:
47:38','2017-01-15 09:48:49','2017-01-15 21:18:30'], '{"leafcount": 16, "attributes": {"hea
ding": {"type": "float", "length": 4, "nullable": false,"value": [23.0,23.0,23.0,23.0,21.0,
21.0,72.0,72.0,72.0,72.0,73.0,74.0,73.0,73.0,73.0,73.0]}}}'),'heading') as traj;
                                                        traj

-----------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------
 {"trajectory":{"version":1,"type":"STPOINT","leafcount":6,"start_time":"2017-01-15 09:17:4
8","end_time":"2017-01-15 21:18:30","spatial":"LINESTRING(-179.45943 51.75736,-179.44845 51
.77
186,-179.40751 51.82223,-179.40497 51.82505,-179.39499 51.83709)","timeline":["2017-01-15 0
9:17:48","2017-01-15 09:23:19","2017-01-15 09:38:09","2017-01-15 09:39:18","2017-01-15 21:1
8:3
0"],"attributes":{"leafcount":6,"heading":{"type":"float","length":4,"nullable":false,"valu
e":[23.0,21.0,72.0,73.0,74.0,73.0]}}}}
(1 row)
```

8.3.2. ST_Compress

This function compresses a trajectory object based on specified thresholds.

Syntax

```
trajectory ST_Compress (trajectory traj, float8 dist);
trajectory ST_Compress (trajectory traj, float8 dist, float8 angle, float8 acceleration);
trajectory ST_Compress (trajectory traj, float8 dist, float8 angle, float8 acceleration, cs
tring velocity_field);
```

Parameters

Parameter	Description
traj	The original trajectory object.
dist	The Euclidean distance offset threshold. After this value is specified, trajectory points whose Euclidean distance offset is greater than this value are kept, so as to maintain the spatial trend of the original trajectory object.
angle	The angle offset threshold. After this value is specified, trajectory points whose angle changes are greater than this value are kept.
acceleration	The acceleration threshold. After this value is specified, trajectory points whose velocity changes are greater than this value are kept.
velocity_field	The name of the velocity attribute field in the trajectory object. The values of this attribute field are used to compute the acceleration.

Description

This function discards trajectory points based on specified thresholds to compress the trajectory object in lossy mode, and returns the compressed trajectory object.

- Syntax 1: compresses the trajectory object based on the specified offset threshold for the spatial distance. This function maintains the spatial trend of the original trajectory object, but may delete important trajectory points with large angle or velocity changes.
- Syntax 2: compresses the trajectory object based on the specified offset threshold for the spatial distance, angle offset threshold, and acceleration threshold. This function not only maintains the spatial trend of the original trajectory object, but also keeps important trajectory points with large angle or velocity changes. You can specify any one or two of the thresholds for compression.
- Syntax 3: achieves the same effect as syntax 2. This syntax applies when the trajectory object contains the velocity attribute field. After the name of the velocity attribute field is specified, this function directly computes the acceleration of trajectory points based on the values of the velocity attribute field.

Examples

```
-- Create data.
Create table If not exists traj_test(id integer, mmsi integer,traj trajectory);
INSERT INTO traj_test(mmsi, traj) VALUES(477027500,ST_makeTrajectory('STPOINT'::leaftype, '
LINESTRING(-179.48077 51.72814,-179.47416 51.73714,-179.47187 51.74027,-179.46964 51.74325,
-179.46731 51.74634,-179.46502 51.74934,-179.46183 51.75378,-179.45943 51.75736,-179.45560
51.76273,-179.44845 51.77186,-179.43419 51.78977,-179.42595 51.80094,-179.42343 51.80411,-1
79.42078 51.80719,-179.41821 51.81025,-179.41562 51.81308,-179.41259 51.81643,-179.41001 51
.81941,-179.40751 51.82223,-179.40497 51.82505,-179.40242 51.82796,-179.39981 51.83095,-179
.39734 51.83398,-179.39499 51.83709,-179.39264 51.84023,-179.39037 51.84333,-179.38699 51.8
4791,-179.38467 51.85114,-179.38216 51.85439,-179.37997 51.85762,-179.37772 51.86144,-179.3
7474 51.86568,-179.37219 51.86869,-179.36983 51.87156,-179.36755 51.87467,-179.36001 51.884
23,-179.35754 51.88712,-179.34216 51.90644,-179.33935 51.90995,-179.33704 51.91298,-179.188
26 52.10105,-179.18096 52.11031,-179.17504 52.11786,-179.16482 52.12996,-179.16233 52.13289
,-179.15967 52.13590,-179.14599 52.15132,-177.76666 52.85042,-177.48459 52.89898,-177.47841
52.90001,-177.47319 52.90084,-177.46251 52.90268,-177.38188 52.91595,-177.37102 52.91765,-1
77.36378 52.91877,-177.34492 52.92173,-177.33217 52.92364,-177.32581 52.92468,-177.31238 52
.92697,-177.03751 52.97394,-176.93063 52.99160,-176.92406 52.99265,-176.91471 52.99423,-176
.90643 52.99554,-176.89912 52.99674,-176.89246 52.99791,-176.88342 52.99942,-176.87697 53.0
```

```
.90643 52.99554,-176.89912 52.99674,-176.89246 52.99791,-176.88342 52.99942,-176.87697 53.0
0060,-176.86594 53.00256,-176.85946 53.00370,-176.85294 53.00481,-176.84640 53.00592,-176.8
3985 53.00705,-176.83238 53.00830,-176.82589 53.00950,-176.81848 53.01084,-176.80553 53.013
10,-176.79879 53.01419,-176.79115 53.01548,-176.78466 53.01668,-176.77901 53.01765,-176.772
56 53.01879,-176.76301 53.02039,-176.75649 53.02141,-176.74700 53.02296,-176.73757 53.02450
,-176.71683 53.02795,-176.70741 53.02950,-176.68481 53.03327)'::geometry, ARRAY['2017-01-15
09:06:39'::timestamp,'2017-01-15 09:10:08'::timestamp,'2017-01-15 09:11:20'::timestamp,'201
7-01-15 09:12:29'::timestamp,'2017-01-15 09:13:39'::timestamp,'2017-01-15 09:14:48'::timest
amp,'2017-01-15 09:16:28'::timestamp,'2017-01-15 09:17:48'::timestamp,'2017-01-15 09:19:48'
::timestamp,'2017-01-15 09:23:19'::timestamp,'2017-01-15 09:30:28'::timestamp,'2017-01-15 0
9:34:40'::timestamp,'2017-01-15 09:35:49'::timestamp,'2017-01-15 09:36:59'::timestamp,'2017
-01-15 09:38:09'::timestamp,'2017-01-15 09:39:18'::timestamp,'2017-01-15 09:40:40'::timesta
mp,'2017-01-15 09:41:49'::timestamp,'2017-01-15 09:42:58'::timestamp,'2017-01-15 09:44:08':
:timestamp,'2017-01-15 09:45:18'::timestamp,'2017-01-15 09:46:29'::timestamp,'2017-01-15 09
:47:38'::timestamp,'2017-01-15 09:48:49'::timestamp,'2017-01-15 09:49:58'::timestamp,'2017-
01-15 09:51:08'::timestamp,'2017-01-15 09:52:49'::timestamp,'2017-01-15 09:53:58'::timestam
p,'2017-01-15 09:55:09'::timestamp,'2017-01-15 09:56:18'::timestamp,'2017-01-15 09:57:38':
timestamp,'2017-01-15 09:59:09'::timestamp,'2017-01-15 10:00:20'::timestamp,'2017-01-15 10:
01:29'::timestamp,'2017-01-15 10:02:39'::timestamp,'2017-01-15 10:06:29'::timestamp,'2017-0
1-15 10:07:40'::timestamp,'2017-01-15 10:15:00'::timestamp,'2017-01-15 10:16:20'::timestamp
,'2017-01-15 10:17:29'::timestamp,'2017-01-15 11:30:09'::timestamp,'2017-01-15 11:33:58'::t
imestamp,'2017-01-15 11:36:58'::timestamp,'2017-01-15 11:42:00'::timestamp,'2017-01-15 11:4
3:10'::timestamp,'2017-01-15 11:44:20'::timestamp,'2017-01-15 11:50:28'::timestamp,'2017-01
-15 18:01:00'::timestamp,'2017-01-15 18:54:13'::timestamp,'2017-01-15 18:55:21'::timestamp,
'2017-01-15 18:56:22'::timestamp,'2017-01-15 18:58:21'::timestamp,'2017-01-15 19:13:21'::ti
mestamp,'2017-01-15 19:15:21'::timestamp,'2017-01-15 19:16:41'::timestamp,'2017-01-15 19:20
:11'::timestamp,'2017-01-15 19:22:31'::timestamp,'2017-01-15 19:23:41'::timestamp,'2017-01-
15 19:26:10'::timestamp,'2017-01-15 20:15:49'::timestamp,'2017-01-15 20:34:39'::timestamp,'
2017-01-15 20:35:49'::timestamp,'2017-01-15 20:37:30'::timestamp,'2017-01-15 20:39:00'::tim
estamp,'2017-01-15 20:40:19'::timestamp,'2017-01-15 20:41:30'::timestamp,'2017-01-15 20:43:
08'::timestamp,'2017-01-15 20:44:19'::timestamp,'2017-01-15 20:46:19'::timestamp,'2017-01-1
5 20:47:29'::timestamp,'2017-01-15 20:48:40'::timestamp,'2017-01-15 20:49:49'::timestamp,'2
017-01-15 20:50:59'::timestamp,'2017-01-15 20:52:21'::timestamp,'2017-01-15 20:53:29'::time
stamp,'2017-01-15 20:54:50'::timestamp,'2017-01-15 20:57:09'::timestamp,'2017-01-15 20:58:2
0'::timestamp,'2017-01-15 20:59:40'::timestamp,'2017-01-15 21:00:49'::timestamp,'2017-01-15
21:01:50'::timestamp,'2017-01-15 21:02:58'::timestamp,'2017-01-15 21:04:40'::timestamp,'201
7-01-15 21:05:50'::timestamp,'2017-01-15 21:07:29'::timestamp,'2017-01-15 21:09:11'::timest
amp,'2017-01-15 21:12:49'::timestamp,'2017-01-15 21:14:30'::timestamp,'2017-01-15 21:18:30'
::timestamp], '{"leafcount": 89,"attributes" : {"sog" : {"type":"float","length":8,"nullabl
e":false,"value":[10.5,10.4,10.5,10.7,10.8,10.3,10.7,10.4,10.5,10.1,10.2,11.0,11.2,10.8,10.
3,10.3,10.1,10.7,10.6,10.0,10.3,10.5,10.6,10.3,10.8,10.9,10.8,10.8,10.8,11.0,11.2,11.2,10.3
,10.2,10.8,10.0,10.4,10.7,10.2,10.6,9.1,10.2,10.1,9.7,10.4,10.6,9.9,12.3,12.1,12.0,12.0,12.
2,12.3,12.2,12.3,12.2,12.3,12.2,12.2,12.8,12.8,12.9,12.5,12.6,12.5,12.6,12.6,12.3,12.6,12.6
,12.5,12.7,12.8,12.5,12.7,12.5,12.8,13.0,12.9,12.6,12.9,12.8,12.7,12.8,13.0,12.7,12.8,12.6,
12.7]}, "cog" : {"type":"float","length":8,"nullable":false,"value":[23.3,25.7,25.9,23.6,25
.3,24.1,23.0,21.6,20.7,24.8,22.4,28.5,23.1,30.3,26.2,28.1,25.1,28.7,31.4,28.2,30.4,29.4,29.
2,23.0,25.1,25.1,23.5,22.7,27.1,23.3,19.2,27.1,31.0,28.8,22.0,30.1,24.6,26.2,26.7,24.7,26.8
,29.5,19.9,30.1,28.8,28.7,30.0,74.2,69.1,75.2,81.3,81.3,80.1,72.6,82.4,74.2,74.9,67.7,73.4,
74.2,72.2,80.5,78.6,77.3,70.9,80.1,85.4,71.9,67.0,77.5,77.5,72.2,70.5,72.6,70.8,77.8,71.2,7
1.2,73.8,75.4,67.1,77.5,74.3,76.9,80.1,72.8,76.0,75.4,72.9]},"heading" : {"type":"float","l
ength":8,"nullable":false,"value":[22.0,23.0,23.0,23.0,23.0,21.0,21.0,25.0,24.0,26.0,25.0,2
7.0,28.0,29.0,31.0,30.0,28.0,29.0,29.0,28.0,28.0,27.0,24.0,24.0,25.0,25.0,25.0,26.0,25.0,24
.0,25.0,29.0,31.0,28.0,29.0,31.0,28.0,29.0,29.0,29.0,27.0,27.0,26.0,26.0,26.0,27.0,27.0,69.
0,71.0,72.0,72.0,71.0,73.0,72.0,72.0,72.0,72.0,71.0,71.0,72.0,72.0,72.0,73.0,72.0,71.0,72.0
,72.0,72.0,71.0,72.0,72.0,73.0,71.0,72.0,71.0,72.0,73.0,72.0,72.0,72.0,72.0,73.0,74.0,73.0,
```

```
,73.0,73.0,73.0,73.0,73.0,73.0,73.0,73.0,73.0,73.0,73.0,73.0,73.0,73.0,73.0,73.0,73.0,73.0,
73.0,73.0,73.0,73.0,73.0]}}}'));
-- Compress the trajectory object.
select st_compress(traj,0.001) as traj from traj_test;
           traj
-----------------------------------------------------------------
 {"trajectory":{"version":1,"type":"STPOINT","leafcount":8,"start_time":"2017-01-15 09:06:3
9","end_time":"2017-01-15 21:18:30","spatial":"LINESTRING(-179.48077 51.72814,-179.42595 51
.80094,-179.39734 51.83398,-179.37474 51.86568,-179.17504 52.11786,-179.14599 52.15132,-177
.76666 52.85042,-176.68481 53.03327)","timeline":["2017-01-15 09:06:39","2017-01-15 09:34:4
0","2017-01-15 09:47:38","2017-01-15 09:59:09","2017-01-15 11:36:58","2017-01-15 11:50:28",
"2017-01-15 18:01:00","2017-01-15 21:18:30"],"attributes":{"leafcount":8,"sog":{"type":"flo
at","length":8,"nullable":false,"value":[10.5,11.0,10.6,11.2,10.1,9.9,12.3,12.7]},"cog":{"t
ype":"float","length":8,"nullable":false,"value":[23.3,28.5,29.2,27.1,19.9,30.0,74.2,72.9]}
,"heading":{"type":"float","length":8,"nullable":false,"value":[22.0,27.0,24.0,29.0,26.0,27
.0,69.0,73.0]}}}}
(1 row)
select st_compress(traj,0.001,null,null) as traj from traj_test where traj_id=5;
           traj
-----------------------------------------------------------------
{"trajectory":{"type":"STPOINT","leafsize":8,"starttime":"2017-01-15 09:06:39","endtime":"2
017-01-15 21:18:30","spatial":"LINESTRING(-179.48077 51.72814,-179.42595 51.80094,-179.3973
4 51.83398,-179.37474 51.86568,-179.17504 52.11786,-179.14599 52.15132,-177.76666 52.85042,
-176.68481 53.03327)","timeline":["2017-01-15 09:06:39","2017-01-15 09:34:40","2017-01-15 0
9:47:38","2017-01-15 09:59:09","2017-01-15 11:36:58","2017-01-15 11:50:28","2017-01-15 18:0
1:00","2017-01-15 21:18:30"],"themeline":{"leafs":8,"sog":[10.5,11.0,10.6,11.2,10.1,9.9,12.
3,12.7],"cog":[23.3,28.5,29.2,27.1,19.9,30.0,74.2,72.9],"heading":[22.0,27.0,24.0,29.0,26.0
,27.0,69.0,73.0]}}}
(1 row)
select st_compress(traj,0.001,5,0.3) as traj from traj_test;
           traj
-----------------------------------------------------------------
{"trajectory":{"version":1,"type":"STPOINT","leafcount":13,"start_time":"2017-01-15 09:06:3
9","end_time":"2017-01-15 21:18:30","spatial":"LINESTRING(-179.48077 51.72814,-179.42595 51
.80094,-179.39734 51.83398,-179.37474 51.86568,-179.35754 51.88712,-179.18826 52.10105,-179
.17504 52.11786,-179.14599 52.15132,-177.76666 52.85042,-177.47841 52.90001,-177.47319 52.9
0084,-176.83238 53.0083,-176.68481 53.03327)","timeline":["2017-01-15 09:06:39","2017-01-15
09:34:40","2017-01-15 09:47:38","2017-01-15 09:59:09","2017-01-15 10:07:40","2017-01-15 11:
30:09","2017-01-15 11:36:58","2017-01-15 11:50:28","2017-01-15 18:01:00","2017-01-15 18:55:
21","2017-01-15 18:56:22","2017-01-15 20:52:21","2017-01-15 21:18:30"],"attributes":{"leafc
ount":13,"sog":{"type":"float","length":8,"nullable":false,"value":[10.5,11.0,10.6,11.2,10.
4,9.1,10.1,9.9,12.3,12.0,12.0,12.5,12.7]},"cog":{"type":"float","length":8,"nullable":false
,"value":[23.3,28.5,29.2,27.1,24.6,26.8,19.9,30.0,74.2,75.2,81.3,72.6,72.9]},"heading":{"ty
pe":"float","length":8,"nullable":false,"value":[22.0,27.0,24.0,29.0,28.0,27.0,26.0,27.0,69
.0,72.0,72.0,72.0,73.0]}}}}
(1 row)
select st_compress(traj,0.001,5,1.1,'sog') as traj from traj_test;
            traj
-----------------------------------------------------------------
{"trajectory":{"version":1,"type":"STPOINT","leafcount":10,"start_time":"2017-01-15 09:06:3
9","end_time":"2017-01-15 21:18:30","spatial":"LINESTRING(-179.48077 51.72814,-179.42595 51
.80094,-179.39734 51.83398,-179.37474 51.86568,-179.33704 51.91298,-179.18826 52.10105,-179
.17504 52.11786,-179.14599 52.15132,-177.76666 52.85042,-176.68481 53.03327)","timeline":["
2017-01-15 09:06:39","2017-01-15 09:34:40","2017-01-15 09:47:38","2017-01-15 09:59:09","201
7-01-15 10:17:29","2017-01-15 11:30:09","2017-01-15 11:36:58","2017-01-15 11:50:28","2017-0
```

```
1-15 18:01:00","2017-01-15 21:18:30"],"attributes":{"leafcount":10,"sog":{"type":"float","length":8,"nullable":false,"value":[10.5,11.0,10.6,11.2,10.6,9.1,10.1,9.9,12.3,12.7]},"cog":{"type":"float","length":8,"nullable":false,"value":[23.3,28.5,29.2,27.1,24.7,26.8,19.9,30.0,74.2,72.9]},"heading":{"type":"float","length":8,"nullable":false,"value":[22.0,27.0,24.0,29.0,29.0,27.0,26.0,27.0,69.0,73.0]}}}}
(1 row)
```

8.3.3. ST_CompressSED

This function compresses a trajectory object based on the offset threshold of the synchronous Euclidean distance (SED).

Syntax

```
trajectory ST_CompressSed (trajectory traj, float8 dist);
```

Parameters

Parameter	Description
traj	The original trajectory object.
dist	The SED offset threshold. After this value is specified, trajectory points whose SED offset is greater than this value are kept, so as to maintain the spatial trend of the original trajectory object.

Description

This function calculates the SED of trajectory points, discards the points whose SED offset is smaller than the SED offset threshold to compress the trajectory object in lossy mode, and then returns the compressed trajectory object.

Examples

```
select st_compressSED(traj, 0.001) as traj from traj_test;
           traj
-------------------------------------------------------------
{"trajectory":{"version":1,"type":"STPOINT","leafcount":12,"start_time":"2017-01-15 09:06:39","end_time":"2017-01-15 21:18:30","spatial":"LINESTRING(-179.48077 51.72814,-179.42595 51.80094,-179.39734 51.83398,-179.37474 51.86568,-179.18826 52.10105,-179.16482 52.12996,-179.14599 52.15132,-177.76666 52.85042,-177.47319 52.90084,-177.31238 52.92697,-177.03751 52.97394,-176.68481 53.03327)","timeline":["2017-01-15 09:06:39","2017-01-15 09:34:40","2017-01-15 09:47:38","2017-01-15 09:59:09","2017-01-15 11:30:09","2017-01-15 11:42:00","2017-01-15 11:50:28","2017-01-15 18:01:00","2017-01-15 18:56:22","2017-01-15 19:26:10","2017-01-15 20:15:49","2017-01-15 21:18:30"],"attributes":{"leafcount":12,"sog":{"type":"float","length":8,"nullable":false,"value":[10.5,11.0,10.6,11.2,9.1,9.7,9.9,12.3,12.0,12.2,12.8,12.7]},"cog":{"type":"float","length":8,"nullable":false,"value":[23.3,28.5,29.2,27.1,26.8,30.1,30.0,74.2,81.3,73.4,74.2,72.9]},"heading":{"type":"float","length":8,"nullable":false,"value":[22.0,27.0,24.0,29.0,27.0,26.0,27.0,69.0,72.0,71.0,72.0,73.0]}}}}
(1 row)
```

8.3.4. ST_deviation

This function returns the deviation of the processed trajectory object from the original trajectory object.

Syntax

```
float8 ST_deviation(trajectory traj, trajectory after_oper_traj);
```

Parameters

Parameter	Description
traj	The original trajectory object.
after_oper_traj	The processed trajectory object, such as the compressed trajectory object.

Examples

```
select st_deviation(traj, st_compress(traj,0.001)) from traj_test;
    st_deviation
---------------------
0.00919177345596219
(1 row)
```

8.3.5. ST_sort

This function sorts the timeline of a trajectory object in ascending order and returns the trajectory object after sorting.

Syntax

```
trajectory ST_sort(trajectory traj);
```

Parameters

Parameter	Description
traj	The trajectory object.

Examples

```
select st_sort(ST_makeTrajectory('STPOINT'::leaftype, 'LINESTRING(-179.48077 51.72814,-179.
46731 51.74634,-179.46502 51.74934,-179.46183 51.75378,-179.45943 51.75736,-179.45560 51.76
273,-179.44845 51.77186,-179.43419 51.78977,-179.41259 51.81643,-179.41001 51.81941,-179.40
751 51.82223,-179.40497 51.82505,-179.40242 51.82796,-179.39981 51.83095,-179.39734 51.8339
8,-179.39499 51.83709)'::geometry, ARRAY['2017-01-15 09:06:39'::timestamp,'2017-01-15 09:14
:48'::timestamp,'2017-01-15 09:13:39'::timestamp,'2017-01-15 09:16:28'::timestamp,'2017-01-
15 09:19:48'::timestamp,'2017-01-15 09:17:48'::timestamp,'2017-01-15 09:23:19'::timestamp,'
2017-01-15 09:34:40'::timestamp,'2017-01-15 09:30:28'::timestamp,'2017-01-15 09:36:59'::tim
estamp,'2017-01-15 09:38:09'::timestamp,'2017-01-15 09:39:18'::timestamp,'2017-01-15 09:40:
40'::timestamp,'2017-01-15 09:47:38'::timestamp,'2017-01-15 21:18:30'::timestamp,'2017-01-1
5 09:48:49'::timestamp], '{"leafcount": 16, "attributes": {"heading": {"type": "integer", "
length": 4, "nullable": false,"value":[0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15]}}}'));

                                                                st_sort

-----------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------
 {"trajectory":{"version":1,"type":"STPOINT","leafcount":16,"start_time":"2017-01-15 09:06:
39","end_time":"2017-01-15 21:18:30","spatial":"LINESTRING(-179.48077 51.72814,-179.46502 5
1.7
4934,-179.46731 51.74634,-179.46183 51.75378,-179.4556 51.76273,-179.45943 51.75736,-179.44
845 51.77186,-179.41259 51.81643,-179.43419 51.78977,-179.41001 51.81941,-179.40751 51.8222
3,-
179.40497 51.82505,-179.40242 51.82796,-179.39981 51.83095,-179.39499 51.83709,-179.39734 5
1.83398)","timeline":["2017-01-15 09:06:39","2017-01-15 09:13:39","2017-01-15 09:14:48","20
17-
01-15 09:16:28","2017-01-15 09:17:48","2017-01-15 09:19:48","2017-01-15 09:23:19","2017-01-
15 09:30:28","2017-01-15 09:34:40","2017-01-15 09:36:59","2017-01-15 09:38:09","2017-01-15
09:
39:18","2017-01-15 09:40:40","2017-01-15 09:47:38","2017-01-15 09:48:49","2017-01-15 21:18:
30"],"attributes":{"leafcount":16,"heading":{"type":"integer","length":4,"nullable":false,"
val
ue":[0,2,1,3,5,4,6,8,7,9,10,11,12,13,15,14]}}}}
(1 row)
```

8.3.6. ST_removeDriftPoints

You can use this function to delete trajectory points that meet the specified threshold conditions from a trajectory object.

Syntax

```
trajectory ST_removeDriftPoints(trajectory traj, float8 speed, float8 dist, interval offset
);
```

Parameters

Parameter	Description
traj	The trajectory object.

Parameter	Description
speed	The speed threshold. The unit is meter per second.
dist	The distance threshold. The unit is meter.
offset	The time interval threshold.

Description

This function deletes the trajectory points that meet the specified conditions from a trajectory object and returns the new data of the trajectory object. The returned trajectory object contains at least two trajectory points. This function deletes a trajectory point if the trajectory point meets the following three conditions: The speed of the trajectory point is greater than the specified speed threshold, the distance from the trajectory point to the previous trajectory point is greater than the specified distance threshold, and the time interval is greater than the specified offset threshold. The unit of speed is meter per second and the unit of distance is meter. If the specified trajectory object has only two trajectory points, no trajectory point is deleted.

The speed parameter specifies a speed threshold. The speed threshold is calculated in real time based on the position of the trajectory point and the time offset. The unit of speed is meter per second. The dist parameter specifies a distance threshold. The unit of distance is meter. The offset parameter specifies a time interval. By default, the coordinate reference system of a trajectory object is World Geodetic System 1984 (WGS 84). If you do not want to use WGS 84, you can specify a spatial reference system identifier (SRID) when you create a trajectory object or invoke the ST_SetSRID function to set an SRID for the trajectory object. The system calculates the spherical distance between two points based on the specified SRID. The unit of spherical distances is meter.

Examples

```
select id, st_leafcount(traj) from table_name where id = 1;
 id | st_leafcount
----+--------------
  1 |           53
(1 row)
update table_name set traj=st_removeDriftPoints(traj,25.72, 9620, interval '30 seconds') wh
ere id = 1;
--Two trajectory points are deleted.
select id, st_leafcount(traj) from table_name where id = 1;
 id | st_leafcount
----+--------------
  1 |           49
(1 row)
```

8.3.7. ST_Split

The ST_Split function is used to split a trajectory into multiple sub-trajectories based on a spatial geometry object.

Syntax

```
trajectory[] ST_Split(trajectory traj, geometry geom, float8 radius_of_buffer);
trajectory[] ST_Split(trajectory traj, text config);
```

Parameters

Parameter	Description
traj	The trajectory that you want to split.
geom	The spatial geometry object that is used to split the trajectory. Only Point and MultiPoint geometry objects are supported.
radius_of_buffer	The radius of the buffer that is built based on the Point geometry object. Unit: meter.
config	The rule that specifies how to split the trajectory.

Description

- Configuration 1: `ST_Split(trajectory traj, geometry geom, float8 radius_of_buffer)`

  This function splits a trajectory based on the specified geometry object and returns an array of sub-trajectories. If the trajectory is split based on a MultiPoint spatial geometry object, the trajectory may be split into multiple segments.

- Configuration 2: `ST_Split(trajectory traj, text config)`

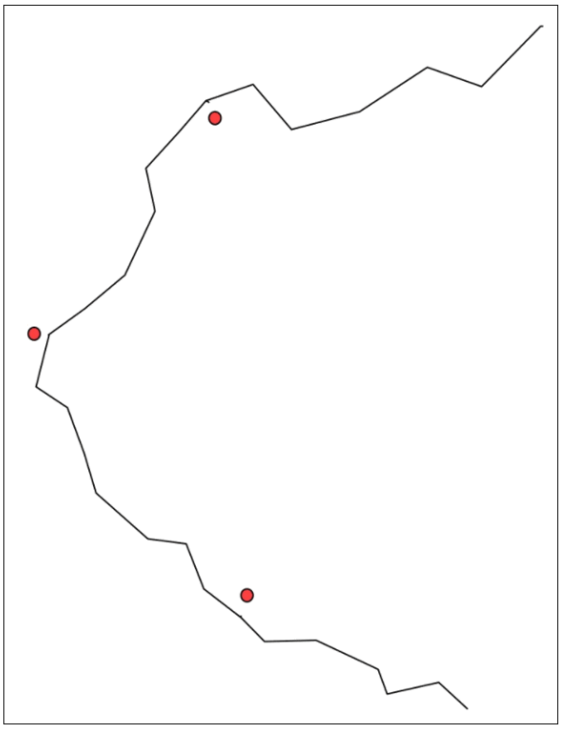
  This function splits a trajectory based on the specified rule and returns an array of sub-trajectories. Only one rule can be specified by the config parameter. The following table describes the rules that you can specify by using the config parameter.

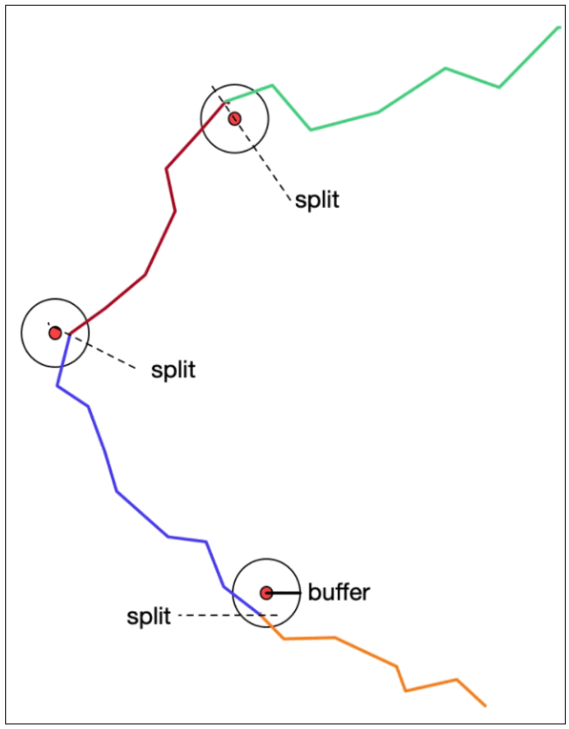
Rule name	Type	Description
cut_point.max_point	Positive integer	This rule specifies a quantity interval between sampling points. The function splits the trajectory into sub-trajectories based on the quantity interval that you specify.
cut_point.even_divide	Positive integer	This rule specifies the number of parts into which the trajectory is divided. The function evenly splits the trajectory into the specified number of sub-trajectories. If the number of edges of the trajectory is smaller than the value that is specified by the rule, the function considers each edge of the trajectory as a sub-trajectory.
cut_edge.time_interval	Positive time range	This rule specifies a time interval. The function splits the trajectory into sub-trajectories based on the time interval that you specify.

Rule name	Type	Description
cut_edge.geohash	Positive even number	This rule specifies a Geohash. The function splits the trajectory based on the Geohash that you specify in the rule and ensures that each sub-trajectory is included in one Geohash grid. If you specify this rule, make sure that the data of the trajectory is represented by a latitude and a longitude.
drop_edge.temporal_length	Time range	This rule specifies a time interval. The function deletes the part between the specified time interval from the trajectory.
drop_edge.spatial_distance_2d	Floating point	This rule specifies a spatial distance. The function deletes the part specified by the spatial distance from the trajectory. The spatial distance is a two-dimensional Euclidean distance.

Diagrams

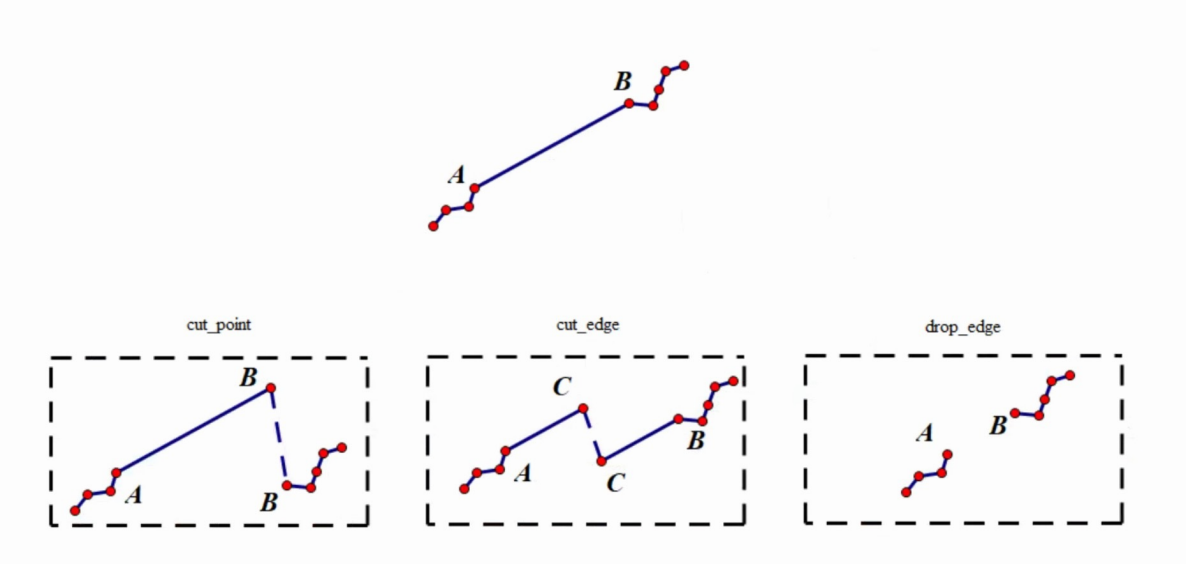
- Split a trajectory based on a specified geometry object.





- If you specify the cut_point.max_point or the cut_point.even_divide rule in the config parameter, the function splits the trajectory based on the specified sampling points. As shown in the following diagram, the function splits the trajectory into two sub-trajectories at Point B. If you specify the cut_edge.time_interval or the cut_edge.geohash rule in the config parameter, the function splits the trajectory based on the specified edges. As shown in the following diagram, the function splits the trajectory into two sub-trajectories at Point C. If you specify the drop_edge.temporal_length or the

drop_edge.spatial_distance_2d rule in the config parameter, the function deletes the specified edge between the specified sampling points. As shown in the following diagram, the function deletes the edge between Point A and Point B.



Example

```
create table tr_split_traj(id integer, traj trajectory);
INSERT INTO tr_split_traj VALUES(3, ST_MakeTrajectory('STPOINT'::leaftype, st_geomfromtext(
'LINESTRING(99.027 29.7555,99.313 29.9975,99.852 30.0745,104.879 35.0795,105.044 35.1235,10
5.187 35.0685,109.906 35.0795,110.071 35.1675,110.192 35.0355,110.544 35.0245,111.017 34.80
45)', 4326), ARRAY['2010-01-01 14:30'::timestamp,'2010-01-01 15:00','2010-01-01 15:10','201
0-01-01 15:20','2010-01-01 15:30','2010-01-01 15:40','2010-01-01 15:50','2010-01-01 16:00',
'2010-01-01 16:10','2010-01-01 16:20','2010-01-01 16:30'],'{"leafcount":11,"attributes":{"v
elocity": {"type": "integer", "length": 2,"nullable" : true,"value": [120, 130, 140, 150, 1
60, 170, 180, 190, 200, 210, 220]}}}'));
select id, unnest(st_split(traj, st_geomfromtext('MULTIPOINT(100 30,105 35,110 35)'), 23000
)) as subtraj from tr_split_traj;
 id |
subtraj

----+-----------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------
------------------
  3 | {"trajectory":{"version":1,"type":"STPOINT","leafcount":3,"start_time":"Fri Jan 01 14
:30:00 2010","end_time":"Fri Jan 01 15:10:00 2010","spatial":"SRID=4326;LINESTRING(99.027 2
9.7555,99.313 29.9975,99.852 30.0745)","timeline":["Fri Jan 01 14:30:00 2010","Fri Jan 01 1
5:00:00 2010","Fri Jan 01 15:10:00 2010"],"attributes":{"leafcount":3,"velocity":{"type":"i
nteger","length":2,"nullable":true,"value":[120,130,140]}}}}
  3 | {"trajectory":{"version":1,"type":"STPOINT","leafcount":2,"start_time":"Fri Jan 01 15
:10:00 2010","end_time":"Fri Jan 01 15:20:00 2010","spatial":"SRID=4326;LINESTRING(99.852 3
0.0745,104.879 35.0795)","timeline":["Fri Jan 01 15:10:00 2010","Fri Jan 01 15:20:00 2010"]
,"attributes":{"leafcount":2,"velocity":{"type":"integer","length":2,"nullable":true,"value
":[140,150]}}}}
  3 | {"trajectory":{"version":1,"type":"STPOINT","leafcount":2,"start_time":"Fri Jan 01 15
```

```
:40:00 2010","end_time":"Fri Jan 01 15:50:00 2010","spatial":"SRID=4326;LINESTRING(105.187
35.0685,109.906 35.0795)","timeline":["Fri Jan 01 15:40:00 2010","Fri Jan 01 15:50:00 2010"
],"attributes":{"leafcount":2,"velocity":{"type":"integer","length":2,"nullable":true,"valu
e":[170,180]}}}}
  3 | {"trajectory":{"version":1,"type":"STPOINT","leafcount":3,"start_time":"Fri Jan 01 16
:10:00 2010","end_time":"Fri Jan 01 16:30:00 2010","spatial":"SRID=4326;LINESTRING(110.192
35.0355,110.544 35.0245,111.017 34.8045)","timeline":["Fri Jan 01 16:10:00 2010","Fri Jan 0
1 16:20:00 2010","Fri Jan 01 16:30:00 2010"],"attributes":{"leafcount":3,"velocity":{"type"
:"integer","length":2,"nullable":true,"value":[200,210,220]}}}}
With traj as(
    select
        '{"trajectory":{"version":1,"type":"STPOINT","leafcount":19,"start_time":"2000-01-0
1 00:01:19.067179","end_time":"2000-01-01 03:24:25.946085","spatial":"LINESTRING(-100 -100
-100,-88.8925775739675 -86.6512698383691 -92.3767832526937,-79.6904716538265 -80.6515727923
252 -84.2357598245144,-75.8435507711644 -73.7572890928326 -80.5007370118983,-70.62384253212
56 -67.8213750167439 -74.5733173238113,-61.6014582272619 -61.0636760429479 -67.987423930317
2,-56.1098577060426 -54.4264591250879 -64.5007972046733,-46.9800617334743 -49.4026757289345
-61.6160059720278,-41.7122942996211 -46.3224360072054 -56.5283147455193,-35.5646221285375 -
38.1688933617746 -49.2775720101781,-31.7230528349367 -33.6970051738123 -44.1693710885011,-2
3.1585765127093 -26.5895827477798 -40.6539742602035,-16.7020264320696 -21.6133877349397 -37
.3055470525287,-12.1044529232507 -14.1236051704424 -28.2295028120279,-3.77185660181567 -7.7
4744770256802 -24.3842111621052,0.488159407706304 -3.68223926316326 -19.9478872027248,6.334
06881305078 4.54123636645575 -15.0410129944794,15.6666049417108 10.5611746329814 -11.277022
0567472,14 11 -10)","timeline":["2000-01-01 00:01:19.067179","2000-01-01 00:12:36.116007","
2000-01-01 00:23:53.164835","2000-01-01 00:35:10.213663","2000-01-01 00:46:27.262491","2000
-01-01 00:57:44.311319","2000-01-01 01:09:01.360147","2000-01-01 01:20:18.408975","2000-01-
01 01:31:35.457803","2000-01-01 01:42:52.506631","2000-01-01 01:54:09.555459","2000-01-01 0
2:05:26.604287","2000-01-01 02:16:43.653115","2000-01-01 02:28:00.701943","2000-01-01 02:39
:17.750771","2000-01-01 02:50:34.799599","2000-01-01 03:01:51.848427","2000-01-01 03:13:08.
897255","2000-01-01 03:24:25.946085"]}}'::trajectory as a
)
select unnest(ST_split(a, '{"cut_point.max_point":4}')) from traj;

unnest
-----------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------
 {"trajectory":{"version":1,"type":"STPOINT","leafcount":5,"start_time":"2000-01-01 00:01:1
9","end_time":"2000-01-01 00:46:27","spatial":"LINESTRING(-100 -100 -100,-88.8925775739675
-86.6512698383691 -92.3767832526937,-79.6904716538265 -80.6515727923252 -84.2357598245144,-
75.8435507711644 -73.7572890928326 -80.5007370118983,-70.6238425321256 -67.8213750167439 -7
4.5733173238113)","timeline":["2000-01-01 00:01:19","2000-01-01 00:12:36","2000-01-01 00:23
:53","2000-01-01 00:35:10","2000-01-01 00:46:27"]}}
 {"trajectory":{"version":1,"type":"STPOINT","leafcount":5,"start_time":"2000-01-01 00:46:2
7","end_time":"2000-01-01 01:31:35","spatial":"LINESTRING(-70.6238425321256 -67.82137501674
39 -74.5733173238113,-61.6014582272619 -61.0636760429479 -67.9874239303172,-56.109857706042
6 -54.4264591250879 -64.5007972046733,-46.9800617334743 -49.4026757289345 -61.6160059720278
,-41.7122942996211 -46.3224360072054 -56.5283147455193)","timeline":["2000-01-01 00:46:27",
"2000-01-01 00:57:44","2000-01-01 01:09:01","2000-01-01 01:20:18","2000-01-01 01:31:35"]}}
 {"trajectory":{"version":1,"type":"STPOINT","leafcount":5,"start_time":"2000-01-01 01:31:3
5","end_time":"2000-01-01 02:16:44","spatial":"LINESTRING(-41.7122942996211 -46.32243600720
```

```
54 -56.5283147455193,-35.5646221285375 -38.1688933617746 -49.2775720101781,-31.723052834936
7 -33.6970051738123 -44.1693710885011,-23.1585765127093 -26.5895827477798 -40.6539742602035
,-16.7020264320696 -21.6133877349397 -37.3055470525287)","timeline":["2000-01-01 01:31:35",
"2000-01-01 01:42:53","2000-01-01 01:54:10","2000-01-01 02:05:27","2000-01-01 02:16:44"]}}
 {"trajectory":{"version":1,"type":"STPOINT","leafcount":5,"start_time":"2000-01-01 02:16:4
4","end_time":"2000-01-01 03:01:52","spatial":"LINESTRING(-16.7020264320696 -21.61338773493
97 -37.3055470525287,-12.1044529232507 -14.1236051704424 -28.2295028120279,-3.7718566018156
7 -7.74744770256802 -24.3842111621052,0.488159407706304 -3.68223926316326 -19.9478872027248
,6.33406881305078 4.54123636645575 -15.0410129944794)","timeline":["2000-01-01 02:16:44","2
000-01-01 02:28:01","2000-01-01 02:39:18","2000-01-01 02:50:35","2000-01-01 03:01:52"]}}
 {"trajectory":{"version":1,"type":"STPOINT","leafcount":3,"start_time":"2000-01-01 03:01:5
2","end_time":"2000-01-01 03:24:26","spatial":"LINESTRING(6.33406881305078 4.54123636645575
-15.0410129944794,15.6666049417108 10.5611746329814 -11.2770220567472,14 11 -10)","timeline
":["2000-01-01 03:01:52","2000-01-01 03:13:09","2000-01-01 03:24:26"]}}
(5 rows)
```

8.4. Attribute metadata

8.4.1. ST_attrDefinition

This function returns the definitions of attributes in a trajectory object.

Syntax

```
text ST_attrDefinition(trajectory traj);
```

Parameters

Parameter	Description
traj	The trajectory object.

Examples

```
With traj AS (
  select '{"trajectory":{"version":1,"type":"STPOINT","leafcount":2,"start_time":"2010-01-0
1 11:30:00","end_time":"2010-01-01 12:30:00","spatial":"SRID=4326;LINESTRING(1 1,3 5)","tim
eline":["2010-01-01 11:30:00","2010-01-01 12:30:00"],"attributes":{"leafcount":2,"velocity"
:{"type":"integer","length":4,"nullable":true,"value":[1,null]},"speed":{"type":"float","le
ngth":8,"nullable":true,"value":[null,1.0]},"angle":{"type":"string","length":64,"nullable"
:true,"value":["test",null]}, "tngel2":{"type":"timestamp","length":8,"nullable":true,"valu
e":["2010-01-01 12:30:00",null]},"bearing":{"type":"bool","length":1,"nullable":true,"value
":[null,true]}}}}'::trajectory a)
Select st_attrDefinition(a) from traj;

st_attrdefinition

-----------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------
------------
 {"size":5,"velocity":{"type":"integer","length":4,"nullable":true},"speed":{"type":"float"
,"length":8,"nullable":true},"angle":{"type":"string","length":64,"nullable":true},"tngel2"
:{"type":"timestamp","length":8,"nullable":true},"bearing":{"type":"bool","length":1,"nulla
ble":true}}
(1 row)
```

8.4.2. ST_attrSize

This function returns the number of attributes in a trajectory object.

Syntax

```
integer ST_attrSize(trajectory traj);
```

Parameters

Parameter	Description
traj	The trajectory object.

Examples

```
With traj AS (
  select '{"trajectory":{"version":1,"type":"STPOINT","leafcount":2,"start_time":"2010-01-0
1 11:30:00","end_time":"2010-01-01 12:30:00","spatial":"SRID=4326;LINESTRING(1 1,3 5)","tim
eline":["2010-01-01 11:30:00","2010-01-01 12:30:00"],"attributes":{"leafcount":2,"velocity"
:{"type":"integer","length":4,"nullable":true,"value":[1,null]},"speed":{"type":"float","le
ngth":8,"nullable":true,"value":[null,1.0]},"angle":{"type":"string","length":64,"nullable"
:true,"value":["test",null]}, "tngel2":{"type":"timestamp","length":8,"nullable":true,"valu
e":["2010-01-01 12:30:00",null]},"bearing":{"type":"bool","length":1,"nullable":true,"value
":[null,true]}}}}'::trajectory a)
Select st_attrSize(a) from traj;
 st_attrsize
-------------
           5
(1 row)
```

8.4.3. ST_attrName

This function returns the name of a trajectory attribute field or the names of all trajectory attribute fields.

Syntax

```
text[] ST_attrName(trajectory traj);
text ST_attrName(trajectory traj, integer index);
```

Parameters

Parameter	Description
traj	The trajectory object.
index	The attribute index. The value starts from 0.

Examples

```
With traj AS (
  select '{"trajectory":{"version":1,"type":"STPOINT","leafcount":2,"start_time":"2010-01-0
1 11:30:00","end_time":"2010-01-01 12:30:00","spatial":"SRID=4326;LINESTRING(1 1,3 5)","tim
eline":["2010-01-01 11:30:00","2010-01-01 12:30:00"],"attributes":{"leafcount":2,"velocity"
:{"type":"integer","length":4,"nullable":true,"value":[1,null]},"speed":{"type":"float","le
ngth":8,"nullable":true,"value":[null,1.0]},"angle":{"type":"string","length":64,"nullable"
:true,"value":["test",null]}, "tngel2":{"type":"timestamp","length":8,"nullable":true,"valu
e":["2010-01-01 12:30:00",null]},"bearing":{"type":"bool","length":1,"nullable":true,"value
":[null,true]}}}}'::trajectory a)
Select st_attrName(a) from traj;
              st_attrname
---------------------------------------
 {velocity,speed,angle,tngel2,bearing}
(1 row)
With traj AS (
  select '{"trajectory":{"version":1,"type":"STPOINT","leafcount":2,"start_time":"2010-01-0
1 11:30:00","end_time":"2010-01-01 12:30:00","spatial":"SRID=4326;LINESTRING(1 1,3 5)","tim
eline":["2010-01-01 11:30:00","2010-01-01 12:30:00"],"attributes":{"leafcount":2,"velocity"
:{"type":"integer","length":4,"nullable":true,"value":[1,null]},"speed":{"type":"float","le
ngth":8,"nullable":true,"value":[null,1.0]},"angle":{"type":"string","length":64,"nullable"
:true,"value":["test",null]}, "tngel2":{"type":"timestamp","length":8,"nullable":true,"valu
e":["2010-01-01 12:30:00",null]},"bearing":{"type":"bool","length":1,"nullable":true,"value
":[null,true]}}}}'::trajectory a)
Select st_attrName(a, 0) from traj;
 st_attrname
-------------
 velocity
(1 row)
```

8.4.4. ST_attrType

This function returns the data type of a trajectory attribute field.

Syntax

```
text ST_attrType(trajectory traj, integer index);
text ST_attrType(trajectory traj, text name);
```

Parameters

Parameter	Description
traj	The trajectory object.
index	The attribute index.
name	The name of the attribute field.

Description

This function returns one of the following strings to indicate the data type of the trajectory attribute field:

- integer
- float
- string
- timestamp
- bool

Examples

```
With traj AS (
  select '{"trajectory":{"version":1,"type":"STPOINT","leafcount":2,"start_time":"2010-01-0
1 11:30:00","end_time":"2010-01-01 12:30:00","spatial":"SRID=4326;LINESTRING(1 1,3 5)","tim
eline":["2010-01-01 11:30:00","2010-01-01 12:30:00"],"attributes":{"leafcount":2,"velocity"
:{"type":"integer","length":4,"nullable":true,"value":[1,null]},"speed":{"type":"float","le
ngth":8,"nullable":true,"value":[null,1.0]},"angle":{"type":"string","length":64,"nullable"
:true,"value":["test",null]}, "tngel2":{"type":"timestamp","length":8,"nullable":true,"valu
e":["2010-01-01 12:30:00",null]},"bearing":{"type":"bool","length":1,"nullable":true,"value
":[null,true]}}}}'::trajectory a)
Select st_attrType(a, 0) from traj;
 st_attrtype
-------------
 integer
(1 row)
With traj AS (
  select '{"trajectory":{"version":1,"type":"STPOINT","leafcount":2,"start_time":"2010-01-0
1 11:30:00","end_time":"2010-01-01 12:30:00","spatial":"SRID=4326;LINESTRING(1 1,3 5)","tim
eline":["2010-01-01 11:30:00","2010-01-01 12:30:00"],"attributes":{"leafcount":2,"velocity"
:{"type":"integer","length":4,"nullable":true,"value":[1,null]},"speed":{"type":"float","le
ngth":8,"nullable":true,"value":[null,1.0]},"angle":{"type":"string","length":64,"nullable"
:true,"value":["test",null]}, "tngel2":{"type":"timestamp","length":8,"nullable":true,"valu
e":["2010-01-01 12:30:00",null]},"bearing":{"type":"bool","length":1,"nullable":true,"value
":[null,true]}}}}'::trajectory a)
Select st_attrType(a, 'velocity') from traj;
 st_attrtype
-------------
 integer
(1 row)
```

8.4.5. ST_attrLength

This function returns the defined length of a trajectory attribute field.

Syntax

```
integer ST_attrLength(trajectory traj, integer index);
integer ST_attrLength(trajectory traj, text name);
```

Parameters

Parameter	Description
traj	The trajectory object.
index	The attribute index.
name	The name of the attribute field.

Examples

```
With traj AS (
  select '{"trajectory":{"version":1,"type":"STPOINT","leafcount":2,"start_time":"2010-01-0
1 11:30:00","end_time":"2010-01-01 12:30:00","spatial":"SRID=4326;LINESTRING(1 1,3 5)","tim
eline":["2010-01-01 11:30:00","2010-01-01 12:30:00"],"attributes":{"leafcount":2,"velocity"
:{"type":"integer","length":4,"nullable":true,"value":[1,null]},"speed":{"type":"float","le
ngth":8,"nullable":true,"value":[null,1.0]},"angle":{"type":"string","length":64,"nullable"
:true,"value":["test",null]}, "tngel2":{"type":"timestamp","length":8,"nullable":true,"valu
e":["2010-01-01 12:30:00",null]},"bearing":{"type":"bool","length":1,"nullable":true,"value
":[null,true]}}}}'::trajectory a)
Select st_attrLength(a, 0) from traj;
 st_attrlength
---------------
             4
(1 row)
With traj AS (
  select '{"trajectory":{"version":1,"type":"STPOINT","leafcount":2,"start_time":"2010-01-0
1 11:30:00","end_time":"2010-01-01 12:30:00","spatial":"SRID=4326;LINESTRING(1 1,3 5)","tim
eline":["2010-01-01 11:30:00","2010-01-01 12:30:00"],"attributes":{"leafcount":2,"velocity"
:{"type":"integer","length":4,"nullable":true,"value":[1,null]},"speed":{"type":"float","le
ngth":8,"nullable":true,"value":[null,1.0]},"angle":{"type":"string","length":64,"nullable"
:true,"value":["test",null]}, "tngel2":{"type":"timestamp","length":8,"nullable":true,"valu
e":["2010-01-01 12:30:00",null]},"bearing":{"type":"bool","length":1,"nullable":true,"value
":[null,true]}}}}'::trajectory a)
Select st_attrLength(a, 'velocity') from traj;
 st_attrlength
---------------
             4
(1 row)
```

8.4.6. ST_attrNullable

This function returns true if a trajectory attribute field can be null.

Syntax

```
bool ST_attrNullable(trajectory traj, integer index);
bool ST_attrNullable(trajectory traj, text name);
```

Parameters

Parameter	Description
traj	The trajectory object.
index	The attribute index.
name	The name of the attribute field.

Examples

```
With traj AS (
  select '{"trajectory":{"version":1,"type":"STPOINT","leafcount":2,"start_time":"2010-01-0
1 11:30:00","end_time":"2010-01-01 12:30:00","spatial":"SRID=4326;LINESTRING(1 1,3 5)","tim
eline":["2010-01-01 11:30:00","2010-01-01 12:30:00"],"attributes":{"leafcount":2,"velocity"
:{"type":"integer","length":4,"nullable":true,"value":[1,null]},"speed":{"type":"float","le
ngth":8,"nullable":true,"value":[null,1.0]},"angle":{"type":"string","length":64,"nullable"
:true,"value":["test",null]}, "tngel2":{"type":"timestamp","length":8,"nullable":true,"valu
e":["2010-01-01 12:30:00",null]},"bearing":{"type":"bool","length":1,"nullable":true,"value
":[null,true]}}}}'::trajectory a)
Select st_attrNullable(a, 'velocity') from traj;
 st_attrnullable
-----------------
 t
(1 row)
With traj AS (
  select '{"trajectory":{"version":1,"type":"STPOINT","leafcount":2,"start_time":"2010-01-0
1 11:30:00","end_time":"2010-01-01 12:30:00","spatial":"SRID=4326;LINESTRING(1 1,3 5)","tim
eline":["2010-01-01 11:30:00","2010-01-01 12:30:00"],"attributes":{"leafcount":2,"velocity"
:{"type":"integer","length":4,"nullable":true,"value":[1,null]},"speed":{"type":"float","le
ngth":8,"nullable":true,"value":[null,1.0]},"angle":{"type":"string","length":64,"nullable"
:true,"value":["test",null]}, "tngel2":{"type":"timestamp","length":8,"nullable":true,"valu
e":["2010-01-01 12:30:00",null]},"bearing":{"type":"bool","length":1,"nullable":true,"value
":[null,true]}}}}'::trajectory a)
Select st_attrNullable(a, 1) from traj;
 st_attrnullable
-----------------
 t
(1 row)
```

8.5. Event functions

8.5.1. ST_addEvent

This function adds an event to a trajectory object.

Syntax

```
trajectory ST_addEvent(trajectory traj, integer event_type, timestamp event_time);
```

Parameters

Parameter	Description
traj	The trajectory object.
event_type	The event type ID.
event_time	The event timestamp.

Description

Event type IDs are predefined. For example, 1000 indicates unlocking and 2000 indicates locking.

Examples

```
With traj AS (
  select '{"trajectory":{"version":1,"type":"STPOINT","leafcount":2,"start_time":"2010-01-0
1 11:30:00","end_time":"2010-01-01 12:30:00","spatial":"SRID=4326;LINESTRING(1 1,3 5)","tim
eline":["2010-01-01 11:30:00","2010-01-01 12:30:00"],"attributes":{"leafcount":2,"velocity"
:{"type":"integer","length":4,"nullable":true,"value":[1,null]},"speed":{"type":"float","le
ngth":8,"nullable":true,"value":[null,1.0]},"angle":{"type":"string","length":64,"nullable"
:true,"value":["test",null]}, "tngel2":{"type":"timestamp","length":8,"nullable":true,"valu
e":["2010-01-01 12:30:00",null]},"bearing":{"type":"bool","length":1,"nullable":true,"value
":[null,true]}}}}'::trajectory a)
Select st_addevent(a, 1, '2010-01-01 11:30:00') from traj;

st_addevent

-----------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------
----------------------------------------------
 {"trajectory":{"version":1,"type":"STPOINT","leafcount":2,"start_time":"2010-01-01 11:30:0
0","end_time":"2010-01-01 12:30:00","spatial":"SRID=4326;LINESTRING(1 1,3 5)","timeline":["
2010-01-01 11:30:00","2010-01-01 12:30:00"],"attributes":{"leafcount":2,"velocity":{"type":
"integer","length":4,"nullable":true,"value":[1,null]},"speed":{"type":"float","length":8,"
nullable":true,"value":[null,1.0]},"angle":{"type":"string","length":64,"nullable":true,"va
lue":["test",null]},"tngel2":{"type":"timestamp","length":8,"nullable":true,"value":["2010-
01-01 12:30:00",null]},"bearing":{"type":"bool","length":1,"nullable":true,"value":[null,tr
ue]}},"events":[{"1":"2010-01-01 11:30:00"}]}}
(1 row)
```

8.5.2. ST_eventTimes

This function returns the time of all events in a trajectory object.

Syntax

```
timestamp[] ST_eventTimes(trajectory traj);
```

Parameters

Parameter	Description
traj	The trajectory object.

Examples

```
With traj AS (
  select '{"trajectory":{"version":1,"type":"STPOINT","leafcount":2,"start_time":"2010-01-0
1 11:30:00","end_time":"2010-01-01 12:30:00","spatial":"SRID=4326;LINESTRING(1 1,3 5)","tim
eline":["2010-01-01 11:30:00","2010-01-01 12:30:00"],"attributes":{"leafcount":2,"velocity"
:{"type":"integer","length":4,"nullable":true,"value":[1,null]},"speed":{"type":"float","le
ngth":8,"nullable":true,"value":[null,1.0]},"angle":{"type":"string","length":64,"nullable"
:true,"value":["test",null]}, "tngel2":{"type":"timestamp","length":8,"nullable":true,"valu
e":["2010-01-01 12:30:00",null]},"bearing":{"type":"bool","length":1,"nullable":true,"value
":[null,true]}},"events":[{"1":"Fri Jan 01 14:30:00 2010"},{"2":"Fri Jan 01 14:30:00 2010"}
]}}'::trajectory a)
Select st_eventTimes(a) from traj;
                 st_eventtimes
-----------------------------------------------
 {"2010-01-01 14:30:00","2010-01-01 14:30:00"}
(1 row)
```

8.5.3. ST_eventTime

This function returns the time of an event with the specified index in a trajectory object.

Syntax

```
timestamp ST_eventTime(trajectory traj, integer index);
```

Parameters

Parameter	Description
traj	The trajectory object.
index	The event index.

Examples

```
With traj AS (
  select '{"trajectory":{"version":1,"type":"STPOINT","leafcount":2,"start_time":"2010-01-0
1 11:30:00","end_time":"2010-01-01 12:30:00","spatial":"SRID=4326;LINESTRING(1 1,3 5)","tim
eline":["2010-01-01 11:30:00","2010-01-01 12:30:00"],"attributes":{"leafcount":2,"velocity"
:{"type":"integer","length":4,"nullable":true,"value":[1,null]},"speed":{"type":"float","le
ngth":8,"nullable":true,"value":[null,1.0]},"angle":{"type":"string","length":64,"nullable"
:true,"value":["test",null]}, "tngel2":{"type":"timestamp","length":8,"nullable":true,"valu
e":["2010-01-01 12:30:00",null]},"bearing":{"type":"bool","length":1,"nullable":true,"value
":[null,true]}},"events":[{"1":"Fri Jan 01 14:30:00 2010"}]}}'::trajectory a)
Select st_eventTime(a, 0) from traj;
    st_eventtime
---------------------
 2010-01-01 14:30:00
(1 row)
```

8.5.4. ST_eventTypes

This function returns the type IDs of all events in a trajectory object.

Syntax

```
integer[] ST_eventTypes( trajectory traj);
```

Parameters

Parameter	Description
traj	The trajectory object.

Examples

```
With traj AS (
  select '{"trajectory":{"version":1,"type":"STPOINT","leafcount":2,"start_time":"2010-01-0
1 11:30:00","end_time":"2010-01-01 12:30:00","spatial":"SRID=4326;LINESTRING(1 1,3 5)","tim
eline":["2010-01-01 11:30:00","2010-01-01 12:30:00"],"attributes":{"leafcount":2,"velocity"
:{"type":"integer","length":4,"nullable":true,"value":[1,null]},"speed":{"type":"float","le
ngth":8,"nullable":true,"value":[null,1.0]},"angle":{"type":"string","length":64,"nullable"
:true,"value":["test",null]}, "tngel2":{"type":"timestamp","length":8,"nullable":true,"valu
e":["2010-01-01 12:30:00",null]},"bearing":{"type":"bool","length":1,"nullable":true,"value
":[null,true]}},"events":[{"1":"Fri Jan 01 14:30:00 2010"},{"2":"Fri Jan 01 14:30:00 2010"}
]}}'::trajectory a)
Select st_eventTypes(a) from traj;
 st_eventtypes
---------------
 {1,2}
```

8.5.5. ST_eventType

This function returns the type ID of an event with the specified index in a trajectory object.

Syntax

```
integer ST_eventType(trajectory traj, integer index);
```

Parameters

Parameter	Description
traj	The trajectory object.
index	The event index.

Examples

```
With traj AS (
  select '{"trajectory":{"version":1,"type":"STPOINT","leafcount":2,"start_time":"2010-01-0
1 11:30:00","end_time":"2010-01-01 12:30:00","spatial":"SRID=4326;LINESTRING(1 1,3 5)","tim
eline":["2010-01-01 11:30:00","2010-01-01 12:30:00"],"attributes":{"leafcount":2,"velocity"
:{"type":"integer","length":4,"nullable":true,"value":[1,null]},"speed":{"type":"float","le
ngth":8,"nullable":true,"value":[null,1.0]},"angle":{"type":"string","length":64,"nullable"
:true,"value":["test",null]}, "tngel2":{"type":"timestamp","length":8,"nullable":true,"valu
e":["2010-01-01 12:30:00",null]},"bearing":{"type":"bool","length":1,"nullable":true,"value
":[null,true]}},"events":[{"1":"Fri Jan 01 14:30:00 2010"},{"2":"Fri Jan 01 14:30:00 2010"}
]}}'::trajectory a)
Select st_eventType(a, 0) from traj;
 st_eventtype
--------------
            1
```

8.6. Attribute functions

8.6.1. ST_startTime

This function returns the start time of a trajectory object.

Syntax

```
timestamp ST_startTime(trajectory traj);
```

Parameters

Parameter	Description
traj	The trajectory object.

Examples

```
Select ST_startTime(traj) From traj_table;
```

8.6.2. ST_endTime

This function returns the end time of a trajectory object.

Syntax

```
timestamp ST_endTime(trajectory traj);
```

Parameters

Parameter	Description
traj	The trajectory object.

Examples

```
Select ST_endTime(traj) From traj_table;
```

8.6.3. ST_trajectorySpatial

This function returns the spatial geometry object of a trajectory object.

Syntax

```
geometry ST_trajectorySpatial(trajectory traj);
geometry ST_trajSpatial(trajectory traj);
```

Parameters

Parameter	Description
traj	The trajectory object.

Example

```
Select ST_AsText(ST_trajectorySpatial(traj)) FROM traj_table;
            st_astext
----------------------------------
 LINESTRING(114 35,115 36,116 37)
```

8.6.4. ST_trajectoryTemporal

This function returns the timeline of a trajectory object.

Syntax

```
text ST_trajectoryTemporal(trajectory traj);
text ST_trajTemporal(trajectory traj);
```

Parameters

Parameter	Description
traj	The trajectory object.

Examples

```
select ST_trajectoryTemporal(ST_MakeTrajectory('STPOINT'::leaftype, st_geomfromtext('LINEST
RING (114 35, 115 36, 116 37)', 4326), '[2010-01-01 14:30, 2010-01-01 15:30)'::tsrange, nul
l));
                              st_trajectorytemporal
-----------------------------------------------------------------------------
 {"timeline":["2010-01-01 14:30:00","2010-01-01 15:00:00","2010-01-01 15:30:00"]}
(1 row)
```

8.6.5. ST_trajAttrs

This function returns the attributes of a trajectory object.

Syntax

```
text ST_trajAttrs(trajectory traj);
```

Parameters

Parameter	Description
traj	The trajectory object.

Examples

```
Select ST_trajAttrs(traj) From traj_table;
```

8.6.6. ST_attrIntMax

This function returns the maximum value of an integer-type attribute field.

Syntax

```
int8 ST_attrIntMax(trajectory traj, cstring attr_field_name);
```

Parameters

Parameter	Description
traj	The trajectory object.
attr_field_name	The name of the attribute field.

Examples

```
select st_attrIntMax(ST_makeTrajectory('STPOINT'::leaftype, 'LINESTRING(-179.48077 51.72814
,-179.46731 51.74634,-179.46502 51.74934,-179.46183 51.75378,-179.45943 51.75736,-179.45560
51.76273,-179.44845 51.77186,-179.43419 51.78977,-179.41259 51.81643,-179.41001 51.81941,-1
79.40751 51.82223,-179.40497 51.82505,-179.40242 51.82796,-179.39981 51.83095,-179.39734 51
.83398,-179.39499 51.83709)'::geometry, ARRAY['2017-01-15 09:06:39'::timestamp,'2017-01-15
09:14:48','2017-01-15 09:13:39','2017-01-15 09:16:28','2017-01-15 09:19:48','2017-01-15 09:
17:48','2017-01-15 09:23:19','2017-01-15 09:34:40','2017-01-15 09:30:28','2017-01-15 09:36:
59','2017-01-15 09:38:09','2017-01-15 09:39:18','2017-01-15 09:40:40','2017-01-15 09:47:38'
,'2017-01-15 21:18:30','2017-01-15 09:48:49'], '{"leafcount": 16, "attributes": {"heading":
{"type": "integer", "length": 4, "nullable": false,"value":[0,1,2,3,4,5,6,7,8,9,10,11,12,13
,14,15]}}}'), 'heading');
 st_attrintmax
---------------
            15
```

8.6.7. ST_attrIntMin

This function returns the minimum value of an integer-type attribute field.

Syntax

```
int8 ST_attrIntMin(trajectory traj, cstring attr_field_name);
```

Parameters

Parameter	Description
traj	The trajectory object.
attr_field_name	The name of the attribute field.

Examples

```
select st_attrIntMin(ST_makeTrajectory('STPOINT'::leaftype, 'LINESTRING(-179.48077 51.72814
,-179.46731 51.74634,-179.46502 51.74934,-179.46183 51.75378,-179.45943 51.75736,-179.45560
51.76273,-179.44845 51.77186,-179.43419 51.78977,-179.41259 51.81643,-179.41001 51.81941,-1
79.40751 51.82223,-179.40497 51.82505,-179.40242 51.82796,-179.39981 51.83095,-179.39734 51
.83398,-179.39499 51.83709)'::geometry, ARRAY['2017-01-15 09:06:39'::timestamp,'2017-01-15
09:14:48','2017-01-15 09:13:39','2017-01-15 09:16:28','2017-01-15 09:19:48','2017-01-15 09:
17:48','2017-01-15 09:23:19','2017-01-15 09:34:40','2017-01-15 09:30:28','2017-01-15 09:36:
59','2017-01-15 09:38:09','2017-01-15 09:39:18','2017-01-15 09:40:40','2017-01-15 09:47:38'
,'2017-01-15 21:18:30','2017-01-15 09:48:49'], '{"leafcount": 16, "attributes": {"heading":
{"type": "integer", "length": 4, "nullable": false,"value":[0,1,2,3,4,5,6,7,8,9,10,11,12,13
,14,15]}}}'), 'heading');
 st_attrintmin
---------------
             0
```

8.6.8. ST_attrIntAverage

This function returns the average value of an integer-type attribute field.

Syntax

```
int8 ST_attrIntAverage(trajectory traj, cstring attr_field_name);
```

Parameters

Parameter	Description
traj	The trajectory object.
attr_field_name	The name of the attribute field.

Examples

```
select st_attrIntAverage(ST_makeTrajectory('STPOINT'::leaftype, 'LINESTRING(-179.48077 51.7
2814,-179.46731 51.74634,-179.46502 51.74934,-179.46183 51.75378,-179.45943 51.75736,-179.4
5560 51.76273,-179.44845 51.77186,-179.43419 51.78977,-179.41259 51.81643,-179.41001 51.819
41,-179.40751 51.82223,-179.40497 51.82505,-179.40242 51.82796,-179.39981 51.83095,-179.397
34 51.83398,-179.39499 51.83709)'::geometry, ARRAY['2017-01-15 09:06:39'::timestamp,'2017-0
1-15 09:14:48','2017-01-15 09:13:39','2017-01-15 09:16:28','2017-01-15 09:19:48','2017-01-1
5 09:17:48','2017-01-15 09:23:19','2017-01-15 09:34:40','2017-01-15 09:30:28','2017-01-15 0
9:36:59','2017-01-15 09:38:09','2017-01-15 09:39:18','2017-01-15 09:40:40','2017-01-15 09:4
7:38','2017-01-15 21:18:30','2017-01-15 09:48:49'], '{"leafcount": 16, "attributes": {"head
ing": {"type": "integer", "length": 4, "nullable": false,"value":[0,1,2,3,4,5,6,7,8,9,10,11
,12,13,14,15]}}}'), 'heading');
 st_attrintaverage
-------------------
                 7
```

8.6.9. ST_attrFloatMax

This function returns the maximum value of a float-type attribute field.

Syntax

```
float8 ST_attrFloatMax(trajectory traj, cstring attr_field_name);
```

Parameters

Parameter	Description
traj	The trajectory object.
attr_field_name	The name of the attribute field.

Examples

```
select st_attrFloatMax(ST_makeTrajectory('STPOINT'::leaftype, 'LINESTRING(-179.48077 51.728
14,-179.46731 51.74634,-179.46502 51.74934,-179.46183 51.75378,-179.45943 51.75736,-179.455
60 51.76273,-179.44845 51.77186,-179.43419 51.78977,-179.41259 51.81643,-179.41001 51.81941
,-179.40751 51.82223,-179.40497 51.82505,-179.40242 51.82796,-179.39981 51.83095,-179.39734
51.83398,-179.39499 51.83709)'::geometry, ARRAY['2017-01-15 09:06:39'::timestamp,'2017-01-1
5 09:13:39','2017-01-15 09:14:48','2017-01-15 09:16:28','2017-01-15 09:17:48','2017-01-15 0
9:19:48','2017-01-15 09:23:19','2017-01-15 09:30:28','2017-01-15 09:34:40','2017-01-15 09:3
6:59','2017-01-15 09:38:09','2017-01-15 09:39:18','2017-01-15 09:40:40','2017-01-15 09:47:3
8','2017-01-15 09:48:49','2017-01-15 21:18:30'], '{"leafcount": 16, "attributes": {"heading
": {"type": "float", "length": 4, "nullable": false,"value": [23.0,23.0,23.0,23.0,21.0,21.0
,72.0,72.0,72.0,72.0,73.0,74.0,73.0,73.0,73.0,73.0]}}}'),'heading');
 st_attrfloatmax
-----------------
               74
```

8.6.10. ST_attrFloatMin

This function returns the minimum value of a float-type attribute field.

Syntax

```
float8 ST_attrFloatMax(trajectory traj, cstring attr_field_name);
```

Parameters

Parameter	Description
traj	The trajectory object.
attr_field_name	The name of the attribute field.

Examples

```
select st_attrFloatMin(ST_makeTrajectory('STPOINT'::leaftype, 'LINESTRING(-179.48077 51.728
14,-179.46731 51.74634,-179.46502 51.74934,-179.46183 51.75378,-179.45943 51.75736,-179.455
60 51.76273,-179.44845 51.77186,-179.43419 51.78977,-179.41259 51.81643,-179.41001 51.81941
,-179.40751 51.82223,-179.40497 51.82505,-179.40242 51.82796,-179.39981 51.83095,-179.39734
51.83398,-179.39499 51.83709)'::geometry, ARRAY['2017-01-15 09:06:39'::timestamp,'2017-01-1
5 09:13:39','2017-01-15 09:14:48','2017-01-15 09:16:28','2017-01-15 09:17:48','2017-01-15 0
9:19:48','2017-01-15 09:23:19','2017-01-15 09:30:28','2017-01-15 09:34:40','2017-01-15 09:3
6:59','2017-01-15 09:38:09','2017-01-15 09:39:18','2017-01-15 09:40:40','2017-01-15 09:47:3
8','2017-01-15 09:48:49','2017-01-15 21:18:30'], '{"leafcount": 16, "attributes": {"heading
": {"type": "float", "length": 4, "nullable": false,"value": [23.0,23.0,23.0,23.0,21.0,21.0
,72.0,72.0,72.0,72.0,73.0,74.0,73.0,73.0,73.0,73.0]}}}'),'heading');
 st_attrfloatmin
-----------------
               21
```

8.6.11. ST_attrFloatAverage

This function returns the average value of a float-type attribute field.

Syntax

```
float8 ST_attrFloatAverage(trajectory traj, cstring attr_field_name);
```

Parameters

Parameter	Description
traj	The trajectory object.
attr_field_name	The name of the attribute field.

Examples

```
select st_attrFloatAverage(ST_makeTrajectory('STPOINT'::leaftype, 'LINESTRING(-179.48077 51
.72814,-179.46731 51.74634,-179.46502 51.74934,-179.46183 51.75378,-179.45943 51.75736,-179
.45560 51.76273,-179.44845 51.77186,-179.43419 51.78977,-179.41259 51.81643,-179.41001 51.8
1941,-179.40751 51.82223,-179.40497 51.82505,-179.40242 51.82796,-179.39981 51.83095,-179.3
9734 51.83398,-179.39499 51.83709)'::geometry, ARRAY['2017-01-15 09:06:39'::timestamp,'2017
-01-15 09:13:39','2017-01-15 09:14:48','2017-01-15 09:16:28','2017-01-15 09:17:48','2017-01
-15 09:19:48','2017-01-15 09:23:19','2017-01-15 09:30:28','2017-01-15 09:34:40','2017-01-15
09:36:59','2017-01-15 09:38:09','2017-01-15 09:39:18','2017-01-15 09:40:40','2017-01-15 09:
47:38','2017-01-15 09:48:49','2017-01-15 21:18:30'], '{"leafcount": 16, "attributes": {"hea
ding": {"type": "float", "length": 4, "nullable": false,"value": [23.0,23.0,23.0,23.0,21.0,
21.0,72.0,72.0,72.0,72.0,73.0,74.0,73.0,73.0,73.0,73.0]}}}'),'heading');
 st_attrfloataverage
---------------------
             53.8125
(1 row)
```

8.6.12. ST_leafType

This function returns the leaf type of a trajectory object.

Syntax

```
text ST_leafType(trajectory traj);
```

Parameters

Parameter	Description
traj	The trajectory object.

Only the ST_POINT type is supported.

Examples

```
select st_leafType(traj) from traj where id = 3;
 st_leaftype
 -------------
STPOINT
(1 row)
```

8.6.13. ST_leafCount

This function returns the number of leaves (trajectory points) in a trajectory object.

Syntax

```
integer ST_leafCount(trajectory traj);
```

Parameters

Parameter	Description
traj	The trajectory object.

Examples

```
select st_leafCount(traj) from traj where id = 2;
 st_leafcount
--------------
16
```

8.6.14. ST_duration

This function returns the duration of a trajectory object.

Syntax

```
interval ST_duration(trajectory traj);
```

Parameters

Parameter	Description
traj	The trajectory object.

Examples

```
select st_duration(traj) from traj where id = 2;
 st_duration
-------------
 00:42:10
(1 row)
```

8.6.15. ST_timeAtPoint

This function returns a set of time points when a trajectory object passes through the specified spatial point.

Syntax

```
timestamp[] ST_timeAtPoint(trajectory traj, geometry g);
```

Parameters

Parameter	Description
traj	The trajectory object.
g	The spatial point.

Example

```
Select ST_timeAtPoint(traj, st_geomfromtext('POINT(115 36)')) from traj_table;
     st_timeatpoint
-------------------------
 {"2010-01-01 15:00:00"}
```

8.6.16. ST_pointAtTime

This function returns the spatial geometry object of a trajectory object at the specified time point.

Syntax

```
geometry ST_pointAtTime(trajectory traj, timestamp t);
```

Parameters

Parameter	Description
traj	The trajectory object.
t	The time point.

This function returns a spatial point.

Examples

```
Select ST_pointAtTime(traj, '2010-1-11 23:40:00') from traj_table;
```

8.6.17. ST_velocityAtTime

This function returns the value of the velocity attribute field at the specified time point.

Syntax

```
float8 ST_velocityAtTime(trajectory traj, timestamp t);
```

Parameters

Parameter	Description
traj	The trajectory object.
t	The time point.

Examples

```
Select ST_velocityAtTime(traj, '2010-1-11 23:40:00') From traj_table;
```

8.6.18. ST_accelerationAtTime

This function returns the value of the acceleration attribute field at the specified time point.

Syntax

```
float8 ST_accelerationAtTime (trajectory traj, timestamp t);
```

Parameters

Parameter	Description
traj	The trajectory object.
t	The specified time point.

Example

```
Select ST_accelerationAtTime(traj,'2010-1-11 23:40:00') From traj_table;
```

8.6.19. ST_timeToDistance

This function returns a line chart in which the time is the x-axis and the Euclidean distance is the y-axis.

Syntax

```
geometry ST_timeToDistance(trajectory traj);
```

Parameters

Parameter	Description
traj	The trajectory object.

Examples

```
Select ST_timeToDistance(traj) from traj_table;
```

8.6.20. ST_timeAtDistance

This function returns the time point when the movement arrives at a spatial point from the start point for the specified distance.

Syntax

```
timestamp[] ST_timeAtDistance(trajectory traj, float8 d);
```

Parameters

Parameter	Description
traj	The trajectory object.
d	The distance of movement.

This function returns an array of timestamps. Multiple spatial points may be at the same distance from the start point.

Examples

```
Select ST_timeAtDistance(traj, 100) from traj_table;
```

8.6.21. ST_cumulativeDistanceAtTime

This function returns the cumulative distance of movement from the start point to the spatial point arrived at the specified time point.

Syntax

```
float8 ST_cumulativeDistanceAtTime(trajectory traj, timestamp t);
```

Parameters

Parameter	Description
traj	The trajectory object.
t	The time point.

Examples

```
Select ST_cumulativeDistanceAtTime(traj, '2011-11-1 04:30:00') from traj_table;
```

8.6.22. ST_timeAtCumulativeDistance

This function returns the time point when the movement arrives at a spatial point from the start point for the specified cumulative distance.

Syntax

```
timestamp ST_timeAtCumulativeDistance(trajectory traj, float d);
```

Parameters

Parameter	Description
traj	The trajectory object.
d	The cumulative distance of movement.

Examples

```
Select ST_timeAtCumulativeDistance(traj, 100.0) from traj_table;
```

8.6.23. ST_subTrajectory

This function returns the sub-trajectory of a trajectory object within the specified time range.

Syntax

```
trajectory ST_subTrajectory(trajectory traj, timestamp starttime, timestamp endtime);
trajectory ST_subTrajectory(trajectory traj, tsrange range);
```

Parameters

Parameter	Description
traj	The trajectory object.
starttime	The start time.
endtime	The end time.
range	The time range.

Examples

```
Select ST_subTrajectory(traj, '2010-1-11 02:45:30', '2010-1-11 03:00:00') FROM traj_table;
```

8.6.24. ST_subTrajectorySpatial

This function returns the spatial geometry object of a trajectory object within the specified time range.

Syntax

```
geometry ST_subTrajectorySpatial(trajectory traj, timestamp starttime, timestamp endtime);
geometry ST_subTrajectorySpatial(trajectory traj, tsrange range);
```

Parameters

Parameter	Description
traj	The trajectory object.
starttime	The start time.
endtime	The end time.
range	The time range.

Examples

```
Select ST_subTrajectorySpatial(traj, '2010-1-11 02:45:30', '2010-1-11 03:00:00') FROM traj_
table;
```

8.6.25. ST_samplingInterval

This function returns the sampling interval.

Syntax

```
interval ST_samplingInterval(trajectory traj);
```

Parameters

Parameter	Description
traj	The trajectory object.

Examples

```
select ST_samplingInterval(ST_MakeTrajectory('STPOINT'::leaftype, st_geomfromtext('LINESTRI
NG (114 35, 115 36, 116 37)', 4326), '[2010-01-01 14:30, 2010-01-01 15:30)'::tsrange, '{"le
afcount":3,"attributes":{"velocity": {"type": "integer", "length": 2,"nullable": true,"valu
e": [120, 130, 140]}, "accuracy": {"type": "float", "length": 4, "nullable": false,"value":
[120, 130, 140]}, "bearing": {"type": "float", "length": 8, "nullable": false,"value": [120
, 130, 140]}, "acceleration": {"type": "string", "length": 20, "nullable": true,"value": ["
120", "130", "140"]}, "active": {"type": "timestamp", "nullable": false,"value": ["Fri Jan
01 14:30:00 2010", "Fri Jan 01 15:00:00 2010", "Fri Jan 01 15:30:00 2010"]}}, "events": [{"
1":"Fri Jan 01 14:30:00 2010"}, {"2":"Fri Jan 01 15:00:00 2010"}, {"3":"Fri Jan 01 15:30:00
2010"}]}'));
 st_samplinginterval
---------------------
 @ 20 mins
(1 row)
```

8.6.26. ST_trajAttrsAsText

This function returns an array of values for a text-type attribute field of a trajectory object.

Syntax

```
text[] st_trajAttrsAsText(trajectory traj, text attr_name);
```

Parameters

Parameter	Description
traj	The trajectory object.

Parameter	Description
attr_name	The name of the attribute field.

Examples

```
With traj AS (
  select '{"trajectory":{"version":1,"type":"STPOINT","leafcount":2,"start_time":"2010-01-0
1 11:30:00","end_time":"2010-01-01 12:30:00","spatial":"SRID=4326;LINESTRING(1 1,3 5)","tim
eline":["2010-01-01 11:30:00","2010-01-01 12:30:00"],"attributes":{"leafcount":2,"velocity"
:{"type":"integer","length":4,"nullable":true,"value":[1,null]},"speed":{"type":"float","le
ngth":8,"nullable":true,"value":[null,1.0]},"angle":{"type":"string","length":64,"nullable"
:true,"value":["test",null]}, "tngel2":{"type":"timestamp","length":8,"nullable":true,"valu
e":["2010-01-01 12:30:00",null]},"bearing":{"type":"bool","length":1,"nullable":true,"value
":[null,true]}}}}'::trajectory a)
Select st_trajAttrsAsText(a, 'angle') from traj;
 st_trajattrsastext
--------------------
 {test,NULL}
(1 row)
With traj AS (
  select '{"trajectory":{"version":1,"type":"STPOINT","leafcount":2,"start_time":"2010-01-0
1 11:30:00","end_time":"2010-01-01 12:30:00","spatial":"SRID=4326;LINESTRING(1 1,3 5)","tim
eline":["2010-01-01 11:30:00","2010-01-01 12:30:00"],"attributes":{"leafcount":2,"velocity"
:{"type":"integer","length":4,"nullable":true,"value":[1,null]},"speed":{"type":"float","le
ngth":8,"nullable":true,"value":[null,1.0]},"angle":{"type":"string","length":64,"nullable"
:true,"value":["test",null]}, "tngel2":{"type":"timestamp","length":8,"nullable":true,"valu
e":["2010-01-01 12:30:00",null]},"bearing":{"type":"bool","length":1,"nullable":true,"value
":[null,true]}}}}'::trajectory a)
Select st_trajAttrsAsText(a, 'tngel2') from traj;
      st_trajattrsastext
------------------------------
 {"2010-01-01 12:30:00",NULL}
(1 row)
With traj AS (
  select '{"trajectory":{"version":1,"type":"STPOINT","leafcount":2,"start_time":"2010-01-0
1 11:30:00","end_time":"2010-01-01 12:30:00","spatial":"SRID=4326;LINESTRING(1 1,3 5)","tim
eline":["2010-01-01 11:30:00","2010-01-01 12:30:00"],"attributes":{"leafcount":2,"velocity"
:{"type":"integer","length":4,"nullable":true,"value":[1,null]},"speed":{"type":"float","le
ngth":8,"nullable":true,"value":[null,1.0]},"angle":{"type":"string","length":64,"nullable"
:true,"value":["test",null]}, "tngel2":{"type":"timestamp","length":8,"nullable":true,"valu
e":["2010-01-01 12:30:00",null]},"bearing":{"type":"bool","length":1,"nullable":true,"value
":[null,true]}}}}'::trajectory a)
Select st_trajAttrsAsText(a, 'bearing') from traj;
 st_trajattrsastext
--------------------
 {NULL,t}
(1 row)
```

8.6.27. ST_trajAttrsAsInteger

This function returns an array of values for an integer-type attribute field of a trajectory object.

Syntax

```
integer[] st_trajAttrsAsInteger(trajectory traj, text attr_name);
```

Parameters

Parameter	Description
traj	The trajectory object.
attr_name	The name of the attribute field.

Examples

```
With traj AS (
  select '{"trajectory":{"version":1,"type":"STPOINT","leafcount":2,"start_time":"2010-01-0
1 11:30:00","end_time":"2010-01-01 12:30:00","spatial":"SRID=4326;LINESTRING(1 1,3 5)","tim
eline":["2010-01-01 11:30:00","2010-01-01 12:30:00"],"attributes":{"leafcount":2,"velocity"
:{"type":"integer","length":4,"nullable":true,"value":[1,null]},"speed":{"type":"float","le
ngth":8,"nullable":true,"value":[null,1.0]},"angle":{"type":"string","length":64,"nullable"
:true,"value":["test",null]}, "tngel2":{"type":"timestamp","length":8,"nullable":true,"valu
e":["2010-01-01 12:30:00",null]},"bearing":{"type":"bool","length":1,"nullable":true,"value
":[null,true]}}}}'::trajectory a)
Select st_trajAttrsAsInteger(a, 'speed') from traj;
 st_trajattrsasinteger
-----------------------
 {NULL,1}
(1 row)
```

8.6.28. ST_trajAttrsAsDouble

This function returns an array of values for a double-type attribute field of a trajectory object.

Syntax

```
float8[] st_trajAttrsAsDouble(trajectory traj, text attr_name);
```

Parameters

Parameter	Description
traj	The trajectory object.
attr_name	The name of the attribute field.

Examples

```
With traj AS (
  select '{"trajectory":{"version":1,"type":"STPOINT","leafcount":2,"start_time":"2010-01-0
1 11:30:00","end_time":"2010-01-01 12:30:00","spatial":"SRID=4326;LINESTRING(1 1,3 5)","tim
eline":["2010-01-01 11:30:00","2010-01-01 12:30:00"],"attributes":{"leafcount":2,"velocity"
:{"type":"integer","length":4,"nullable":true,"value":[1,null]},"speed":{"type":"float","le
ngth":8,"nullable":true,"value":[null,1.0]},"angle":{"type":"string","length":64,"nullable"
:true,"value":["test",null]}, "tngel2":{"type":"timestamp","length":8,"nullable":true,"valu
e":["2010-01-01 12:30:00",null]},"bearing":{"type":"bool","length":1,"nullable":true,"value
":[null,true]}}}}'::trajectory a)
Select st_trajAttrsAsDouble(a, 'velocity') from traj;
 st_trajattrsasdouble
----------------------
 {1,NULL}
(1 row)
```

8.6.29. ST_trajAttrsAsBool

This function returns an array of values for a Boolean-type attribute field of a trajectory object.

Syntax

```
bool[] st_trajAttrsAsBool(trajectory traj, text attr_name);
```

Parameters

Parameter	Description
traj	The trajectory object.
attr_name	The name of the attribute field.

Examples

```
With traj AS (
  select '{"trajectory":{"version":1,"type":"STPOINT","leafcount":2,"start_time":"2010-01-0
1 11:30:00","end_time":"2010-01-01 12:30:00","spatial":"SRID=4326;LINESTRING(1 1,3 5)","tim
eline":["2010-01-01 11:30:00","2010-01-01 12:30:00"],"attributes":{"leafcount":2,"velocity"
:{"type":"integer","length":4,"nullable":true,"value":[1,null]},"speed":{"type":"float","le
ngth":8,"nullable":true,"value":[null,1.0]},"angle":{"type":"string","length":64,"nullable"
:true,"value":["test",null]}, "tngel2":{"type":"timestamp","length":8,"nullable":true,"valu
e":["2010-01-01 12:30:00",null]},"bearing":{"type":"bool","length":1,"nullable":true,"value
":[null,true]}}}}'::trajectory a)
Select st_trajAttrsAsBool(a, 'velocity') from traj;
 st_trajattrsasbool
--------------------
 {t,NULL}
(1 row)
```

8.6.30. ST_trajAttrsAsTimestamp

This function returns an array of values for a timestamp-type attribute field of a trajectory object.

Syntax

```
timestamp[] st_trajAttrsAsTimestamp(trajectory traj, text attr_name);
```

Parameters

Parameter	Description
traj	The trajectory object.
attr_name	The name of the attribute field.

Examples

```
With traj AS (
  select '{"trajectory":{"version":1,"type":"STPOINT","leafcount":2,"start_time":"2010-01-0
1 11:30:00","end_time":"2010-01-01 12:30:00","spatial":"SRID=4326;LINESTRING(1 1,3 5)","tim
eline":["2010-01-01 11:30:00","2010-01-01 12:30:00"],"attributes":{"leafcount":2,"velocity"
:{"type":"integer","length":4,"nullable":true,"value":[1,null]},"speed":{"type":"float","le
ngth":8,"nullable":true,"value":[null,1.0]},"angle":{"type":"string","length":64,"nullable"
:true,"value":["test",null]}, "tngel2":{"type":"timestamp","length":8,"nullable":true,"valu
e":["2010-01-01 12:30:00",null]},"bearing":{"type":"bool","length":1,"nullable":true,"value
":[null,true]}}}}'::trajectory a)
Select st_trajAttrsAsTimestamp(a, 'tngel2') from traj;
   st_trajattrsastimestamp
------------------------------
 {"2010-01-01 12:30:00",NULL}
```

8.6.31. ST_attrIntFilter

You can specify the trajectory attribute field. This function filters out the trajectory points that meet the condition based on a fixed value, and returns the filtered attribute field values (array).

Syntax

```
int8[] ST_attrIntFilter(trajectory traj, cstring attr_field_name,cstring operator, int8 val
ue);
int8[] ST_attrIntFilter(trajectory traj, cstring attr_field_name,cstring operator, int8 val
ue1, int8 value2);
```

Parameters

Parameter	Description
traj	The trajectory object attr
attr_field_name	The specified attribute name

Parameter	Description
operator	The filter operator: `'='` , `'!='` , `'>'` , `'<'` , `'>='` , `'<='` , `'[]'` , `'(]'` , `'[)'` , or `'()'`
value, value1	The minimum fixed value of the attribute
value2	The maximum fixed value of the attribute

Description

Only the attribute fields of the integer type are supported.

Examples

```
create table traj(id integer, traj trajectory);
insert into traj(traj) values(ST_makeTrajectory('STPOINT'::leaftype, 'LINESTRING(-179.48077
51.72814,-179.46731 51.74634,-179.46502 51.74934,-179.46183 51.75378,-179.45943 51.75736,-1
79.45560 51.76273,-179.44845 51.77186,-179.43419 51.78977,-179.41259 51.81643,-179.41001 51
.81941,-179.40751 51.82223,-179.40497 51.82505,-179.40242 51.82796,-179.39981 51.83095,-179
.39734 51.83398,-179.39499 51.83709)'::geometry, ARRAY['2017-01-15 09:06:39'::timestamp,'20
17-01-15 09:14:48','2017-01-15 09:13:39','2017-01-15 09:16:28','2017-01-15 09:19:48','2017-
01-15 09:17:48','2017-01-15 09:23:19','2017-01-15 09:34:40','2017-01-15 09:30:28','2017-01-
15 09:36:59','2017-01-15 09:38:09','2017-01-15 09:39:18','2017-01-15 09:40:40','2017-01-15
09:47:38','2017-01-15 21:18:30','2017-01-15 09:48:49'], '{"leafcount": 16, "attributes" : {
"heading" : {"type": "integer", "length": 4, "nullable" : false,"value":[0,1,2,3,4,5,6,7,8,
9,10,11,12,13,14,15]}}}'));
select st_attrIntFilter(traj, 'heading', '>', 5) from traj where id = 1;
      st_attrintfilter
-----------------------------
 {6,7,8,9,10,11,12,13,14,15}
(1 row)
select st_attrIntFilter(traj, 'heading', '>=', 5) from traj where id = 1;
       st_attrintfilter
------------------------------
 {5,6,7,8,9,10,11,12,13,14,15}
(1 row)
select st_attrIntFilter(traj, 'heading', '<', 5) from traj where id = 1;
 st_attrintfilter
------------------
 {0,1,2,3,4}
(1 row)
select st_attrIntFilter(traj, 'heading', '<=', 5) from traj where id = 1;
 st_attrintfilter
------------------
 {0,1,2,3,4,5}
(1 row)
select st_attrIntFilter(traj, 'heading', '=', 5) from traj where id = 1;
 st_attrintfilter
------------------
 {5}
(1 row)
select st_attrIntFilter(traj, 'heading', '!=', 5) from traj where id = 1;
           st_attrintfilter
```

```
----------------------------------------
 {0,1,2,3,4,6,7,8,9,10,11,12,13,14,15}
(1 row)
select st_attrIntFilter(traj, 'heading', '()', 5,8) from traj where id = 1;
 st_attrintfilter
------------------
 {6,7}
(1 row)
select st_attrIntFilter(traj, 'heading', '(]', 5,8) from traj where id = 1;
 st_attrintfilter
------------------
 {6,7,8}
(1 row)
select st_attrIntFilter(traj, 'heading', '[)', 5,8) from traj where id = 1;
 st_attrintfilter
------------------
 {5,6,7}
(1 row)
select st_attrIntFilter(traj, 'heading', '[]', 5,8) from traj where id = 1;
 st_attrintfilter
------------------
 {5,6,7,8}
(1 row)
```

8.6.32. ST_attrFloatFilter

You can specify the trajectory attribute field. This function filters out the trajectory points that meet the condition based on a fixed value, and returns the filtered attribute field values (array).

Syntax

```
float8[] ST_attrFloatFilter(trajectory traj, cstring attr_field_name,cstring operator, floa
t8 value);
float8[] ST_attrFloatFilter(trajectory traj, cstring attr_field_name,cstring operator, floa
t8 value1, float8 value2);
```

Parameters

Parameter	Description
traj	The trajectory object
attr_field_name	The specified attribute name
operator	The filter operator: `'='` , `'!='` , `'>'` , `'<'` , `'>='` , `'<='` , `'[]'` , `'(]'` , `'[)'` , and `'()'`
value, value1	The minimum fixed value of the attribute
value2	The maximum fixed value of the attribute

Description

Only the attribute fields of the float type are supported.

Examples

```
create table traj(id integer, traj trajectory);
insert into traj values(2,ST_makeTrajectory('STPOINT'::leaftype, 'LINESTRING(-179.48077 51.
72814,-179.46731 51.74634,-179.46502 51.74934,-179.46183 51.75378,-179.45943 51.75736,-179.
45560 51.76273,-179.44845 51.77186,-179.43419 51.78977,-179.41259 51.81643,-179.41001 51.81
941,-179.40751 51.82223,-179.40497 51.82505,-179.40242 51.82796,-179.39981 51.83095,-179.39
734 51.83398,-179.39499 51.83709)'::geometry, ARRAY['2017-01-15 09:06:39'::timestamp,'2017-
01-15 09:14:48','2017-01-15 09:13:39','2017-01-15 09:16:28','2017-01-15 09:19:48','2017-01-
15 09:17:48','2017-01-15 09:23:19','2017-01-15 09:34:40','2017-01-15 09:30:28','2017-01-15
09:36:59','2017-01-15 09:38:09','2017-01-15 09:39:18','2017-01-15 09:40:40','2017-01-15 09:
47:38','2017-01-15 21:18:30','2017-01-15 09:48:49'], '{"leafcount": 16, "attributes" : {"he
ading" : {"type": "float", "length": 8, "nullable" : false,"value":[23.0,23.0,23.0,23.0,21.
0,21.0,72.0,72.0,72.0,72.0,73.0,74.0,73.0,73.0,73.0,73.0]}}}'));
select st_attrFloatFilter(traj, 'heading', '>', 23.0) from traj where id = 2;
       st_attrfloatfilter
---------------------------------
 {72,72,72,72,73,74,73,73,73,73}
(1 row)
```

8.6.33. ST_attrTimestampFilter

This function filters all trajectory points to obtain the trajectory points whose values for the specified attribute field meet a filter condition based on a fixed value or a value range and returns an array of the attribute field values of these points.

Syntax

```
timestamp[] ST_attrTimestampFilter(trajectory traj, cstring attr_field_name,cstring operato
r, timestamp value);
timestamp[] ST_attrTimestampFilter(trajectory traj, cstring attr_field_name,cstring operato
r, timestamp value1, timestamp value2);
```

Parameters

Parameter	Description
traj	The trajectory object.
attr_field_name	The name of the specified attribute field.
operator	The filter operator. Valid values: `'='` , `'!='` , `'>'` , `'<'` , `'>='` , `'<='` , `'[]'` , `'(]'` , `'[)'` , `'()'` .
value and value1	The fixed value of the attribute field and the minimum fixed value of the attribute field.

Parameter	Description
value2	The maximum fixed value of the attribute field.

Examples

```
insert into traj values(3,ST_makeTrajectory('STPOINT'::leaftype, st_geomfromtext('LINESTRIN
G (114 35, 115 36, 116 37)', 4326), ARRAY['2010-01-01 14:30'::timestamp, '2010-01-01 15:00'
, '2010-01-01 15:30'], '{"leafcount": 3, "attributes": {"heading": {"type": "timestamp", "n
ullable": false,"value":["Fri Jan 01 14:30:00 2010", "Fri Jan 01 15:00:00 2010", "Fri Jan 0
1 15:30:00 2010"]}}}'));
select st_attrTimestampFilter(traj, 'heading', '>', 'Fri Jan 01 15:00:00 2010'::timestamp)
from traj where id = 3;
 st_attrtimestampfilter
------------------------
{"2010-01-01 15:30:00"}
(1 row)
```

8.6.34. ST_attrNullFilter

This function filters all trajectory points to obtain the trajectory points whose values are null for the specified attribute field and returns a new trajectory object that is composed of these points.

Syntax

```
trajectory ST_attrNullFilter(trajectory traj, cstring attr_field_name);
trajectory ST_attrNullFilter(trajectory traj, cstring attr_field_name);
```

Parameters

Parameter	Description
traj	The trajectory object.
attr_field_name	The name of the specified attribute field.

Description

This function supports all types of attribute fields.

Examples

```
select st_attrNullFilter(ST_makeTrajectory('STPOINT'::leaftype, 'LINESTRING(-179.48077 51.7
2814,-179.46731 51.74634,-179.46502 51.74934,-179.46183 51.75378,-179.45943 51.75736,-179.4
5560 51.76273,-179.44845 51.77186,-179.43419 51.78977,-179.41259 51.81643,-179.41001 51.819
41,-179.40751 51.82223,-179.40497 51.82505,-179.40242 51.82796,-179.39981 51.83095,-179.397
34 51.83398,-179.39499 51.83709)'::geometry, ARRAY['2017-01-15 09:06:39'::timestamp,'2017-0
1-15 09:13:39','2017-01-15 09:14:48','2017-01-15 09:16:28','2017-01-15 09:17:48','2017-01-1
5 09:19:48','2017-01-15 09:23:19','2017-01-15 09:30:28','2017-01-15 09:34:40','2017-01-15 0
9:36:59','2017-01-15 09:38:09','2017-01-15 09:39:18','2017-01-15 09:40:40','2017-01-15 09:4
7:38','2017-01-15 09:48:49','2017-01-15 21:18:30'], '{"leafcount": 16, "attributes": {"head
ing": {"type": "float", "length": 4, "nullable": true,"value": [23.0,23.0,23.0,null,21.0,21
.0,null,72.0,72.0,null,73.0,74.0,73.0,73.0,null,73.0]}}}'),'heading');
                              st_attrnullfilter

-----------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------
 {"trajectory":{"version":1,"type":"STPOINT","leafcount":4,"start_time":"2017-01-15 09:16:2
8","end_time":"2017-01-15 09:48:49","spatial":"LINESTRING(-179.46183 51.75378,-179.44845 51
.77
186,-179.41001 51.81941,-179.39734 51.83398)","timeline":["2017-01-15 09:16:28","2017-01-15
09:23:19","2017-01-15 09:36:59","2017-01-15 09:48:49"],"attributes":{"leafcount":4,"heading
":
{"type":"float","length":4,"nullable":true,"value":[null,null,null,null]}}}}
(1 row)
```

8.6.35. ST_attrNotNullFilter

This function filters all trajectory points to obtain the trajectory points whose values are not null for the specified attribute field and returns a new trajectory object that is composed of these points.

Syntax

```
trajectory ST_attrNotNullFilter(trajectory traj, cstring attr_field_name);
trajectory ST_attrNotNullFilter(trajectory traj, cstring attr_field_name);
```

Parameters

Parameter	Description
traj	The trajectory object.
attr_field_name	The name of the specified attribute field.

Description

This function supports all types of attribute fields.

Examples

```
select st_attrNotNullFilter(ST_makeTrajectory('STPOINT'::leaftype, 'LINESTRING(-179.48077 5
1.72814,-179.46731 51.74634,-179.46502 51.74934,-179.46183 51.75378,-179.45943 51.75736,-17
9.45560 51.76273,-179.44845 51.77186,-179.43419 51.78977,-179.41259 51.81643,-179.41001 51.
81941,-179.40751 51.82223,-179.40497 51.82505,-179.40242 51.82796,-179.39981 51.83095,-179.
39734 51.83398,-179.39499 51.83709)'::geometry, ARRAY['2017-01-15 09:06:39'::timestamp,'201
7-01-15 09:13:39','2017-01-15 09:14:48','2017-01-15 09:16:28','2017-01-15 09:17:48','2017-0
1-15 09:19:48','2017-01-15 09:23:19','2017-01-15 09:30:28','2017-01-15 09:34:40','2017-01-1
5 09:36:59','2017-01-15 09:38:09','2017-01-15 09:39:18','2017-01-15 09:40:40','2017-01-15 0
9:47:38','2017-01-15 09:48:49','2017-01-15 21:18:30'], '{"leafcount": 16, "attributes": {"h
eading": {"type": "float", "length": 4, "nullable": true,"value": [23.0,23.0,23.0,null,21.0
,21.0,null,72.0,72.0,null,73.0,74.0,73.0,73.0,null,73.0]}}}'),'heading');
                                st_attrnotnullfilter

-----------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------
 {"trajectory":{"version":1,"type":"STPOINT","leafcount":12,"start_time":"2017-01-15 09:06:
39","end_time":"2017-01-15 21:18:30","spatial":"LINESTRING(-179.48077 51.72814,-179.46731 5
1.7
4634,-179.46502 51.74934,-179.45943 51.75736,-179.4556 51.76273,-179.43419 51.78977,-179.41
259 51.81643,-179.40751 51.82223,-179.40497 51.82505,-179.40242 51.82796,-179.39981 51.8309
5,-
179.39499 51.83709)","timeline":["2017-01-15 09:06:39","2017-01-15 09:13:39","2017-01-15 09
:14:48","2017-01-15 09:17:48","2017-01-15 09:19:48","2017-01-15 09:30:28","2017-01-15 09:34
:40
","2017-01-15 09:38:09","2017-01-15 09:39:18","2017-01-15 09:40:40","2017-01-15 09:47:38","
2017-01-15 21:18:30"],"attributes":{"leafcount":12,"heading":{"type":"float","length":4,"nu
lla
ble":true,"value":[23.0,23.0,23.0,21.0,21.0,72.0,72.0,73.0,74.0,73.0,73.0,73.0]}}}}
(1 row)
```

8.6.36. ST_trajAttrsMeanMax

This function returns the maximum average value of an attribute field by using the MEAN-MAX algorithm.

Syntax

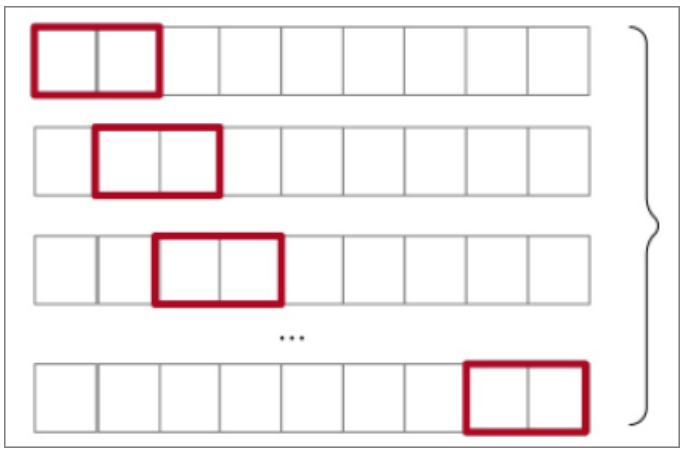
```
SETOF record ST_trajAttrsMeanMax(trajectory traj, cstring attr_field_name, out interval dur
ation, out float8 max);
```

Parameters

Parameter	Description
traj	The trajectory object.
attr_field_name	The name of the specified attribute field.

Description

The MEAN-MAX algorithm uses a sliding window to compute the average value of the specified attribute field in each time range specified by the window, and then returns the maximum value of all the average values.



This function supports only attribute fields of the integer and float types. The values of the specified attribute field cannot be null.

Examples

```
With traj AS (
    Select ST_makeTrajectory('STPOINT', 'LINESTRING(1 1, 6 6, 9 8, 10 12)'::geometry,
    ARRAY['2010-01-01 11:30'::timestamp, '2010-01-01 12:30', '2010-01-01 13:30', '2010-01-0
1 14:30'],
    '{"leafcount":4, "attributes":{"velocity": {"type": "float", "length": 8,"nullable": tr
ue,"value": [120.0, 130.0, 140.0, 120.0]}, "power": {"type": "float", "length": 4,"nullable
": true,"value": [120.0, 130.0, 140.0, 120.0]}}}') a)
Select st_trajAttrsMeanMax(a, 'velocity') from traj;
 st_trajattrsmeanmax
---------------------
 ("@ 1 hour",135)
 ("@ 2 hours",130)
 ("@ 3 hours",127.5)
(3 rows)
 With traj AS (
    Select ST_makeTrajectory('STPOINT', 'LINESTRING(1 1, 6 6, 9 8, 10 12)'::geometry,
    ARRAY['2010-01-01 11:30'::timestamp, '2010-01-01 12:30', '2010-01-01 13:30', '2010-01-0
1 14:30'],
    '{"leafcount":4, "attributes":{"velocity": {"type": "float", "length": 8,"nullable": tr
ue,"value": [120.0, 130.0, 140.0, 120.0]}, "power": {"type": "float", "length": 4,"nullable
": true,"value": [120.0, 130.0, 140.0, 120.0]}}}') a)
Select (st_trajAttrsMeanMax(a, 'velocity')).* from traj;
 duration |  max
----------+-------
 01:00:00 |   135
 02:00:00 |   130
 03:00:00 | 127.5
(3 rows)
```

8.7. Functions to process bounding boxes

8.7.1. ST_MakeBox

This function is used to query the bounding box of an object over a specified time range.

Syntax

```
boxndf ST_MakeBox(geometry geom);
boxndf ST_MakeBox(trajectory traj);
boxndf ST_MakeBox(geometry geom, timestamp ts, timestamp te);
boxndf ST_MakeBox(trajectory traj, timestamp ts, timestamp te);
boxndf ST_MakeBox(timestamp ts, timestamp te);
```

Parameters

Parameter	Description
geom	The geometry whose bounding box you want to query.
traj	The trajectory whose bounding box you want to query.
ts	The beginning of the time range to query.
te	The end of the time range to query.

Description

A bounding box is the minimal multi-dimensional rectangle that contains a spatio-temporal object.

- If you specify a two- or three-dimensional spatial object, this function returns the bounding box of the object based on the dimensions of the object.
- If you specify a spatial object and a time range by using the `ST_MakeBox(geometry geom, timestamp ts, timestamp te)` syntax, this function returns the bounding box of the object over the time range.
- If you specify a trajectory but do not specify a time range, this function returns the bounding box of the complete trajectory. If you specify a trajectory and a time range, this function returns the bounding box of the sub-trajectory over the time range.
- Bounding boxes use the FLOAT data type. Therefore, this function may return a bounding box that is slightly larger than specified by the parameter settings. For example, the returned lower bound is slightly smaller than the actual lower bound, or the returned upper bound is slightly larger than the actual upper bound.

Example

```
WITH geom AS(
    SELECT ('POLYGON((12.7243236691148 4.35238368367118,12.9102992732078 1.49748113937676,1
2.5926592946053 1.67643963359296' ||
            ',12.0197574747333 3.19258554889152,12.7243236691148 4.35238368367118))')::geom
etry a
)
SELECT ST_MakeBox(a),ST_MakeBox(a,'2000-01-01 00:00:10'::timestamp, '2000-01-01 02:13:20'::
timestamp) from geom;
                           st_makebox                           |
st_makebox
----------------------------------------------------------------+-------------------------
-----------------------------------------------------------------------------------------
--
 BOX2D(12.0197572708 1.49748110771,12.9102993011 4.35238409042) | BOX2DT(12.0197572708 1.49
748110771 2000-01-01 00:00:09.999999,12.9102993011 4.35238409042 2000-01-01 02:13:20.000381
)
With traj AS (
    Select ST_makeTrajectory('STPOINT', 'LINESTRING(0 0, 50 50, 100 100)'::geometry,
                             tsrange('2000-01-01 00:00:00'::timestamp, '2000-01-01 00:01:40
'::timestamp),
                             '{"leafcount":3,"attributes":{"velocity": {"type": "integer",
"length": 2,"nullable" : true,"value": [120,130,140]}, "accuracy": {"type": "float", "lengt
h": 4, "nullable" : false,"value": [120,130,140]}, "bearing": {"type": "float", "length": 8
, "nullable" : false,"value": [120,130,140]}, "acceleration": {"type": "string", "length":
20, "nullable" : true,"value": ["120","130","140"]}, "active": {"type": "timestamp", "nulla
ble" : false,"value": ["Fri Jan 01 11:35:00 2010", "Fri Jan 01 12:35:00 2010", "Fri Jan 01
13:30:00 2010"]}}, "events": [{"2" : "Fri Jan 02 15:00:00 2010"}, {"3" : "Fri Jan 02 15:30:
00 2010"}]}') a
)
SELECT  ST_MakeBox(a), ST_MakeBox(a,'1999-12-31 23:00:00'::timestamp, '2000-01-01 00:00:30'
::timestamp) from traj;
                          st_makebox                           |                            s
t_makebox
----------------------------------------------------------------+----------------------------
-------------------------------------
 BOX2DT(0 0 2000-01-01 00:00:00,100 100 2000-01-01 00:01:40) | BOX2DT(0 0 2000-01-01 00:00:
00,30 30 2000-01-01 00:00:30.000001)
```

8.7.2. ST_MakeBox{Z|T|2D|2DT|3D|3DT}

This function is used to build a bounding box based on specified parameter settings.

Syntax

```
boxndf ST_MakeBoxZ(float8 zmin, float8 zmax);
boxndf ST_MakeBoxT(timestamp tmin, timestamp tmax);
boxndf ST_MakeBox2D(float8 xmin, float8 ymin, float8 xmax, float8 ymax);
boxndf ST_MakeBox2DT(float8 xmin, float8 ymin, timestamp tmin, float8 xmax,float8 yax, time
stamp tmax);
boxndf ST_MakeBox3D(float8 xmin, float8 ymin, float8 zmin, float8 xmax, float8 ymax, float8
zmax);
boxndf ST_MakeBox3DT(float8 xmin, float8 ymin, float8 zmin, timestamp tmin, float8 xmax, fl
oat8 ymax, float8 zmax, timestamp tmax);
```

Parameters

Parameter	Description
xmin	The lower bound on the x axis of the bounding box.
xmax	The upper bound on the x axis of the bounding box.
ymin	The lower bound on the y axis of the bounding box.
ymax	The upper bound on the y axis of the bounding box.
zmin	The lower bound on the z axis of the bounding box.
zmax	The upper bound on the z axis of the bounding box.
tmin	The lower bound on the t axis of the bounding box.
tmax	The upper bound on the t axis of the bounding box.

Description

This function allows you to build a bounding box based on the function name and the specified parameter settings.

Bounding boxes use the FLOAT data type. Therefore, this function may return a bounding box that is slightly larger than specified by the parameter settings. For example, the returned lower bound is slightly smaller than the actual lower bound, or the returned upper bound is slightly larger than the actual upper bound.

Example

```
SELECT ST_MakeBox2d(0,0,3,3);
  st_makebox2d
----------------
 BOX2D(0 0,3 3)
 SELECT ST_MakeBox3dt(0,0,3,'2000-01-01 00:00:03'::timestamp, 2,5,4,'2000-01-01 02:46:40'::
timestamp);
                                st_makebox3dt
------------------------------------------------------------------------------
 BOX3DT(0 0 3 2000-01-01 00:00:02.999999,2 5 4 2000-01-01 02:46:40.000476)
(1 row)
```

8.7.3. ST_BoxndfToGeom

This function is used to convert a bounding box into a geometry.

Syntax

```
geometry ST_BoxNDFToGeom(boxndf box);
```

Parameters

Parameter	Description
box	The bounding box that you want to convert into a geometry.

Description

This function allows you to convert a bounding box into a geometry.

- If the bounding box has the z dimension, this function converts the bounding box into a three-dimensional geometry.
- If the bounding box does not have the z dimension, this function converts the bounding box into a two-dimensional geometry.
- If the bounding box does not have the x or y dimension, this function returns NULL.

Example

```
select ST_AsText(ST_BoxndfToGeom(ST_MakeBox2dt(0,0,'2000-01-01'::timestamp, 20,20, '2020-01
-01'::timestamp)));
             st_astext
-----------------------------------
 POLYGON((0 0,0 20,20 20,20 0,0 0))
```

8.7.4. ST_Has{xy|z|t}

The ST_Has{xy|z|t} function is used to check whether a bounding box has one or more specified dimensions.

Syntax

```
bool ST_HasXY(boxndf box);
bool ST_HasZ(boxndf box);
bool ST_HasT(boxndf box);
```

Parameters

Parameter	Description
box	The bounding box that you want to query.

Description

This function allows you to check whether a bounding box has one or more specified dimensions. The x and y dimensions are bound to each other. Therefore, the bounding box has both or neither of the x and y dimensions.

Example

```
postgres=# select ST_hasz(ST_MakeBox2dt(0,0,'2000-01-01'::timestamp, 20,20, '2020-01-01'::t
imestamp));
 st_hasz
---------
 f
postgres=# select ST_hast(ST_MakeBox2dt(0,0,'2000-01-01'::timestamp, 20,20, '2020-01-01'::t
imestamp));
 st_hast
---------
 t
```

8.7.5. ST_{X|Y|Z|T}Min

The function is used to query the minimum value of a bounding box in a specified dimension.

Syntax

```
float8 ST_XMin(boxndf box);
float8 ST_YMin(boxndf box);
float8 ST_ZMin(boxndf box);
timestamp ST_TMin(boxndf box);
```

Parameters

Parameter	Description
box	The bounding box whose minimum value in a specified dimension you want to query.

Description

This function allows you to query the minimum value of a bounding box in a specified dimension.

- If the bounding box does not have the specified x, y, or z dimension, this function returns `NaN`.
- If the bounding box does not have the specified t dimension, this function returns `-infinity`.

Example

```
select ST_xmin(ST_MakeBox2dt(0,0,'2000-01-01'::timestamp, 20,20, '2020-01-01'::timestamp));
 st_xmin
---------
       0
select ST_zmax(ST_MakeBox2dt(0,0,'2000-01-01'::timestamp, 20,20, '2020-01-01'::timestamp));
 st_zmin
---------
     NaN
select ST_tmax(ST_MakeBox2d(0,0, 20,20));
  st_tmin
-----------
 -infinity
```

8.7.6. ST_{X|Y|Z|T}Max

This function is used to query the maximum value of a bounding box in a specified dimension.

Syntax

```
float8 ST_XMax(boxndf box);
float8 ST_YMax(boxndf box);
float8 ST_ZMax(boxndf box);
timestamp ST_TMax(boxndf box);
```

Parameters

Parameter	Description
box	The bounding box whose maximum value in a specified dimension you want to query.

Description

This function allows you to query the maximum value of a bounding box in a specified dimension.

- If the bounding box does not have the specified x, y, or z dimension, this function returns `NaN`.
- If the bounding box does not have the specified t dimension, this function returns `-infinity`.

Example

```
select ST_tmax(ST_MakeBox2dt(0,0,'2000-01-01'::timestamp, 20,20, '2020-01-01'::timestamp));
       st_tmax
---------------------
 2020-01-01 00:00:00
select ST_zmin(ST_MakeBox2dt(0,0,'2000-01-01'::timestamp, 20,20, '2020-01-01'::timestamp));
 st_zmin
---------
     NaN
select ST_tmin(ST_MakeBox2d(0,0, 20,20));
  st_tmin
-----------
 -infinity
```

8.7.7. ST_ExpandSpatial

This function is used to expand a bounding box to a specified spatial scope.

Syntax

```
boxndf ST_ExpandSpatial(boxndf box, float8 len);
```

Parameters

Parameter	Description
box	The bounding box that you want to expand.
len	The length to which you want to expand the bounding box.

Description

This function allows you to expand a bounding box to a specified spatial scope. If the bounding box has the x, y, and z dimensions, the lower bound is decreased by the value of the len parameter, and the upper bound is increased by the value of the len parameter.

Example

```
Select ST_ExpandSpatial(ST_MakeBox2dt(0,0,'2000-01-02', 20,20, '2000-03-03'), 4);
                      st_expandspatial
-------------------------------------------------------------
 BOX2DT(-4 -4 2000-01-02 00:00:00,24 24 2000-03-03 00:00:00)
```

8.8. Operators to process bounding boxes

8.8.1. Basic concepts

A bounding box-processing operator is used to check whether the bounding boxes of the two objects that are specified by the left and right operands meet a specified spatio-temporal relationship.

Operator marks

The following operator marks are supported:

- The mark for INTERSECT operators: `&& or &`
- The mark for INCLUDE operators: `@>`
- The mark for INCLUDED operators: `<@`

> **Note**
> - If a dimension mark resides between a pair of operator marks, the operator compares two objects in this dimension in addition to the x and y dimensions.
> - If a dimension mark resides on the left and right sides of an operator mark, the operator compares two objects only in this dimension. In this case, the operator does not compare the two objects in the other dimensions, such as the x and y dimensions.

Dimension marks

The following dimension marks are supported:

- The mark for the z dimension: /
- The mark for the t dimension: #

> **Note**
> - If an operator does not contain a dimension mark, the operator specifies a two-dimensional relationship.
> - If an operator contains both dimension marks, the mark for the z dimension precedes the mark for the t dimension.

8.8.2. INTERSECT operators

This topic describes the INTERSECT operators. These operators are used to check whether the bounding box of the left operand-specified object intersects with the bounding box of the right operand-specified object in a specified dimension.

Syntax

```
{geometry, trajectory, boxndf} /&/ {geometry, trajectory, boxndf}
{trajectory, boxndf} #&# {trajectory, boxndf}
{geometry, trajectory, boxndf} && {geometry, trajectory, boxndf}
{geometry, trajectory, boxndf} &/& {geometry, trajectory, boxndf}
{trajectory, boxndf} &#& {trajectory, boxndf}
{trajectory, boxndf} &/#& {trajectory, boxndf}
```

Parameters

Parameter	Description
Left operand	The object whose bounding box you want to compare.
Right operand	The object whose bounding box you want to compare.

Description

The INTERSECT operators allow you to check whether the bounding box of the left operand-specified object intersects with the bounding box of the right operand-specified object in a specified dimension. These bounding boxes are generated by the ST_MakeBox function. For more information, see ST_MakeBox.

The following INTERSECT operators are supported:

- `/&/` : checks whether the bounding boxes of two objects intersect in the z dimension.
- `#&#` : checks whether the bounding boxes of two objects intersect in the t dimension.
- `&&` : checks whether the bounding boxes of two objects intersect in the x and y dimensions.
- `&/&` : checks whether the bounding boxes of two objects intersect in the x, y, and z dimensions.
- `&#&` : checks whether the bounding boxes of two objects intersect in the x, y, and t dimensions.
- `&/#&` : checks whether the bounding boxes of two objects intersect in the x, y, z, t, and t dimensions.

Example

```
WITH box AS(
    SELECT ST_MakeBox3d(0,0,0,5,5,5) a,
           ST_MakeBox3dt(6,6, 3,'2010-04-12 00:00:00',8,8,5,'2013-02-01 00:00:00') b
)
SELECT a /&/ b AS OpZ, a #&# b AS OpT, a && b AS Op2D, a &/& b AS Op3d, a &#& b AS Op2DT, a
&/#& b AS Op3DT from box;
 opz | opt | op2d | op3d | op2dt | op3dt
-----+-----+------+------+-------+-------
 t   | t   | f    | f    | f     | f
```

8.8.3. INCLUDE operators

This topic describes the INCLUDE operators. These operators are used to check whether the bounding box of the left operand-specified object includes the bounding box of the right operand-specified object in a specified dimension.

Syntax

```
{geometry, trajectory, boxndf} /@>/ {geometry, trajectory, boxndf}
{trajectory, boxndf} #@># {trajectory, boxndf}
{geometry, trajectory, boxndf} @> {geometry, trajectory, boxndf}
{geometry, trajectory, boxndf} @/> {geometry, trajectory, boxndf}
{trajectory, boxndf} @#> {trajectory, boxndf}
{trajectory, boxndf} @/#> {trajectory, boxndf}
```

Parameters

Parameter	Description
Left operand	The object whose bounding box you want to compare.
Right operand	The object whose bounding box you want to compare.

Description

The INCLUDE operators allow you to check whether the bounding box of the left operand-specified object includes the bounding box of the right operand-specified object in a specified dimension. These bounding boxes are generated by the ST_MakeBox function. For more information, see ST_MakeBox. The effect of the INCLUDE operators is opposite to that of the INCLUDED operators. For more information, see INCLUDED operators.

The following INCLUDE operators are supported:

- `/@>/` : checks whether the bounding box of the left operand-specified object includes the bounding box of the right operand-specified object in the z dimension.
- `#@>#` : checks whether the bounding box of the left operand-specified object includes the bounding box of the right operand-specified object in the t dimension.
- `@>` : checks whether the bounding box of the left operand-specified object includes the bounding box of the right operand-specified object in the x and y dimensions.
- `@&>` : checks whether the bounding box of the left operand-specified object includes the bounding box of the right operand-specified object in the x, y, and z dimensions.
- `<#@` : checks whether the bounding box of the left operand-specified object is included in the bounding box of the right operand-specified object in the x, y, and t dimensions.
- `@/#>` : checks whether the bounding box of the left operand-specified object is included in the bounding box of the right operand-specified object in the x, y, z, and t dimensions.

Example

```
WITH box AS(
    SELECT ST_MakeBox3dt(0,0,0, '2010-01-01 00:00:00',10,10,10, '2012-01-01 00:00:00') a,
           ST_MakeBox3dt(6,6,3,'2010-01-01 00:00:00',8,8,5,'2013-01-01 00:00:00') b
)
SELECT a /@>/ b AS OpZ, a #@># b AS OpT, a @> b AS Op2D, a @/> b AS Op3D, a @#> b AS Op2DT,
a @/#> b AS Op3DT from box;
 opz | opt | op2d | op3d | op2dt | op3dt
-----+-----+------+------+-------+-------
 t   | f   | t    | t    | f     | f
```

8.8.4. INCLUDED operators

This topic describes the INCLUDED operators. These operators are used to check whether the bounding box of the left operand-specified object is included in the bounding box of the right operand-specified object in a specified dimension.

Syntax

```
{geometry, trajectory, boxndf} /<@/ {geometry, trajectory, boxndf}
{trajectory, boxndf} #<@# {trajectory, boxndf}
{geometry, trajectory, boxndf} <@ {geometry, trajectory, boxndf}
{geometry, trajectory, boxndf} </@ {geometry, trajectory, boxndf}
{trajectory, boxndf} <#@ {trajectory, boxndf}
{trajectory, boxndf} </#@ {trajectory, boxndf}
```

Parameters

Parameter	Description
Left operand	The object whose bounding box you want to compare.
Right operand	The object whose bounding box you want to compare.

Description

The INCLUDED operators allow you to check whether the bounding box of the left operand-specified object is included in the bounding box of the right operand-specified object in a specified dimension. These bounding boxes are generated by the ST_MakeBox function. For more information, see ST_MakeBox. The effect of the INCLUDED operators is opposite to that of the INCLUDE operators. For more information, see INCLUDE operators.

The following INCLUDED operators are supported:

- `/<@/` : checks whether the bounding box of the left operand-specified object is included in the bounding box of the right operand-specified object in the z dimension.
- `#<@#` : checks whether the bounding box of the left operand-specified object is included in the bounding box of the right operand-specified object in the t dimension.
- `<@` : checks whether the bounding box of the left operand-specified object is included in the bounding box of the right operand-specified object in the x and y dimensions.

- `<&@` : checks whether the bounding box of the left operand-specified object is included in the bounding box of the right operand-specified object in the x, y, and z dimensions.
- `<#@` : checks whether the bounding box of the left operand-specified object is included in the bounding box of the right operand-specified object in the x, y, and t dimensions.
- `</#@` : checks whether the bounding box of the left operand-specified object is included in the bounding box of the right operand-specified object in the x, y, z, and t dimensions.

Example

```
WITH box AS(
    SELECT ST_MakeBox3dt(0,0,0, '2010-01-01 00:00:00',10,10,10, '2012-01-01 00:00:00') a,
           ST_MakeBox3dt(6,6,3,'2010-01-01 00:00:00',8,8,5,'2013-01-01 00:00:00') b
)
SELECT b /<@/ a AS OpZ, b #<@# a AS OpT, b <@ a AS Op2D, b </@ a AS Op3D, b <#@ a AS Op2DT,
b </#@ a AS Op3DT from box;
 opz | opt | op2d | op3d | op2dt | op3dt
-----+-----+------+------+-------+-------
 t   | f   | t    | t    | f     | f
```

8.9. Spatial relationship judgment

8.9.1. ST_intersects

This function returns true if a trajectory object and the specified geometry object intersect in space within the specified time range.

Syntax

```
boolean ST_intersects(trajectory traj, tsrange range, geometry g);
boolean ST_intersects(trajectory traj, timestamp t1, timestamp t2, geometry g);
```

Parameters

Parameter	Description
traj	The trajectory object.
t1	The start time.
t2	The end time.
range	The time range.
g	The specified geometry object.

Example

```
Select ST_intersects(traj, '2010-1-1 13:00:00', '2010-1-1 14:00:00', 'LINESTRING(0 0, 5 5,
9 9)'::geometry) from traj_table;
```

8.9.2. ST_equals

This function returns true if a trajectory object and the specified geometry object are spatially equal within the specified time range.

Syntax

```
boolean ST_equals(trajectory traj, tsrange range, geometry g);
boolean ST_equals(trajectory traj, timestamp t1, timestamp t2, geometry g);
```

Parameters

Parameter	Description
traj	The trajectory object.
t1	The start time.
t2	The end time.
range	The time range.
g	The specified geometry object.

Example

```
Select ST_equals(traj, '2010-1-1 13:00:00', '2010-1-1 14:00:00', 'LINESTRING(0 0, 5 5, 9 9)
'::geometry) from traj_table;
```

8.9.3. ST_distanceWithin

This function returns true if the distance between a trajectory object within the specified time range and the specified geometry object is within the reference distance.

Syntax

```
boolean ST_distanceWithin(trajectory traj, tsrange range, geometry g, float8 d);
boolean ST_distanceWithin(trajectory traj, timestamp t1, timestamp t2, geometry g, float8 d
);
```

Parameters

Parameter	Description
traj	The trajectory object.
t1	The start time.
t2	The end time.

Parameter	Description
range	The time range.
g	The specified geometry object.
d	The reference distance.

Example

```
Select ST_distanceWithin(traj, '2010-1-1 13:00:00', '2010-1-1 14:00:00', 'LINESTRING(0 0, 5
5, 9 9)'::geometry, 10) from traj_table;
```

8.10. Spatial processing

8.10.1. ST_intersection

This function returns a new trajectory object that indicates the intersection of a trajectory object within the specified time range and the specified geometry object.

Syntax

```
trajectory[] ST_intersection(trajectory traj, tsrange range, geometry g);
trajectory[] ST_intersection(trajectory traj, timestamp t1, timestamp t2, geometry g);
```

Parameters

Parameter	Description
traj	The trajectory object.
t1	The start time.
t2	The end time.
range	The time range.
g	The specified geometry object.

Description

If the trajectory object and the geometry object intersect at multiple spatial points, this function returns multiple sub-trajectories.

Example

```
Select ST_intersection(traj, '2010-1-1 13:00:00', '2010-1-1 14:00:00', 'LINESTRING(0 0, 5 5
, 9 9)'::geometry) from traj_table;
```

8.10.2. ST_difference

This function returns a new trajectory object that indicates the difference between a trajectory object within the specified time range and the specified geometry object.

Syntax

```
trajectory ST_difference(trajectory traj, tsrange range, geometry g);
trajectory ST_difference(trajectory traj, timestamp t1, timestamp t2, geometry g);
```

Parameters

Parameter	Description
traj	The trajectory object.
t1	The start time.
t2	The end time.
range	The time range.
g	The specified geometry object.

Example

```
Select ST_difference(traj, '2010-1-1 13:00:00', '2010-1-1 14:00:00', 'LINESTRING(0 0, 5 5,
9 9)'::geometry) from traj_table;
```

8.11. Spatial statistics

8.11.1. ST_nearestApproachPoint

This function returns the spatial point in a trajectory object that is nearest to the specified geometry object within the specified time range.

Syntax

```
geometry ST_nearestApproachPoint(trajectory traj, tsrange range, geometry g);
geometry ST_nearestApproachPoint(trajectory traj, timestamp t1, timestamp t2, geometry g);
```

Parameters

Parameter	Description
traj	The trajectory object.
t1	The start time.

Parameter	Description
t2	The end time.
range	The time range.
g	The specified geometry object.

Example

```
Select ST_nearestApproachPoint(traj, '2010-1-1 13:00:00', '2010-1-1 14:00:00', 'LINESTRING(
0 0, 5 5, 9 9)'::geometry) from traj_table;
```

8.11.2. ST_nearestApproachDistance

This function returns the nearest distance between a trajectory object and the specified geometry object within the specified time range.

Syntax

```
float8 ST_nearestApproachDistance(trajectory traj, tsrange range, geometry g);
float8 ST_nearestApproachDistance(trajectory traj, timestamp t1, timestamp t2, geometry g);
```

Parameters

Parameter	Description
traj	The trajectory object.
t1	The start time.
t2	The end time.
range	The time range.
g	The specified geometry object.

Example

```
Select ST_nearestApproachDistance(traj, '2010-1-1 13:00:00', '2010-1-1 14:00:00', 'LINESTRI
NG(0 0, 5 5, 9 9)'::geometry) from traj_table;
```

8.12. Spatio-temporal relationship judgment

8.12.1. ST_intersects

This function returns true if trajectory objects 1 and 2 intersect in space within the specified time range.

Syntax

```
boolean ST_intersects(trajectory traj1, trajectory traj2);
boolean ST_intersects(trajectory traj1, trajectory traj2, tsrange range);
boolean ST_intersects(trajectory traj1, trajectory traj2, timestamp t1, timestamp t2);
```

Parameters

Parameter	Description
traj	The trajectory object.
t1	The start time.
t2	The end time.
range	The time range.

Examples

```
Select ST_intersects((Select traj from traj_table where id=1), (Select traj from traj_table
where id=2), '2010-1-1 13:00:00', '2010-1-1 14:00:00');
```

8.12.2. ST_equals

This function returns true if trajectory objects 1 and 2 are spatially equal within the specified time range.

Syntax

```
boolean ST_equals(trajectory traj1, trajectory traj2, tsrange range);
boolean ST_equals(trajectory traj1, trajectory traj2, timestamp t1, timestamp t2);
```

Parameters

Parameter	Description
traj	The trajectory object.
t1	The start time.
t2	The end time.
range	The time range.

Examples

```
Select ST_equals((Select traj from traj_table where id=1), (Select traj from traj_table where id=2), '2010-1-1 13:00:00', '2010-1-1 14:00:00');
```

8.12.3. ST_distanceWithin

This function returns true if the distance between trajectory objects 1 and 2 within the specified time range is within the reference distance.

Syntax

```
boolean ST_distanceWithin(trajectory traj1, trajectory traj2, tsrange range, float8 d);
boolean ST_distanceWithin(trajectory traj1, trajectory traj2, timestamp t1, timestamp t2, float8 d);
```

Parameters

Parameter	Description
traj	The trajectory object.
t1	The start time.
t2	The end time.
range	The time range.
d	The reference distance.

Examples

```
Select ST_distanceWithin((Select traj from traj_table where id=1), (Select traj from traj_table where id=2), '2010-1-1 13:00:00', '2010-1-1 14:00:00', 100);
```

8.12.4. ST_durationWithin

This function returns true if the difference between the time when trajectory objects 1 and 2 intersect at the same spatial point within the specified time range is within the reference interval.

Syntax

```
boolean ST_durationWithin(trajectory traj1, trajectory traj2, tsrange range, interval i);
boolean ST_durationWithin(trajectory traj1, trajectory traj2, timestamp t1, timestamp t2, interval i);
```

Parameters

Parameter	Description
traj	The trajectory object.
t1	The start time.
t2	The end time.
range	The time range.
i	The reference interval.

Description

If two trajectory objects intersect at the same spatial point multiple times, this function returns true so long as any time difference is within the reference interval.

Examples

```
Select ST_durationWithin((Select traj from traj_table where id=1), (Select traj from traj_t
able where id=2), '2010-1-1 13:00:00', '2010-1-1 14:00:00', INTERVAL '30s');
```

8.12.5. ST_{Z|T|2D|2DT|3D|3DT}Intersects

This function is used to check whether two objects intersect on a specified axis.

Syntax

```
bool ST_{2D|3D}Intersects(geometry geom, trajectory traj);
bool ST_{2D|3D}Intersects(trajectory traj, geometry geom);
bool ST_{2D|3D}Intersects(geometry geom, trajectory traj, timestamp ts, timestamp te);
bool ST_{2D|3D}Intersects(trajectory traj, geometry geom, timestamp ts, timestamp te);
bool ST_{Z|T|2D|2DT|3D|3DT}Intersects(boxndf box, trajectory traj);
bool ST_{Z|T|2D|2DT|3D|3DT}Intersects(trajectory traj, boxndf box);
bool ST_{Z|T|2D|2DT|3D|3DT}Intersects(boxndf box, trajectory traj, timestamp ts, timestamp
te);
bool ST_{Z|T|2D|2DT|3D|3DT}Intersects(trajectory traj, boxndf box, timestamp ts, timestamp
te);
bool ST_{2D|2DT|3D|3DT}Intersects(trajectory traj1, trajectory traj2);
bool ST_{2D|2DT|3D|3DT}Intersects(trajectory traj1, trajectory traj2, timestamp ts, timesta
mp te);
```

Parameters

Parameter	Description
geom	The geometry that you want to compare.
traj	The trajectory that you want to compare, or the original trajectory that includes the sub-trajectory that you want to compare.

Parameter	Description
traj1	The trajectory that you want to compare, or the original trajectory that includes the sub-trajectory that you want to compare.
traj2	The trajectory that you want to compare, or the original trajectory that includes the sub-trajectory that you want to compare.
box	The bounding box that you want to compare.
ts	The beginning of the time range over which you want to extract sub-trajectories. This parameter is optional.
te	The end of the time range over which you want to extract sub-trajectories. This parameter is optional.

Description

This function allows you to check whether two objects intersect in a specified dimension. If a dimension does not have a specified value, this function considers this dimension to have any value.

If this function contains the ts and te parameters, it compares two objects to check whether the sub-trajectories of the two objects over the specified time range intersect. If this function does not contain the ts or te parameter, it compares two objects to check whether the complete trajectories of the two objects intersect. The two objects are specified by the traj or traj1 parameter and the traj2 parameter.

The `ST_Intersects(...)` function delivers the same effect as the `ST_2DIntersects(...)` function (if one of the specified objects is a geometry) or the `ST_3DTIntersects(...)` function (if both the specified objects are trajectories).

Example

```
WITH traj AS(
    SELECT (' {"trajectory":{"version":1,"type":"STPOINT","leafcount":6,"start_time":"2000-
01-01 03:15:42","end_time":"2000-01-01 05:16:43",' ||
            '"spatial":"LINESTRING(2 2 0,33.042158099636 36.832684322819 0,47.244002354518
47.230026333034 0,64.978971942887 60.618813472986 0,77.621717839502 78.012496630661 0,80 78
0)",' ||
            '"timeline":["2000-01-01 03:15:42","2000-01-01 03:39:54","2000-01-01 04:04:06",
"2000-01-01 04:28:18","2000-01-01 04:52:31","2000-01-01 05:16:43"]}}')::trajectory a,
           ('{"trajectory":{"version":1,"type":"STPOINT","leafcount":4,"start_time":"2000-0
1-01 02:17:58.656079","end_time":"2000-01-01 03:43:59.620923",' ||
            '"spatial":"LINESTRING(40 2 0,15.17549143297 51.766017656152 0,1.444002354518 6
9.630026333034 0,3 70 0)","timeline":["2000-01-01 02:17:58.656079",' ||
            '"2000-01-01 02:46:38.977693","2000-01-01 03:15:19.299307","2000-01-01 03:43:59
.620923"]}}')::trajectory b
)
SELECT ST_2dIntersects(a,b), ST_2dtIntersects(a,b), ST_3dIntersects(a,b), ST_3dtIntersects(
a,b) from traj;
 st_2dintersects | st_2dtintersects | st_3dintersects | st_3dtintersects
-----------------+------------------+-----------------+------------------
 t               | f                | t               | f
```

8.12.6. ST_{2D|2DT|3D|3DT}Intersects_IndexLeft

This function is used to determine whether two objects intersect on a specified axis. This function also allows you to specify to use the index on the column that stores the first specified object.

Syntax

```
bool ST_{2D|2DT|3D|3DT}Intersects_IndexLeft(trajectory traj1, trajectory traj2, timestamp t
s, timestamp te);
```

Parameters

Parameter	Description
traj1	The trajectory that you want to compare, or the original trajectory that includes the sub-trajectory that you want to compare.
traj2	The trajectory that you want to compare, or the original trajectory that includes the sub-trajectory that you want to compare.
ts	The beginning of the time range over which you want to extract sub-trajectories. This parameter is optional.
te	The end of the time range over which you want to extract sub-trajectories. This parameter is optional.

Description

If you call the ST_ndIntersects function to compare two sub-trajectories by using the `ST_{2D|2DT|3D|3DT}Intersects(trajectory traj1, trajectory traj2, timestamp ts, timestamp te)` syntax, the function cannot determine which index to use. This causes additional computing overheads.

We recommend that you use the `ST_ndIntersects_IndexLeft(...)` function. This function allows you to manually specify to use the index on the column that stores the first specified sub-trajectory. This reduces computing overheads.

Example

```
EXPLAIN SELECT count(*) FROM sqltr_test_trajs WHERE ST_2DIntersects_IndexLeft(traj, ST_make
Trajectory('STPOINT', 'LINESTRING(-70 -30, -72 -34, -73 -35)'::geometry, '[2000-01-01 00:01
:30, 2000-01-01 00:15:00)'::tsrange,
  '{"leafcount":3,"attributes":{"velocity": {"type": "integer", "length": 2,"nullable" : tr
ue,"value": [120,130,140]}, "accuracy": {"type": "float", "length": 4, "nullable" : false,"
value": [120,130,140]}, "bearing": {"type": "float", "length": 8, "nullable" : false,"value
": [120,130,140]}, "acceleration": {"type": "string", "length": 20, "nullable" : true,"valu
e": ["120","130","140"]}, "active": {"type": "timestamp", "nullable" : false,"value": ["Fri
Jan 01 11:35:00 2010", "Fri Jan 01 12:35:00 2010", "Fri Jan 01 13:30:00 2010"]}}, "events":
[{"2" : "Fri Jan 02 15:00:00 2010"}, {"3" : "Fri Jan 02 15:30:00 2010"}]}'), ST_PGEpochToTS
(0), ST_PGEpochToTs(7000));
Aggregate  (cost=4201.04..4201.05 rows=1 width=8)
   ->  Bitmap Heap Scan on sqltr_test_trajs  (cost=98.64..4199.77 rows=505 width=0)
         Recheck Cond: ('BOX2DT(-73 -35 2000-01-01 00:01:29.999994,-70 -30 2000-01-01 00:15
:00)'::boxndf &#& traj)
         Filter: _st_2dintersects(traj, '{"trajectory":{"version":1,"type":"STPOINT","leafc
ount":3,"start_time":"2000-01-01 00:01:30","end_time":"2000-01-01 00:15:00","spatial":"LINE
STRING(-70 -30,-72 -34,-73 -35)","timeline":["2000-01-01 00:01:30","2000-01-01 00:08:15","2
000-01-01 00:15:00"],"attributes":{"leafcount":3,"velocity":{"type":"integer","length":2,"n
ullable":true,"value":[120,130,140]},"accuracy":{"type":"float","length":4,"nullable":false
,"value":[120.0,130.0,140.0]},"bearing":{"type":"float","length":8,"nullable":false,"value"
:[120.0,130.0,140.0]},"acceleration":{"type":"string","length":20,"nullable":true,"value":[
"120","130","140"]},"active":{"type":"timestamp","length":8,"nullable":false,"value":["2010
-01-01 11:35:00","2010-01-01 12:35:00","2010-01-01 13:30:00"]}},"events":[{"2":"2010-01-02
15:00:00"},{"3":"2010-01-02 15:30:00"}]}}'::trajectory, '2000-01-01 00:00:00'::timestamp wi
thout time zone, '2000-01-01 01:56:40'::timestamp without time zone)
         ->  Bitmap Index Scan on sqltr_test_traj_gist_2dt  (cost=0.00..98.51 rows=1515 wid
th=0)
               Index Cond: (traj &#& 'BOX2DT(-73 -35 2000-01-01 00:01:29.999994,-70 -30 200
0-01-01 00:15:00)'::boxndf)
```

8.12.7. ST_{2D|2DT|3D|3DT}DWithin

This function is used to check whether the distance between two objects on a specified axis is less than or equal to a specified threshold.

Syntax

```
bool ST_{2D|3D}DWithin(geometry geom, trajectory traj, float8 dist);
bool ST_{2D|3D}DWithin(trajectory traj, geometry geom, float8 dist);
bool ST_{2D|3D}DWithin(geometry geom, trajectory traj, timestamp ts, timestamp te, float8 d
ist);
bool ST_{2D|3D}DWithin(trajectory traj, geometry geom, timestamp ts, timestamp te, float8 d
ist);
bool ST_{2D|2DT|3D|3DT}DWithin(boxndf box, trajectory traj, float8 dist);
bool ST_{2D|2DT|3D|3DT}DWithin(trajectory traj, boxndf box, float8 dist);
bool ST_{2D|2DT|3D|3DT}DWithin(boxndf box, trajectory traj, timestamp ts, timestamp te, flo
at8 dist);
bool ST_{2D|2DT|3D|3DT}DWithin(trajectory traj, boxndf box, timestamp ts, timestamp te, flo
at8 dist);
bool ST_{2D|2DT|3D|3DT}DWithin(trajectory traj1, trajectory traj2, float8 dist);
bool ST_{2D|2DT|3D|3DT}DWithin(trajectory traj1, trajectory traj2, timestamp ts, timestamp
te, float8 dist);
```

Parameters

Parameter	Description
geom	The geometry that you want to compare.
traj	The trajectory that you want to compare, or the original trajectory that includes the sub-trajectory that you want to compare.
traj1	The trajectory that you want to compare, or the original trajectory that includes the sub-trajectory that you want to compare.
traj2	The trajectory that you want to compare, or the original trajectory that includes the sub-trajectory that you want to compare.
box	The bounding box that you want to compare.
ts	The beginning of the time range over which you want to extract sub-trajectories. This parameter is optional.
te	The end of the time range over which you want to extract sub-trajectories. This parameter is optional.
dist	The threshold for the distance between the two objects.

Description

- For 2D and 3D operations, this function checks whether the distance between the two- or three-dimensional projections of two objects is less than or equal to the specified threshold.
- For 2DT and 3DT operations, this function checks whether the distance between two objects in a specified dimension at a specified point in time is less than or equal to the specified threshold.

If this function contains the ts and te parameters, it compares two objects to check whether the distance between the sub-trajectories of the two objects over the specified time range is less than or equal to a specified threshold. If this function does not contain the ts or te parameter, it compares two objects to check whether the distance between the complete trajectories of the two objects is less than or equal to a specified threshold. The two objects are specified by the traj or traj1 parameter and the traj2 parameter.

The `ST_DistanceWithin(..)` function delivers the same effect as the `ST_2DDWithin(...)` function (if one of the specified objects is a geometry) and the `ST_3DTDWithin(...)` function (if both the specified objects are trajectories).

Example

```
WITH traj AS(
    Select ST_makeTrajectory('STPOINT', 'LINESTRING(0 0 10, 50 0 10, 100 0 10)'::geometry,
                              ('[' || ST_PGEpochToTS(0) || ',' || ST_PGEpochToTS(100) || ')'
)::tsrange,
                              '{"leafcount":3,"attributes":{"velocity": {"type": "integer",
"length": 2,"nullable" : true,"value": [120,130,140]}, "accuracy": {"type": "float", "lengt
h": 4, "nullable" : false,"value": [120,130,140]}, "bearing": {"type": "float", "length": 8
, "nullable" : false,"value": [120,130,140]}, "acceleration": {"type": "string", "length":
20, "nullable" : true,"value": ["120","130","140"]}, "active": {"type": "timestamp", "nulla
ble" : false,"value": ["Fri Jan 01 11:35:00 2010", "Fri Jan 01 12:35:00 2010", "Fri Jan 01
13:30:00 2010"]}}, "events": [{"2" : "Fri Jan 02 15:00:00 2010"}, {"3" : "Fri Jan 02 15:30:
00 2010"}]}') a,
           ST_makeTrajectory('STPOINT', 'LINESTRING(0 20, 50 20, 100 20)'::geometry,
                              ('[' || ST_PGEpochToTS(0) || ',' || ST_PGEpochToTS(100) ||')')
::tsrange,
                              '{"leafcount":3,"attributes":{"velocity": {"type": "integer",
"length": 2,"nullable" : true,"value": [120,130,140]}, "accuracy": {"type": "float", "lengt
h": 4, "nullable" : false,"value": [120,130,140]}, "bearing": {"type": "float", "length": 8
, "nullable" : false,"value": [120,130,140]}, "acceleration": {"type": "string", "length":
20, "nullable" : true,"value": ["120","130","140"]}, "active": {"type": "timestamp", "nulla
ble" : false,"value": ["Fri Jan 01 11:35:00 2010", "Fri Jan 01 12:35:00 2010", "Fri Jan 01
13:30:00 2010"]}}, "events": [{"2" : "Fri Jan 02 15:00:00 2010"}, {"3" : "Fri Jan 02 15:30:
00 2010"}]}') b
)
SELECT ST_2dDWithin(a,b, ST_PGEpochToTS(0),ST_PGEpochToTS(49), 19), ST_3dDWithin(a,b, ST_PG
EpochToTS(0),ST_PGEpochToTS(49), 19),
       ST_2dDWithin(a,b, ST_PGEpochToTS(0),ST_PGEpochToTS(50), 20), ST_3dDWithin(a,b, ST_PG
EpochToTS(0),ST_PGEpochToTS(50), 20),
       ST_2dDWithin(a,b, ST_PGEpochToTS(0),ST_PGEpochToTS(70), 21), ST_3dDWithin(a,b, ST_PG
EpochToTS(0),ST_PGEpochToTS(70), 21),
       ST_2dtDWithin(a,b, ST_PGEpochToTS(0),ST_PGEpochToTS(49), 19), ST_3dtDWithin(a,b, ST_
PGEpochToTS(0),ST_PGEpochToTS(49), 19),
       ST_2dtDWithin(a,b, ST_PGEpochToTS(0),ST_PGEpochToTS(50), 20), ST_3dtDWithin(a,b, ST_
PGEpochToTS(0),ST_PGEpochToTS(50), 20),
       ST_2dtDWithin(a,b, ST_PGEpochToTS(0),ST_PGEpochToTS(70), 21), ST_3dtDWithin(a,b, ST_
PGEpochToTS(0),ST_PGEpochToTS(70), 21) from traj;
NOTICE:  One or both of the geometries is missing z-value. The unknown z-value will be rega
rded as "any value"
NOTICE:  One or both of the geometries is missing z-value. The unknown z-value will be rega
rded as "any value"
 st_2ddwithin | st_3ddwithin | st_2ddwithin | st_3ddwithin | st_2ddwithin | st_3ddwithin |
st_2dtdwithin | st_3dtdwithin | st_2dtdwithin | st_3dtdwithin | st_2dtdwithin | st_3dtdwith
in
--------------+--------------+--------------+--------------+--------------+--------------+-
--------------+---------------+---------------+---------------+---------------+------------
---
 f            | f            | t            | t            | t            | t            |
f             | f             | t             | t             | t             | t
```

8.12.8. ST_{2D|2DT|3D|3DT}DWithin_IndexLeft

This function is used to check whether the distance between two objects on a specified axis is less than or equal to a specified threshold. This function also allows you to specify to use the index on the column that stores the first specified object.

Syntax

```
bool ST_{2D|2DT|3D|3DT}DWithin_IndexLeft(trajectory traj1, trajectory traj2, float8 dist);
bool ST_{2D|2DT|3D|3DT}DWithin_IndexLeft(trajectory traj1, trajectory traj2, timestamp ts,
timestamp te, float8 dist);
```

Parameters

Parameter	Description
traj1	The trajectory that you want to compare, or the original trajectory that includes the sub-trajectory that you want to compare.
traj2	The trajectory that you want to compare, or the original trajectory that includes the sub-trajectory that you want to compare.
ts	The beginning of the time range over which you want to extract sub-trajectories. This parameter is optional.
te	The end of the time range over which you want to extract sub-trajectories. This parameter is optional.
dist	The threshold for the distance between the two objects.

Description

If you call the `ST_ndDWithin(...)` function to compare two trajectories by using the `ST_{2D|2DT|3D|3DT}DWithin(trajectory traj1, trajectory traj2, float8 dist)` or `ST_{2D|2DT|3D|3DT}DWithin(trajectory traj1, trajectory traj2, timestamp ts, timestamp te, float8 dist)` syntax, the function cannot determine which index to use. This causes additional computing overheads.

We recommend that you use the ST_ndDwithin_IndexLeft function. This function allows you to manually specify to use the index on the column that stores the first specified object. This reduces computing overheads.

Example

```
EXPLAIN SELECT count(*) FROM sqltr_test_trajs WHERE ST_3DTDWithin_IndexLeft(traj, ST_makeTr
ajectory('STPOINT', 'LINESTRING(-70 -30, -72 -34, -73 -35)'::geometry, '[2000-01-01 00:01:3
0, 2000-01-01 00:15:00)'::tsrange,
  '{"leafcount":3,"attributes":{"velocity": {"type": "integer", "length": 2,"nullable" : tr
ue,"value": [120,130,140]}, "accuracy": {"type": "float", "length": 4, "nullable" : false,"
value": [120,130,140]}, "bearing": {"type": "float", "length": 8, "nullable" : false,"value
": [120,130,140]}, "acceleration": {"type": "string", "length": 20, "nullable" : true,"valu
e": ["120","130","140"]}, "active": {"type": "timestamp", "nullable" : false,"value": ["Fri
Jan 01 11:35:00 2010", "Fri Jan 01 12:35:00 2010", "Fri Jan 01 13:30:00 2010"]}}, "events":
[{"2" : "Fri Jan 02 15:00:00 2010"}, {"3" : "Fri Jan 02 15:30:00 2010"}]}'), 2);
Aggregate  (cost=4341.89..4341.90 rows=1 width=8)
   ->  Bitmap Heap Scan on sqltr_test_trajs  (cost=103.31..4340.56 rows=529 width=0)
         Recheck Cond: ('BOX2DT(-75 -37 2000-01-01 00:01:29.999994,-68 -28 2000-01-01 00:15
:00)'::boxndf &/#& traj)
         Filter: _st_3dtdwithin(traj, '{"trajectory":{"version":1,"type":"STPOINT","leafcou
nt":3,"start_time":"2000-01-01 00:01:30","end_time":"2000-01-01 00:15:00","spatial":"LINEST
RING(-70 -30,-72 -34,-73 -35)","timeline":["2000-01-01 00:01:30","2000-01-01 00:08:15","200
0-01-01 00:15:00"],"attributes":{"leafcount":3,"velocity":{"type":"integer","length":2,"nul
lable":true,"value":[120,130,140]},"accuracy":{"type":"float","length":4,"nullable":false,"
value":[120.0,130.0,140.0]},"bearing":{"type":"float","length":8,"nullable":false,"value":[
120.0,130.0,140.0]},"acceleration":{"type":"string","length":20,"nullable":true,"value":["1
20","130","140"]},"active":{"type":"timestamp","length":8,"nullable":false,"value":["2010-0
1-01 11:35:00","2010-01-01 12:35:00","2010-01-01 13:30:00"]}},"events":[{"2":"2010-01-02 15
:00:00"},{"3":"2010-01-02 15:30:00"}]}}'::trajectory, '2'::double precision)
         ->  Bitmap Index Scan on sqltr_test_traj_gist_2dt  (cost=0.00..103.18 rows=1586 wi
dth=0)
               Index Cond: (traj &/#& 'BOX2DT(-75 -37 2000-01-01 00:01:29.999994,-68 -28 20
00-01-01 00:15:00)'::boxndf)
```

8.12.9. ST_{T|2D|2DT|3D|3DT}Contains

This function is used to check whether the first specified object includes the second specified object on a specified axis.

Syntax

```
bool ST_TContains(tsrange r, trajectory traj);
bool ST_TContains(trajectory traj, tsrange r);
bool ST_2DContains(geometry geom, trajectory traj);
bool ST_2DContains(trajectory traj, geometry geom);
bool ST_2DContains(geometry geom, trajectory traj, timestamp ts, timestamp te);
bool ST_2DContains(trajectory traj, geometry geom, timestamp ts, timestamp te);
bool ST_{2D|2DT|3D|3DT}Contains(boxndf box, trajectory traj);
bool ST_{2D|2DT|3D|3DT}Contains(boxndf box, trajectory traj, timestamp ts, timestamp te);
```

Parameters

Parameter	Description
geom	The geometry that you want to compare.

Parameter	Description
traj	The trajectory that you want to compare, or the original trajectory that includes the sub-trajectory that you want to compare.
box	The bounding box that you want to compare.
r	The time range to query.
ts	The beginning of the time range over which you want to extract sub-trajectories. This parameter is optional.
te	The end of the time range over which you want to extract sub-trajectories. This parameter is optional.

Description

This function allows you to check whether the first specified object includes the second specified object on a specified axis.

- If you specify a geometry, this function compares the two-dimensional projections of the complete trajectories or the sub-trajectories over a specified time range. Then, this function checks whether the geometry includes or is included in the other specified object.
- If the first specified object is a bounding box and the second specified object is a trajectory, this function compares the bounding box and the trajectory to check whether the trajectory or its sub-trajectory over a specified time range is included in the bounding box in all dimensions. If the bounding box, trajectory, or sub-trajectory does not have a specified dimension, this function considers this dimension to have any value. In this case, the axis in this dimension automatically meets the specified condition.

> **Note** This function is not supported for geometry types such as POLYHEDRALSURFACE.

Example

```
WITH traj AS(
    Select ST_makeTrajectory('STPOINT', 'LINESTRING(0 0, 50 50, 100 100)'::geometry,
                              ('[' || ST_PGEpochToTS(0) || ',' || ST_PGEpochToTS(100) || ')'
)::tsrange,
                              '{"leafcount":3,"attributes":{"velocity": {"type": "integer",
"length": 2,"nullable" : true,"value": [120,130,140]}, "accuracy": {"type": "float", "lengt
h": 4, "nullable" : false,"value": [120,130,140]}, "bearing": {"type": "float", "length": 8
, "nullable" : false,"value": [120,130,140]}, "acceleration": {"type": "string", "length":
20, "nullable" : true,"value": ["120","130","140"]}, "active": {"type": "timestamp", "nulla
ble" : false,"value": ["Fri Jan 01 11:35:00 2010", "Fri Jan 01 12:35:00 2010", "Fri Jan 01
13:30:00 2010"]}}, "events": [{"2" : "Fri Jan 02 15:00:00 2010"}, {"3" : "Fri Jan 02 15:30:
00 2010"}]}') b,
           ST_MakeBox3dt(0,0,0,ST_PgEpochToTS(0), 50,50,50, ST_PgEpochToTS(49)) a
)
SELECT ST_2dContains(a,b,ST_PGEpochToTS(0), ST_PGEpochToTS(49)), ST_3dContains(a,b,ST_PGEpo
chToTS(0), ST_PGEpochToTS(49)),
       ST_2dtContains(a,b,ST_PGEpochToTS(0), ST_PGEpochToTS(49)), ST_3dtContains(a,b,ST_PGE
pochToTS(0), ST_PGEpochToTS(49)),
       ST_2dContains(a,b,ST_PGEpochToTS(0), ST_PGEpochToTS(50)), ST_3dContains(a,b,ST_PGEpo
chToTS(0), ST_PGEpochToTS(50)),
       ST_2dtContains(a,b,ST_PGEpochToTS(0), ST_PGEpochToTS(50)), ST_3dtContains(a,b,ST_PGE
pochToTS(0), ST_PGEpochToTS(50)),
       ST_2dContains(a,b,ST_PGEpochToTS(0), ST_PGEpochToTS(70)), ST_3dContains(a,b,ST_PGEpo
chToTS(0), ST_PGEpochToTS(70)),
       ST_2dtContains(a,b,ST_PGEpochToTS(0), ST_PGEpochToTS(70)), ST_3dtContains(a,b,ST_PGE
pochToTS(0), ST_PGEpochToTS(70)) from traj;
 st_2dcontains | st_3dcontains | st_2dtcontains | st_3dtcontains | st_2dcontains | st_3dcon
tains | st_2dtcontains | st_3dtcontains | st_2dcontains | st_3dcontains | st_2dtcontains |
st_3dtcontains
---------------+---------------+----------------+----------------+---------------+---------
------+----------------+----------------+---------------+---------------+----------------+-
---------------
 t             | t             | t              | t              | t             | t
| f             | f             | f              | f              | f             | f
```

8.12.10. ST_{T|2D|2DT|3D|3DT}Within

This topic describes the ST_{T|2D|2DT|3D|3DT}Within function, which is used to determine whether the first specified object is included in the second specified object on a specified axis.

Syntax

```
bool ST_TWithin(tsrange r, trajectory traj);
bool ST_TWithin(trajectory traj, tsrange r);
bool ST_2DWithin(geometry geom, trajectory traj);
bool ST_2DWithin(trajectory traj, geometry geom);
bool ST_2DWithin(geometry geom, trajectory traj, timestamp ts, timestamp te);
bool ST_2DWithin(trajectory traj, geometry geom, timestamp ts, timestamp te);
bool ST_{2D|2DT|3D|3DT}Within(trajectory traj, boxndf box);
bool ST_{2D|2DT|3D|3DT}Within(trajectory traj, boxndf box, timestamp ts, timestamp te);
```

Parameters

Parameter	Description
geom	The geometry that you want to compare.
traj	The trajectory that you want to compare, or the original trajectory that includes the sub-trajectory that you want to compare.
box	The bounding box that you want to compare.
r	The time range that you want to compare.
ts	The beginning of the time range over which you want to extract sub-trajectories. This parameter is optional.
te	The end of the time range over which you want to extract sub-trajectories. This parameter is optional.

Description

This function allows you to determine whether the first specified object is included in the second specified object. This function works in the same way as the ST_ndContains function.

> **Note** This function is not supported for geometry types such as POLYHEDRALSURFACE.

Example

```
WITH traj AS(
    SELECT (' {"trajectory":{"version":1,"type":"STPOINT","leafcount":6,"start_time":"2000-01-01 03:15:42","end_time":"2000-01-01 05:16:43",' ||
            '"spatial":"LINESTRING(2 2 0,33.042158099636 36.832684322819 0,47.244002354518 47.230026333034 0,64.978971942887 60.618813472986 0,77.621717839502 78.012496630661 0,80 78 0)",' ||
            '"timeline":["2000-01-01 03:15:42","2000-01-01 03:39:54","2000-01-01 04:04:06","2000-01-01 04:28:18","2000-01-01 04:52:31","2000-01-01 05:16:43"]}}')::trajectory a,
           'LINESTRING(2 2 0,33.042158099636 36.832684322819 0,47.244002354518 47.230026333034 0,64.978971942887 60.618813472986 0,77.621717839502 78.012496630661 0,80 78 0)'::geometry b
)
SELECT ST_2dWithin(b,a) from traj;
 st_2dwithin
-------------
 t
```

8.13. Spatio-temporal processing

8.13.1. ST_intersection

This function returns a new trajectory object that indicates the intersection of trajectory objects 1 and 2 within the specified time range.

Syntax

```
geometry ST_intersection(trajectory traj1, trajectory traj2, tsrange range);
geometry ST_intersection(trajectory traj1, trajectory traj2, timestamp t1, timestamp t2);
```

Parameters

Parameter	Description
traj1	Trajectory object 1.
traj2	Trajectory object 2.
t1	The start time.
t2	The end time.
range	The time range.

Examples

```
Select ST_intersection((Select traj from traj_table where id=1), (Select traj from traj_tab
le where id=2), '2010-1-1 13:00:00', '2010-1-1 14:00:00');
```

8.14. Spatio-temporal statistics

8.14.1. ST_nearestApproachPoint

This function returns the spatial point in trajectory object 1 that is nearest to trajectory object 2 within the specified time range.

Syntax

```
geometry ST_nearestApproachPoint(trajectory traj1, trajectory traj2);
geometry ST_nearestApproachPoint(trajectory traj1, trajectory traj2,tsrange range);
geometry ST_nearestApproachPoint(trajectory traj1, trajectory traj2, timestamp t1, timestam
p t2);
```

Parameters

Parameter	Description
traj1	The trajectory object 1.
traj2	The trajectory object 2.

Parameter	Description
t1	The start time.
t2	The end time.
range	The time range.

Example

```
Select ST_nearestApproachPoint((Select traj from traj_table where id=1), (Select traj from
traj_table where id=2), '2010-1-1 13:00:00', '2010-1-1 14:00:00');
```

8.14.2. ST_nearestApproachDistance

This function returns the nearest distance between trajectory objects 1 and 2 within the specified time range.

Syntax

```
float8 ST_nearestApproachDistance(trajectory traj, trajectory traj2);
float8 ST_nearestApproachDistance(trajectory traj, trajectory traj2, tsrange range);
float8 ST_nearestApproachDistance(trajectory traj, trajectory traj2, timestamp t1, timestam
p t2);
```

Parameters

Parameter	Description
traj1	Trajectory object 1.
traj2	Trajectory object 2.
t1	The start time.
t2	The end time.
range	The time range.

Examples

```
Select ST_nearestApproachDistance((Select traj from traj_table where id=1), (Select traj fr
om traj_table where id=2), '2010-1-1 13:00:00', '2010-1-1 14:00:00');
```

8.15. Distance measurement

8.15.1. ST_length

This function returns the total length of a trajectory object, in meters.

Syntax

```
float8 ST_length(trajectory traj, integer srid default 0);
```

Parameters

Parameter	Description
traj	The trajectory object.
srid	The spatial reference system identifier (SRID) of the trajectory object. Default value: 0 (4326).

Description

This function computes the spherical length in meters based on the SRID of the trajectory object. You can specify the SRID for a trajectory object by setting the srid parameter. If not specified, the SRID is 4326 by default.

Examples

```
select st_length(traj) from traj where id = 2;
    st_length
------------------
 13494.6660605311
(1 row)
```

8.15.2. ST_euclideanDistance

This function returns the nearest Euclidean distance between two trajectory objects at the same time point.

Syntax

```
float ST_euclideanDistance(trajectory traj1, trajectory traj2);
```

Parameters

Parameter	Description
traj1	The trajectory object 1.
traj2	The trajectory object 2.

Description

The distance has been standardized.

Examples

```
Select ST_euclideanDistance((Select traj from traj_table where id=1), (Select traj from tra
j_table where id=2));
```

8.15.3. ST_mdistance

This function returns an array of the Euclidean distance between two trajectory objects at the same time point.

Syntax

```
float[] ST_mdistance(trajectory traj1, trajectory traj2);
```

Parameters

Parameter	Description
traj1	The trajectory object 1.
traj2	The trajectory object 2.

Description

The distance has not been standardized.

Examples

```
Select ST_mDistance((Select traj from traj_table where id=1), (Select traj from traj_table
where id=2));
```

8.16. Similarity analysis

8.16.1. ST_lcsSimilarity

This function computes the similarity between two trajectory objects based on the longest common sub-sequence (LCSS) algorithm.

Syntax

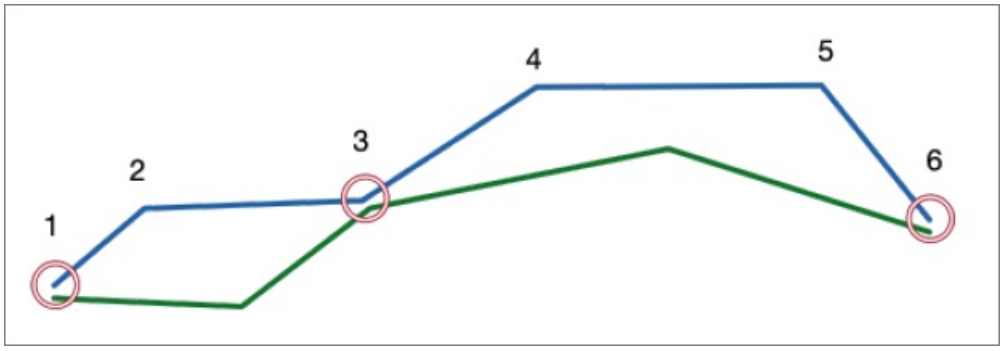
```
integer ST_lcsSimilarity(trajectory traj1, trajectory traj2, float8 dist, distanceUnit unit
default 'M');
integer ST_lcsSimilarity(trajectory traj1, trajectory traj2, float8 dist, interval lag, dis
tanceUnit unit default 'M');
```

Parameters

Parameter	Description
traj1	The trajectory object 1.
traj2	The trajectory object 2.
dist	The distance tolerance between two trajectory points. Unit: meters.
lag	The time tolerance between two trajectory points.
unit	The unit of the distance. Valid values: • 'M': meters. • 'KM': kilometers. • 'D': degrees. This value is valid only when the spatial reference system identifier (SRID) of a trajectory object is WGS84 (4326 by default).

Description

The LCSS algorithm is used to compute the maximum similarity between two trajectory objects. The consistency between two trajectory points is determined based on their distance in space and time. This function computes the similarity between two trajectory objects and returns the number of consistent trajectory points.



As shown in the preceding figure, trajectory points 1, 3, and 6 meet the conditions of consistency. In this case, this function returns 3.

Generally, a trajectory object has a valid SRID. If not specified, the SRID is 4326 by default.

Examples

```
With traj AS (
    Select ST_makeTrajectory('STPOINT', 'LINESTRINGZ(114.000528 33.588163 54.87, 114.000535
33.588235 54.85, 114.000447 33.588272 54.69, 114.000348 33.588287 54.73, 114.000245 33.5883
05 55.26, 114.000153 33.588305 55.3)'::geometry,
                              ARRAY['2010-01-01 11:30'::timestamp, '2010-01-01 11:31', '2010
-01-01 11:32', '2010-01-01 11:33','2010-01-01 11:34','2010-01-01 11:35'], NULL) a,
           ST_makeTrajectory('STPOINT', 'LINESTRINGZ(114.000529 33.588163 54.87, 114.000535
33.578235 54.85, 114.000447 33.578272 54.69, 114.000348 33.578287 54.73, 114.000245 33.5783
05 55.26, 114.000163 33.588305 55.3)'::geometry,
                              ARRAY['2010-01-01 11:29:58'::timestamp, '2010-01-01 11:31:02',
'2010-01-01 11:33', '2010-01-01 11:33:09','2010-01-01 11:34','2010-01-01 11:34:30'], NULL)
b)
Select st_LCSSimilarity(a, b, 100) from traj;
  st_lcssimilarity
-------------------
 2
 (1 row)
 With traj AS (
    Select ST_makeTrajectory('STPOINT', 'LINESTRINGZ(114.000528 33.588163 54.87, 114.000535
33.588235 54.85, 114.000447 33.588272 54.69, 114.000348 33.588287 54.73, 114.000245 33.5883
05 55.26, 114.000153 33.588305 55.3)'::geometry,
                              ARRAY['2010-01-01 11:30'::timestamp, '2010-01-01 11:31', '2010
-01-01 11:32', '2010-01-01 11:33','2010-01-01 11:34','2010-01-01 11:35'], NULL) a,
           ST_makeTrajectory('STPOINT', 'LINESTRINGZ(114.000529 33.588163 54.87, 114.000535
33.578235 54.85, 114.000447 33.578272 54.69, 114.000348 33.578287 54.73, 114.000245 33.5783
05 55.26, 114.000163 33.588305 55.3)'::geometry,
                              ARRAY['2010-01-01 11:29:58'::timestamp, '2010-01-01 11:31:02',
'2010-01-01 11:33', '2010-01-01 11:34:15','2010-01-01 11:34:50','2010-01-01 11:34:30'], NUL
L) b)
Select st_LCSSimilarity(a, b, 100, interval '30 seconds') from traj;
 st_lcssimilarity
-------------------
2
(1 row)
```

8.16.2. ST_lcsDistance

This function computes the distance between two trajectory objects based on the longest common sub-sequence (LCSS) algorithm.

Syntax

```
float8 ST_lcsDistance(trajectory traj1, trajectory traj2, float8 dist, distanceUnit unit de
fault 'M');
float8 ST_lcsDistance(trajectory traj1, trajectory traj2, float8 dist, interval lag, distan
ceUnit unit default 'M');
```

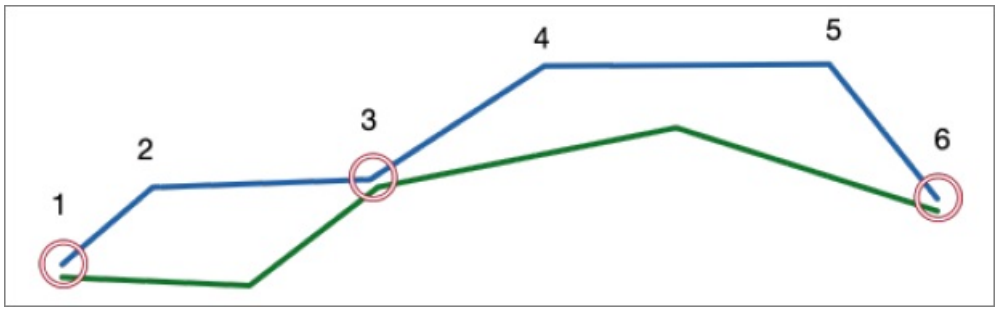
Parameters

Parameter	Description
traj1	The trajectory object 1.
traj2	The trajectory object 2.
dist	The distance tolerance between two trajectory points. Unit: meters.
lag	The time tolerance between two trajectory points.
unit	The unit of the distance. Valid values: • 'M': meters. • 'KM': kilometers. • 'D': degrees. This value is valid only when the spatial reference system identifier (SRID) of a trajectory object is WGS84 (4326 by default).

Description

The LCSS algorithm is used to compute the maximum similarity between two trajectory objects. The consistency between two trajectory points is determined based on their distance in space and time.

This function computes the distance between two trajectory objects by using the following formula: 1 - (LCSS)/min(leafcount(traj1), leafcount(traj2)).



As shown in the preceding figure, trajectory points 1, 3, and 6 meet the conditions of consistency. In this case, the value of LCSS is 3, and 1 - 3/5 = 0.4. This function returns 0.4.

Generally, a trajectory object has a valid SRID. If not specified, the SRID is 4326 by default.

Examples

```
With traj AS (
    Select ST_makeTrajectory('STPOINT', 'LINESTRINGZ(114.000528 33.588163 54.87, 114.000535
33.588235 54.85, 114.000447 33.588272 54.69, 114.000348 33.588287 54.73, 114.000245 33.5883
05 55.26, 114.000153 33.588305 55.3)'::geometry,
                              ARRAY['2010-01-01 11:30'::timestamp, '2010-01-01 11:31', '2010
-01-01 11:32', '2010-01-01 11:33','2010-01-01 11:34','2010-01-01 11:35'], NULL) a,
           ST_makeTrajectory('STPOINT', 'LINESTRINGZ(114.000529 33.588163 54.87, 114.000535
33.578235 54.85, 114.000447 33.578272 54.69, 114.000348 33.578287 54.73, 114.000245 33.5783
05 55.26, 114.000163 33.588305 55.3)'::geometry,
                              ARRAY['2010-01-01 11:29:58'::timestamp, '2010-01-01 11:31:02',
'2010-01-01 11:33', '2010-01-01 11:33:09','2010-01-01 11:34','2010-01-01 11:34:30'], NULL)
b)
Select st_LCSDistance(a, b, 100) from traj;
  st_lcsdistance
-------------------
 0.666666666666667
 (1 row)
 With traj AS (
    Select ST_makeTrajectory('STPOINT', 'LINESTRINGZ(114.000528 33.588163 54.87, 114.000535
33.588235 54.85, 114.000447 33.588272 54.69, 114.000348 33.588287 54.73, 114.000245 33.5883
05 55.26, 114.000153 33.588305 55.3)'::geometry,
                              ARRAY['2010-01-01 11:30'::timestamp, '2010-01-01 11:31', '2010
-01-01 11:32', '2010-01-01 11:33','2010-01-01 11:34','2010-01-01 11:35'], NULL) a,
           ST_makeTrajectory('STPOINT', 'LINESTRINGZ(114.000529 33.588163 54.87, 114.000535
33.578235 54.85, 114.000447 33.578272 54.69, 114.000348 33.578287 54.73, 114.000245 33.5783
05 55.26, 114.000163 33.588305 55.3)'::geometry,
                              ARRAY['2010-01-01 11:29:58'::timestamp, '2010-01-01 11:31:02',
'2010-01-01 11:33', '2010-01-01 11:34:15','2010-01-01 11:34:50','2010-01-01 11:34:30'], NUL
L) b)
Select st_LCSDistance(a, b, 100, interval '30 seconds') from traj;
 st_lcsdistance
-------------------
0.666666666666667
(1 row)
With traj AS (
    Select ST_makeTrajectory('STPOINT', 'LINESTRINGZ(114.000528 33.588163 54.87, 114.000535
33.588235 54.85, 114.000447 33.588272 54.69, 114.000348 33.588287 54.73, 114.000245 33.5883
05 55.26, 114.000153 33.588305 55.3)'::geometry,
                              ARRAY['2010-01-01 11:30'::timestamp, '2010-01-01 11:31', '2010
-01-01 11:32', '2010-01-01 11:33','2010-01-01 11:34','2010-01-01 11:35'], NULL) a,
           ST_makeTrajectory('STPOINT', 'LINESTRINGZ(114.000529 33.588163 54.87, 114.000535
33.578235 54.85, 114.000447 33.578272 54.69, 114.000348 33.578287 54.73, 114.000245 33.5783
05 55.26, 114.000163 33.588305 55.3)'::geometry,
                              ARRAY['2010-01-01 11:29:58'::timestamp, '2010-01-01 11:31:02',
'2010-01-01 11:33', '2010-01-01 11:34:15','2010-01-01 11:34:50','2010-01-01 11:34:30'], NUL
L) b)
Select st_LCSDistance(a, b, 100, interval '30 seconds', 'M') from traj;
 st_lcsdistance
-------------------
0.666666666666667
(1 row)
```

8.16.3. ST_lcsSubDistance

This function computes the distance between a longest common sub-sequence (LCSS) sub-trajectory and a trajectory object.

Syntax

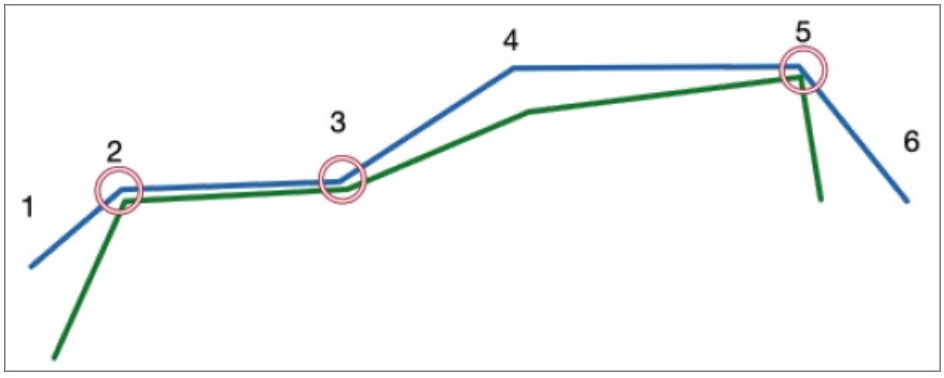
```
float8 ST_lcsSubDistance(trajectory traj1, trajectory traj2, float8 dist, distanceUnit unit
default 'M');
float8 ST_lcsSubDistance(trajectory traj1, trajectory traj2, float8 dist, interval lag, dis
tanceUnit unit default 'M');
```

Parameters

Parameter	Description
traj1	The trajectory object 1.
traj2	The trajectory object 2.
dist	The distance tolerance between two trajectory points. Unit: meters.
lag	The time tolerance between two trajectory points.
unit	The unit of the distance. Valid values: • 'M': meters. • 'KM': kilometers. • 'D': degrees. This value is valid only when the spatial reference system identifier (SRID) of a trajectory object is WGS84 (4326 by default).

Description

This function computes the sub-trajectory of trajectory object 1 in which trajectory points are consistent with those in the LCSS sub-trajectory, and then returns the ratio between the total number of trajectory points in the sub-trajectory of trajectory object 1 and the number of trajectory points in the LCSS sub-trajectory.



As shown in the preceding figure, trajectory points 2, 3, and 5 are in the LCSS sub-trajectory. Trajectory points 2, 3, 4, and 5 are in the sub-trajectory of trajectory object 1. In this case, 1 - 3/4 = 0.25. This function returns 0.25.

A smaller value indicates a higher similarity between the LCSS sub-trajectory and trajectory object 1.

Generally, a trajectory object has a valid SRID. If not specified, the SRID is 4326 by default.

Examples

```
With traj AS (
    Select ST_makeTrajectory('STPOINT', 'LINESTRINGZ(114.000528 33.588163 54.87, 114.000535
33.588235 54.85, 114.000447 33.588272 54.69, 114.000348 33.588287 54.73, 114.000245 33.5883
05 55.26, 114.000153 33.588305 55.3)'::geometry,
                             ARRAY['2010-01-01 11:30'::timestamp, '2010-01-01 11:31', '2010
-01-01 11:32', '2010-01-01 11:33','2010-01-01 11:34','2010-01-01 11:35'], NULL) a,
           ST_makeTrajectory('STPOINT', 'LINESTRINGZ(114.000529 33.588163 54.87, 114.000535
33.578235 54.85, 114.000447 33.578272 54.69, 114.000348 33.578287 54.73, 114.000245 33.5783
05 55.26, 114.000163 33.588305 55.3)'::geometry,
                             ARRAY['2010-01-01 11:29:58'::timestamp, '2010-01-01 11:31:02',
'2010-01-01 11:33', '2010-01-01 11:33:09','2010-01-01 11:34','2010-01-01 11:34:30'], NULL)
b)
Select st_LCSSubDistance(a, b, 100) from traj;
st_lcssubdistance
------------------------
 0.66666666666666666667
 (1 row)
 With traj AS (
    Select ST_makeTrajectory('STPOINT', 'LINESTRINGZ(114.000528 33.588163 54.87, 114.000535
33.588235 54.85, 114.000447 33.588272 54.69, 114.000348 33.588287 54.73, 114.000245 33.5883
05 55.26, 114.000153 33.588305 55.3)'::geometry,
                             ARRAY['2010-01-01 11:30'::timestamp, '2010-01-01 11:31', '2010
-01-01 11:32', '2010-01-01 11:33','2010-01-01 11:34','2010-01-01 11:35'], NULL) a,
           ST_makeTrajectory('STPOINT', 'LINESTRINGZ(114.000529 33.588163 54.87, 114.000535
33.578235 54.85, 114.000447 33.578272 54.69, 114.000348 33.578287 54.73, 114.000245 33.5783
05 55.26, 114.000163 33.588305 55.3)'::geometry,
                             ARRAY['2010-01-01 11:29:58'::timestamp, '2010-01-01 11:31:02',
'2010-01-01 11:33', '2010-01-01 11:33:09','2010-01-01 11:34','2010-01-01 11:34:30'], NULL)
b)
Select st_LCSSubDistance(a, b, 100, interval '30 seconds') from traj;
st_lcssubdistance
-------------------
0.666666666666667
(1 row)
```

8.16.4. ST_JaccardSimilarity

This topic describes the ST_JaccardSimilarity function, which calculates the Jaccard index based on two trajectories or sub-trajectories.

Syntax

```
record ST_JaccardSimilarity(trajectory tr1, trajectory tr2, double tol_dist,
                            text unit default '{}', interval tol_time default NULL,
                            timestamp ts default '-infinity', timestamp te default 'infinit
y') ;
```

Parameters

Parameter	Description
tr1	The trajectory numbered 1.
tr2	The trajectory numbered 2.
tol_dist	The tolerance that is allowed for the distance between a pair of trajectory points. Unit: meters.
unit	The JSON string that specifies how to calculate the distance.
tol_time	The tolerance that is allowed for the time difference between a pair of trajectory points. The default value is NULL. If you set this parameter to NULL or a negative value, this function calculates the Jaccard index based only on the distance and does not consider the time difference.
ts	The start of the time range for the calculation. The default value is `-infinity`. If you specify this parameter, this function calculates only the sub-trajectories between the specified start time and end time.
te	The end of the time range for the calculation. The default value is `infinity`. If you specify this parameter, this function calculates only the sub-trajectories between the specified start time and end time.

The following table describes the fields in the unit parameter.

Field	Type	Default value	Description

Field	Type	Default value	Description
Projection	string	None	The coordinate system to which the specified trajectories or sub-trajectories are re-projected. Valid values: • auto: This function dynamically selects the most suitable coordinate system based on the longitude and latitude coordinates of the specified trajectories or sub-trajectories. Then, this function re-projects the specified trajectories or sub-trajectories to the selected coordinate system. The supported coordinate systems include Lambert Azimuthal and UTM. You do not need to specify the Unit parameter again. The unit is meters. • srid: This function re-projects the specified trajectories or sub-trajectories based on the spatial reference identifier (SRID) that is specified by this parameter. **Note** If you do not specify this parameter, this function performs the calculation based on the original coordinate system.
Unit	string	null	The unit that is used to measure the distance. Valid values: • null: No units are specified. This function calculates the Euclidean distance between each pair of trajectory points based on the coordinates of these trajectory points. • M: This function calculates the distance between each pair of trajectory points based on the unit that is used for the spatial reference of the specified trajectories or sub-trajectories. In most cases, the distance is measured in the unit of meters.
useSpheroid	bool	true	Specifies whether to use an ellipsoid. You can specify this parameter if you set the Unit parameter to M. Valid values: • true: An ellipsoid is used. This function can obtain accurate distances. • false: A sphere that represents the earth is used. This function can obtain approximate distances.

The following table describes the response parameters.

Parameter	Type	Description
nleaf1	int	The number of trajectory points where Trajectory 1 intersects with Trajectory 2.
nleaf2	int	The number of trajectory points where Trajectory 2 intersects with Trajectory 1. **Note** The return value of this parameter may differ from that of the nleaf1 parameter. Assume that Trajectory 1 passes the same trajectory point on Trajectory 2 twice. In this case, the return value of the nleaf1 parameter is 1, and the return value of the nleaf2 parameter is 2.
inter1	int	The number of trajectory points between which the distance from Trajectory 1 to Trajectory 2 meets both the specified time and distance tolerances.
inter2	int	The number of trajectory points between which the distance from Trajectory 2 to Trajectory 1 meets both the specified time and distance tolerances.
jaccard_lower	double	The minimum distance that is calculated based on the Jaccard index. The value of this parameter is calculated based on the following formula: Minimum distance = `inter/(nleaf1 + nleaf2 - inter)`. **Note** In the preceding formula, the value of the inter parameter is the smaller value between the inter1 and inter2 parameters.
jaccard_upper	double	The maximum distance that is calculated based on the Jaccard index. The value of this parameter is calculated based on the following formula: Maximum distance = `inter/(nleaf1 + nleaf2 - inter)`. **Note** In the preceding formula, the value of the inter parameter is the larger one between the inter1 and inter2 parameters.

Description

The Jaccard index of two sets is the number of elements in the intersection of these sets divided by the number of elements in the union of these sets. For two trajectories, the definition of the Jaccard index is extended to calculate the number of trajectory points where Trajectory 1 intersects with Trajectory 2, the number of trajectory points where Trajectory 2 intersects with Trajectory 1, and the distance between each pair of trajectory points. The distances are calculated based on the preceding formulas that are used to calculate the values of the jaccard_lower and jaccard_upper parameters.

Example

```
With traj as(
SELECT ST_makeTrajectory('STPOINT'::leaftype, 'SRID=4326;LINESTRING(114.49211 37.97921,114.
49211 37.97921,114.49211 37.97921,114.49211 37.97921)'::geometry,
    ARRAY[to_timestamp(1590287775) AT TIME ZONE 'UTC', to_timestamp(1590287778) AT TIME ZON
E 'UTC', to_timestamp(1590302169) AT TIME ZONE 'UTC',to_timestamp(1590302171) AT TIME ZONE
'UTC'], '{}') a,
    ST_makeTrajectory('STPOINT'::leaftype, 'SRID=4326;LINESTRING(114.49211 37.97921,114.492
11 37.97921,114.49211 37.97921,114.49211 37.97921,114.49145 37.97781,114.49145 37.97781,114
.49145 37.97781,114.49145 37.97781,114.49145 37.97781,114.49145 37.97781,114.49145 37.97781
,114.49145 37.97781,114.49145 37.97781,114.49145 37.97781,114.49211 37.97921,114.49211 37.9
7921,114.49211 37.97921,114.49211 37.97921,114.49211 37.97921,114.49211 37.97921)'::geometr
y,
    ARRAY[ to_timestamp(1590287765) AT TIME ZONE 'UTC', to_timestamp(1590287771) AT TIME ZO
NE 'UTC', to_timestamp(1590287778) AT TIME ZONE 'UTC', to_timestamp(1590287780) AT TIME ZON
E 'UTC', to_timestamp(1590295992) AT TIME ZONE 'UTC', to_timestamp(1590295997) AT TIME ZONE
'UTC', to_timestamp(1590296013) AT TIME ZONE 'UTC', to_timestamp(1590296018) AT TIME ZONE '
UTC',  to_timestamp(1590296025) AT TIME ZONE 'UTC', to_timestamp(1590296032) AT TIME ZONE '
UTC', to_timestamp(1590296055) AT TIME ZONE 'UTC', to_timestamp(1590296073) AT TIME ZONE 'U
TC', to_timestamp(1590296081) AT TIME ZONE 'UTC', to_timestamp(1590296081) AT TIME ZONE 'UT
C', to_timestamp(1590302169) AT TIME ZONE 'UTC', to_timestamp(1590302174) AT TIME ZONE 'UTC
',  to_timestamp(1590302176) AT TIME ZONE 'UTC', to_timestamp(1590302176) AT TIME ZONE 'UTC
', to_timestamp(1590302172) AT TIME ZONE 'UTC', to_timestamp(1590302176) AT TIME ZONE 'UTC'
],'{}') b
)
select ST_JaccardSimilarity(a,b, 100,'{"unit":"M"}','20 second', '2020-05-23'::timestamptz
AT TIME ZONE 'UTC', '2020-05-26'::timestamptz AT TIME ZONE 'UTC') from traj;
       st_jaccardsimilarity
-----------------------------------
 (4,20,4,10,0.2,0.714285714285714)
(1 row)
```

8.17. Indexing

8.17.1. GiST indexing

This topic describes GiST indexing. You can create a GiST index on a column that stores trajectory data.

Syntax

```
CREATE INDEX [index_name] on table_name USING GIST(traj_col [operator_family]);
```

- index_name: the name of the index. This parameter is optional.

- table_name: the name of the table to which the column belongs.
- traj_col: the name of the column.
- operator_family: the operator family that is used to create the index. Default value: trajgist_ops_multi. You can use the ganos.trajectory.index_split_config parameter to specify the operator family. This parameter is optional.

> **Note** You can use the index that you create to accelerate the queries that are run by various operators and the following functions: ST_ndIntersect, ST_ndDWithin, ST_ndContains, and ST_ndWithin.

Supported operator families

The following table describes the operator families that are supported for GiST indexing.

Operator family	Description
trajgist_ops_z	Creates an index on the z axis. This type of index supports queries that cover only the z axis.
trajgist_ops_t	Creates an index on the t axis. This type of index supports queries that cover only the t axis.
trajgist_ops_2d	Creates an index on the x and y axes. This type of index supports queries that cover only the x and y axes.
trajgist_ops_2dt	Creates an index on the x, y, and t axes. This type of index supports queries that cover only the x and y axes, queries that cover only the t axis, and queries that cover the x, y, and t axes.
trajgist_ops_3d	Creates an index on the x, y, and z axes. This type of index supports queries that cover only the x and y axes, queries that cover only the z axis, and queries that cover the x, y, and z axes.
trajgist_ops_3dt	Creates an index on the x, y, z, and t axes. This type of index supports all queries that are supported by the preceding five operator families.
trajgist_ops_multi	Creates an index on multiple bounding boxes to accelerate queries. However, the system requires more time and storage to create the index.

Examples

- Create an index on the t axis.

```
CREATE INDEX on table_name USING GIST(traj_col trajgist_ops_t);
```

- Create an index on the x and y axes.

```
CREATE INDEX on table_name USING GIST(traj_col trajgist_ops_2d);
```

- Create an index on the x, y, and t axes.

```
CREATE INDEX on table_name USING GIST(traj_col trajgist_ops_2dt);
```

8.17.2. TrajGiST indexing

This topic describes TrajGiST indexing, which is an extension to GiST indexing. You can create a TrajGiST index on a column that stores trajectory data.

Background information

TrajGiST performs better than GiST in the following ways:

- TrajGiST provides a better method to estimate the overheads of indexes. If more than one index is created, TrajGiST can select an index in a more efficient manner than GiST.
- TrajGiST supports upward compatibility of indexes. If an accurate result cannot be obtained because the index does not provide sufficient information, you can still obtain a coarse-grained result by using an index-based scan. For example, you have created an index only on the t axis, but you want to call the ST_{2DT}Intersects function. In this case, you can filter out the trajectories that do not fall within a specified time range based on the index on the t axis.

Syntax

```
CREATE INDEX [index_name] on table_name USING TRAJGIST(traj_col [operator_family]);
```

- index_name: the name of the index. This parameter is optional.
- table_name: the name of the table to which the column belongs.
- traj_col: the name of the column.
- operator_family: the operator family that is used to create the index. Default value: trajgist_ops_multi. You can use the ganos.trajectory.index_split_config parameter to specify the operator family. This parameter is optional.

> **Note** The created index helps accelerate queries that are run by operators and the following functions: ST_ndIntersect, ST_ndDWithin, ST_ndContains, and ST_ndWithin.

Supported operator families

The following table describes the operator families that are supported by TrajGisT.

Operator family	Description
trajgist_ops_z	Creates an index on the z axis. This type of index supports queries that cover only the z axis.
trajgist_ops_t	Creates an index on the t axis. This type of index supports queries that cover only the t axis.
trajgist_ops_2d	Creates an index on the x and y axes. This type of index supports queries that cover only the x and y axes.
trajgist_ops_2dt	Creates an index on the x, y, and t axes. This type of index supports queries that cover the x, y, and t axes.
trajgist_ops_3d	Creates an index on the x, y, and z axes. This type of index supports queries that cover the x, y, and z axes.

Operator family	Description
trajgist_ops_3dt	Creates an index on the x, y, z, and t axes. This type of index supports all queries that are supported by the preceding five operator families.
trajgist_ops_multi	Creates an index on multiple bounding boxes to accelerate queries. However, the system requires more time and storage to create the index.

Note TrajGiST allows you to create an index on a single column. TrajGiST does not allow you to create an index on multiple columns when some of the columns store other types of data than trajectory data.

8.18. External storage

8.18.1. ST_ExportTo

You can use this function to export the data of a trajectory object to a folder in an Object Storage Service (OSS) bucket.

Syntax

```
trajectory ST_ExportTo(trajectory traj, text path, text config);
```

Parameters

Parameter	Description
traj	The trajectory object.
path	The string that specifies the folder in which you want to store the data of the specified trajectory object. Only folders in OSS are supported. The format of the parameter value is 'OSS://<AccessKey ID>:< AccessKey secret>@<Endpoint>/<Bucket>/<Directory>'. The data of the specified trajectory object is stored in a .gtf file in the specified folder.
config	A JSON string. The following list describes the parameter in this string: • Parameter: compress • Default value: none • Valid values: none, lz4, lzo, zstd, snappy, and zlib

Description

This function stores the time, space, event, and property information of a specified trajectory object to the specified folder in OSS.

Replace <Endpoint> with the endpoint that you use to access OSS in the region where the trajectory data is stored. To ensure import performance, make sure that the PolarDB database is deployed in the same region as the OSS bucket. For more information, see OSS domain names.

The binary data of the trajectory object is appended to a *.gtf* file in the specified folder. The format of the .gtf file name is *<Sequence number of the file>_0_1.gtf*. The .gtf files in a folder are sorted in descending order by sequence number. The size of a single file is approximately 2 GB. The size is specified by the ganos.trajectory.ext_storage_block_size parameter.

If a *.gtf* file that stores the data of the specified trajectory object already exists in the specified folder, the system appends the data of the trajectory object to the existing *.gtf* file. If the data of the specified trajectory object is stored in a folder in OSS and the system detects that the trajectory data is already stored in the specified folder, the system does not write the data to the folder again. If the system detects that the trajectory data is not stored in the specified folder, the system reads the data and then writes the data to the folder.

If you delete or update the data in the database that stores the trajectory data, the data in the OSS file is not updated. If you update the data in the OSS file, the data in the database can become unavailable.

You can choose a compression algorithm based on your business requirements. The zlib algorithm provides the highest compression ratio. The zstd algorithm provides the highest overall efficiency.

Examples

```
update trajs
set traj = ST_exportTo(traj, 'OSS://<access key>:<access secret>@oss-cn-beijing-internal/<b
ucket>/<directory>', '{}');
UPDATE 113395
```

8.18.2. ST_IsExternal

You can use this function to check whether the data of a trajectory object is stored in Object Storage Service (OSS).

Syntax

```
bool ST_IsExternal(trajectory traj);
```

Parameters

Parameter	Description
traj	The trajectory object.

Description

If the data of the specified trajectory object is stored in OSS, the function returns true. Otherwise, this function returns false.

Examples

```
select ST_IsExternal(traj) from trajs;
 st_isexternal
---------------
 t
 t
 t
 f
 t
(5 rows)
```

8.18.3. ST_importFrom

You can use this function to import the data of a trajectory object from Object Storage Service (OSS) to the PolarDB database that is used to store the trajectory data.

Syntax

```
trajectory ST_importFrom(trajectory traj);
```

Parameters

Parameter	Description
traj	The trajectory object.

Description

This function performs operations that are the reverse of the operations performed by the ST_ExportTo function. If the data of the specified trajectory object is stored in a PolarDB database table, this function returns the trajectory data. If the trajectory data is stored in a file in OSS, this function imports the trajectory data into the PolarDB database that is used to store the trajectory data. You can use this function and the `Update` command to change the data storage location for the specified trajectory object from OSS to the PolarDB database that is used to store the trajectory data.

This function does not change the OSS file that stores the trajectory data.

Examples

```
update trajs
set traj = ST_ImportFrom(traj);
UPDATE 4
```

8.18.4. ST_StorageLocation

You can use this function to query the location of the Object Storage Service (OSS) file that stores the data of a trajectory object.

Syntax

```
bool ST_StorageLocation(trajectory traj);
```

Parameters

Parameter	Description
traj	The trajectory object.

Description

If the data of the specified trajectory object is stored in a *.gtf* file in OSS, this function returns the information of the folder that stores the file. Otherwise, this function returns null.

Examples

```
select ST_StorageLocation(traj) from trajs;
                                                st_storagelocation

-----------------------------------------------------------------------------------------------
-----------------------------
 OSS://$(OSS_USER):<AccessKeySecret>@$(OSS_ENDPOINT)/$(OSS_BUCKET)/$(OSS_E_EXPORT)1/
 OSS://$(OSS_USER):<AccessKeySecret>@$(OSS_ENDPOINT)/$(OSS_BUCKET)/$(OSS_E_EXPORT)2/
 OSS://$(OSS_USER):<AccessKeySecret>@$(OSS_ENDPOINT)/$(OSS_BUCKET)/$(OSS_E_EXPORT)2/
(4 rows)
```

8.18.5. ST_AKID

You can use this function to query the AccessKey ID that is used to access the Object Storage Service (OSS) bucket in which the data of a trajectory object is stored.

Syntax

```
text ST_AKID(trajectory traj);
```

Parameters

Parameter	Description
traj	The trajectory object.

Description

If the data of a specified trajectory is stored in an OSS bucket, this function returns the AccessKey ID that is used to access the bucket. Otherwise, this function returns null.

When you read trajectory data from a file in an OSS bucket, you must provide the AccessKey pair information to access the OSS bucket. An AccessKey pair consists of an AccessKey ID and an AccessKey secret. An AccessKey ID is similar to a username and an AccessKey secret is similar to a password.

Examples

```
select ST_AKID(traj) from trajs;
         st_akid
--------------------------
 $(OSS_USER)
 $(OSS_USER)
 $(OSS_USER)
```

8.18.6. ST_SetAccessKey

You can invoke this function to set an AccessKey pair. The AccessKey pair is used to access the Object Storage Service (OSS) bucket that stores the data of a trajectory object.

Syntax

```
text ST_SetAccessKey(trajectory traj, text akid, text aksecret, bool checkvalid default tru
e);
```

Parameters

Parameter	Description
traj	The trajectory object.
akid	The AccessKey ID that is used to access the OSS bucket.
aksecret	The AccessKey secret that is used to access the OSS bucket.
checkvalid	Specifies whether to check the validity of the AccessKey pair. Default value: true. If this parameter is set to true, the system tests whether the specified AccessKey pair can be used to access the OSS bucket. If the access test fails, an error is returned.

Description

This function resets an AccessKey pair. The AccessKey pair is used to access the OSS bucket that stores the data of a specified trajectory object.

When you read trajectory data from a file in an OSS bucket, you must provide the AccessKey pair information to access the OSS bucket. An AccessKey pair consists of an AccessKey ID and an AccessKey secret. An AccessKey ID is similar to a username and an AccessKey secret is similar to a password.

Examples

```
select ST_SetAccessKey(traj, '<OSS_USER>', '<OSS_PWD>') from trajs;

st_setaccesskey

--------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------
---------------------------------
 null
 null
 null
 TRAJECTORY EMPTY
(4 rows)
```

8.18.7. ST_SetAkId

You can use this function to set an AccessKey ID. The AccessKey ID is used to access the Object Storage Service (OSS) bucket that stores data of a trajectory object.

Syntax

```
text ST_SetAkId(trajectory traj, text akid, bool checkvalid default true);
```

Parameters

Parameter	Description
traj	The trajectory object.
akid	The AccessKey ID that is used to access the OSS bucket.
checkvalid	Specifies whether to check the validity of the AccessKey ID. Default value: true. If this parameter is set to true, the system tests whether the specified AccessKey ID can be used to access the OSS bucket. If the access test fails, an error is returned.

Description

This function resets an AccessKey ID. The AccessKey ID is used to access the OSS bucket that stores the data of a specified trajectory object.

When you read trajectory data from a file in an OSS bucket, you must provide the AccessKey pair information to access the OSS bucket. An AccessKey pair consists of an AccessKey ID and an AccessKey secret. An AccessKey ID is similar to a username and an AccessKey secret is similar to a password.

Examples

```
select ST_SetAKId(traj, '<OSS_USER>') from trajs;

st_setakid

--------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------
---------------------------------
 null
 null
 null
 TRAJECTORY EMPTY
(4 rows)
```

8.18.8. ST_SetAkSecret

You can use this function to set an AccessKey secret. The AccessKey secret is used to access the Object Storage Service (OSS) bucket that stores the data of a specified trajectory object.

Syntax

```
text ST_SetAkSecret(trajectory traj, text aksecret, bool checkvalid default true);
```

Parameters

Parameter	Description
traj	The trajectory object.
aksecret	The AccessKey secret that is used to access the OSS bucket.
checkvalid	Specifies whether to check the validity of the AccessKey secret. Default value: true. If this parameter is set to true, the system tests whether the specified AccessKey secret can be used to access the OSS bucket. If the access test fails, an error is returned.

Description

This function resets an AccessKey secret. The AccessKey secret is used to access the OSS bucket that stores the data of a specified trajectory.

When you read trajectory data from a file in an OSS bucket, you must provide the AccessKey pair information to access the OSS bucket. An AccessKey pair consists of an AccessKey ID and an AccessKey secret. An AccessKey ID is similar to a username and an AccessKey secret is similar to a password.

Examples

```
select ST_SetAkSecret(traj, '<OSS_PWD>') from trajs;

st_setaksecret

----------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------
---------------------------------
 null
 null
 null
 TRAJECTORY EMPTY
(4 rows)
```

8.18.9. ST_SetStorageLocation

You can use this function to change the location of the Object Storage Service (OSS) file that stores the data of a trajectory object.

Syntax

```
bool ST_SetStorageLocation(trajectory traj, text new_path, bool checkValid default true);
```

Parameters

Parameter	Description
traj	The trajectory object.
new_path	The string that specifies the folder in which you want to store the data of the specified trajectory object. Only folders in OSS are supported. The format of the parameter value is 'OSS://<AccessKey ID>:< AccessKey secret>@<Endpoint>/<Bucket>/<Directory>'. The data of the specified trajectory object is stored in a *.gtf* file in the specified folder.
checkvalid	Specifies whether to check the validity of the path. Default value: true. If this parameter is set to true, the system tests whether the specified OSS directory can be accessed by using your AccessKey pair. If the access test fails, an error is returned.

Description

This function changes the location of the file that stores the data of a specified trajectory object. This function does not move the original file that stores the data of the trajectory object. Before you use this function to specify a new location, you first need to move the folder that contains the trajectory data file in OSS to the new location.

After the file is moved to the new location, new trajectory data may not be appended to the file. In this case, an error message that contains `is not appendable` is returned. Make sure that the last data block is uploaded by calling the AppendObject operation of OSS.

Examples

```
select ST_SetStorageLocation(traj, '<New Path>') from trajs;

st_setstoragelocation

-----------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------
---------------------------------
 null
 null
 null
 TRAJECTORY EMPTY
(4 rows)
```

8.18.10. ST_DeleteGTF

You can use this function to delete all the trajectory data files from an Object Storage Service (OSS) folder.

Syntax

```
void ST_DeleteGTF(text directory);
```

Parameters

Parameter	Description
directory	The string that specifies the folder in which trajectory data files are located. Only folders in OSS are supported. The format of the parameter value is 'OSS://<AccessKey ID>:< AccessKey secret>@<Endpoint>/<Bucket>/<Directory>'. Trajectory data is stored in *.gtf* files in the specified folder.

Description

This function deletes the files that store trajectory data.

Examples

```
select ST_deleteGtf('OSS_PATH');
 st_deletegtf
--------------
(1 row)
```

8.19. Variables

8.19.1. ganos.trajectory.attr_string_length

This variable sets a default length for string-type attribute fields.

Description

The value of the `attr_string_length` parameter is integer.

Examples

```
Set ganos.trajectory.attr_string_length = 32;
```

8.19.2. ganos.trajectory.index_split_config

This topic describes the ganos.trajectory.index_split_config parameter. This parameter is used to specify the rule that is used to split a trajectory when GiST indexing is used.

Description

The value of the `index_split_config` parameter is a string. The value of the index_split_config parameter is in the same format as the value of the config parameter of the ST_Split function.

Default value

`"{\"cut_point.even_divide\":10}"`

9.GeomGrid SQL reference

9.1. Usage notes

GeomGrid is the data type that is used to represent grids in the Ganos GeomGrid extension.

Data types

- The following data types can be converted into the GeomGrid data type:
  - text
  - bytea
- The GeomGrid data type can be converted into the following data types:
  - text
  - bytea
  - geometry
  - box

9.2. Functions for output

9.2.1. ST_AsText

This topic describes the ST_AsText function, which converts a grid into a text that is encoded in a specified format.

Syntax

```
text ST_AsText(geomgrid    grid,
               integer precision,
               text standard default 'GGER')
text[] ST_AsText(geomgrid[]    grid,
               integer precision,
               text standard default 'GGER')
```

Parameters

Parameter	Description
grid	The grid that you want to convert.
precision	The precision level based on which you want to convert the grid. Valid values: 0 to 31. The value -1 specifies the default precision level.
standard	The specifications based on which you want to convert the grid. Set the value to GGER. Geospatial Grid Encoding Rule (GGER) is developed by the Ministry of Natural Resources of the People's Republic of China. Default value: GGER.

Description

This function converts a grid into a text based on the specified layer, precision level, and encoding format. If the specified precision level is higher than the precision level of the grid, the system does not fill 0s in the return result.

Examples

```
--Retain the default layer that you want to convert.
with g as (
  select unnest(st_asgrid(
    ST_geomfromtext('POINT(116.31522216796875 39.910277777777778)',4490), 15)) as grid)
select ST_asText(grid) from g;
    st_astext
------------------
 G001310322230230
--Specify the layer that you want to convert.
with g as (
  select unnest(st_asgrid(
    ST_geomfromtext('POINT(116.31522216796875 39.910277777777778)',4490), 15)) as grid)
select ST_asText(grid, 8) from g;
  st_astext
-------------
 G01310322
```

9.2.2. ST_AsBinary

This topic describes the ST_AsBinary function, which converts a grid into a binary structure.

Syntax

```
bytea ST_AsBinary(geomgrid    grid)
```

Parameters

Parameter	Description
grid	The grid that you want to convert.

Examples

```
with g as (
select unnest(st_asgridcode(
    ST_geomfromtext('POINT(116.31522216796875 39.910277777777778)',4490), 15)) as grid)
select ST_asBinary(grid) from g;
   st_asbinary
--------------------
 \x01010c0f74271236
```

9.2.3. ST_AsGeometry

This topic describes the ST_AsGeometry function, which obtains the geometry-represented range of a grid.

Syntax

```
geometry ST_AsGeometry(geomgrid grid);
geometry[] ST_AsGeometry(geomgrid[] grid);
```

Parameters

Parameter	Description
grid	The grid whose range you want to obtain.

Examples

```
select st_astext(
   st_asgeometry(st_gridfromtext('G0013103220310313')));
      st_astext
--------------------------------------------------------------------------------
 POLYGON((116.458888888889 39.3088888888889,116.458888888889 39.3166666666667,11
6.466666666667 39.3166666666667,116.466666666667 39.3088888888889,116.4588888888
89 39.3088888888889))
```

9.2.4. ST_AsBox

This topic describes the ST_AsBox function, which obtains the box-represented range of a grid.

Syntax

```
box ST_AsBox(geomgrid grid);
box[] ST_AsBox(geomgrid[] grid);
```

Parameters

Parameter	Description
grid	The grid whose range you want to obtain.

Examples

```
select st_asbox(
   st_gridfromtext('G0013103220310313'));
                                    st_asbox
--------------------------------------------------------------------------------
 (116.46666666666667,39.31666666666667),(116.4588888888889,39.30888888888889)
```

9.3. Functions for input

9.3.1. ST_GridFromText

This topic describes the ST_GridFromText function, which converts a grid from a string into a GeomGrid structure.

Syntax

```
geomgrid ST_GridFromText(text   Code)
```

Parameters

Parameter	Description
Code	The string of the grid that you want to convert.

Examples

```
select st_gridfromtext('G0013103220310313');
   st_gridfromtext
----------------------
 01011C1074271B120101
```

9.3.2. ST_GridFromBinary

This topic describes the ST_GridFromBinary function, which converts a grid from a binary structure into a GeomGrid structure.

Syntax

```
geomgrid ST_GridFromBinary(bytea   binary)
```

Parameters

Parameter	Description
binary	The binary structure of the grid that you want to convert.

Examples

```
select st_astext(
  st_gridfrombinary('\x01010c0f74271236'));
    st_astext
------------------
 G001310322230230
```

9.4. Functions to query spatial relationships

9.4.1. ST_Intersects

This topic describes the ST_Intersects function, which queries whether a grid intersects with a geometry.

Syntax

```
boolean ST_Intersects(geomgrid grid, geometry geom);
boolean ST_Intersects(geometry geom, geomgrid grid);
```

Parameters

Parameter	Description
grid	The grid whose spatial relationship you want to query.
geom	The geometry whose spatial relationship you want to query.

Description

This function returns the spatial relationship between a grid and a geometry. The geometry must use the CGC2000 spatial reference system. In addition, the spatial reference system identifier (SRID) of the geometry must be 4490.

Examples

```
select st_intersects(ST_gridfromtext('G001331032213300013'),
      ST_geomfromtext('LINESTRING(122.48077 51.72814, 122.47416 51.73714)',4490));
 st_intersects
---------------
 t
```

9.4.2. ST_Contains

This topic describes the ST_Contains function, which queries whether a grid contains a geometry.

Syntax

```
boolean ST_Contains(geomgrid grid, geometry geom);
boolean ST_Contains(geometry geom, geomgrid grid);
boolean ST_Contains(geomgrid grid1, geomgrid grid2);
```

Parameters

Parameter	Description
grid	The grid whose spatial relationship you want to query.
geom	The geometry whose spatial relationship you want to query.

Examples

```
select st_contains(ST_gridfromtext('G001331032213300013'),
      ST_geomfromtext('POINT(116.31522216796875 39.910277777777778)',4490));
 st_intersects
---------------
 f
 select st_contains(ST_gridfromtext('G00133103221330'),
                    ST_gridfromtext('G001331032213300013'))
  st_contains
-------------
 t
```

9.4.3. ST_Within

This topic describes the ST_Within function, which queries whether a grid is contained in a geometry.

Syntax

```
boolean ST_Within(geomgrid grid, geometry geom);
boolean ST_Within(geometry geom, geomgrid grid);
boolean ST_Within(geomgrid grid1, geomgrid grid2);
```

Parameters

Parameter	Description
grid	The grid whose spatial relationship you want to query.
geom	The geometry whose spatial relationship you want to query.

Description

This function returns the spatial relationship between a grid and a geometry. The geometry must use the CGC2000 spatial reference system. In addition, the spatial reference system identifier (SRID) of the geometry must be 4490.

Examples

```
select st_within(ST_geomfromtext('POINT(116.31522216796875 39.910277777777778)',4490),
  ST_gridfromtext('G001331032213300013'));
 st_within
---------------
 f
 select st_within(ST_gridfromtext('G001331032213300013'),
                  ST_gridfromtext('G001331032213300'))
  st_within
-------------
 t
```

9.5. Operators

9.5.1. @>

This topic describes the @> operator, which queries whether a grid contains a geometry.

Syntax

```
boolean @>(geomgrid grid, geometry geom);
boolean @>(geometry geom, geomgrid grid);
boolean @>(geomgrid grid1, geomgrid grid2);
```

Parameters

Parameter	Description
grid	The grid whose spatial relationship you want to query.
geom	The geometry whose spatial relationship you want to query.

Description

This function returns the spatial relationship between a grid and a geometry. The geometry must use the CGC2000 spatial reference system. In addition, the spatial reference system identifier (SRID) of the geometry must be 4490.

Examples

```
select ST_gridfromtext('G001331032213300013') @>
      ST_geomfromtext('POINT(116.31522216796875 39.910277777777778)',4490) as t;
 t
---------------
 f
select ST_gridfromtext('G00133103221330') @>
       ST_gridfromtext('G001331032213300013') as t
 t
-------------
 t
```

9.5.2. <@

This topic describes the <@ operator, which queries whether a grid is contained in a geometry.

Syntax

```
boolean <@(geomgrid grid, geometry geom);
boolean <@(geometry geom, geomgrid grid);
boolean <@(geomgrid grid1, geomgrid grid2);
```

Parameters

Parameter	Description
grid	The grid whose spatial relationship you want to query.
geom	The geometry whose spatial relationship you want to query.

Description

This function returns the spatial relationship between a grid and a geometry. The geometry must use the CGC2000 spatial reference system. In addition, the spatial reference system identifier (SRID) of the geometry must be 4490.

Examples

```
select ST_geomfromtext('POINT(116.31522216796875 39.910277777777778)',4490) <@
  ST_gridfromtext('G001331032213300013') as t;
 t
---------------
 f
select ST_gridfromtext('G001331032213300013') <@
        ST_gridfromtext('G001331032213300') as t
 t
-------------
 t
```

9.6. Functions to compute grids

9.6.1. ST_AsGrid

This topic describes the ST_AsGrid function, which queries the grids that intersect with a geometry.

Syntax

```
geomgrid[] ST_AsGrid(geometry geom, integer precision);
```

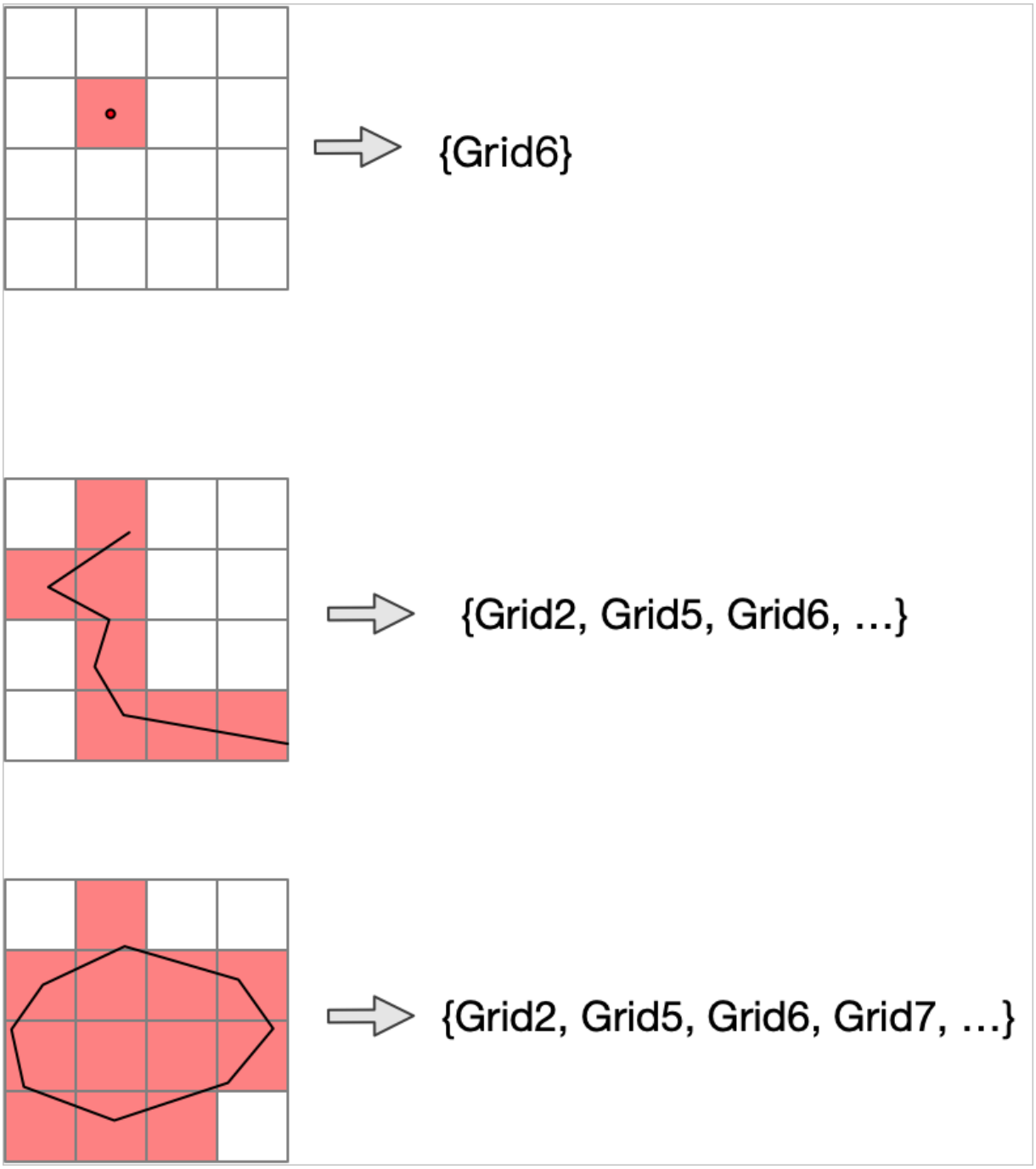
Parameters

Parameter	Description
geom	The geometry that you want to query.
precision	The precision level based on which you want to query the geometry. Valid values: 0 to 31.

Description

The geometry must use the CGC2000 spatial reference system. In addition, the spatial reference system identifier (SRID) of the geometry must be 4490. If the geometry does not use the CGC2000 spatial reference system, the ST_Transform function is invoked to convert the coordinates of the geometry into CGC2000 coordinates.

This function returns an array that consists of geometry-represented grids that intersect. The following figures are examples of how grids intersect with a point, a line, and a polygon.



Examples

```
select st_astext(st_asgrid(
    ST_geomfromtext('POINT(116.31522216796875 39.910277777777778)',4490), 15))
     st_astext
--------------------
 {G001310322230230}
select st_astext(st_asgrid(
    ST_geomfromtext('LINESTRING(122.48077 51.72814,122.47416 51.73714)',4490), 18))
                  st_astext

--------------------------------------------------------------------------------
 {G001331032213300011,G001331032213300013,G001331032213122320,G00133103221312232
2,G001331032213300100,G001331032213122303,G001331032213122321,G00133103221312231
2}
```

10.Geometry Pyramid SQL reference

10.1. Usage notes

Ganos Geometry Pyramid is used to display two-dimensional geometry big data at high speeds.

Data requirements

- Data tables can contain only the following types of spatial data: Point, Line, Polygon, MultiPoint, MultiLine, and MultiPolygon.
- Each data record must contain an ID field whose value is a unique positive integer of the uint4 or uint8 data type.
- You must create spatial indexes on the geometry fields.
- Geometry data supports all `EPSG` coordinate systems. MTP-formatted geometry data supports only the `EPSG:3857 and EPSG:4326` coordinate systems.

Hardware requirements

- We recommended that you configure 32 cores, 64 GB of memory, and 500 GB of SSD storage capacity. This allows you to process 0.1 billion data records within about 15 minutes.
- In most cases, the extra storage space that is occupied by the produced pyramids does not exceed 10% of the storage space that is occupied by the original data.

10.2. Functions to build pyramids

10.2.1. ST_BuildPyramid

This topic describes the ST_BuildPyramid function, which creates a vector pyramid for a spatial geometry data table to accelerate the display of data.

Syntax

```
boolean ST_BuildPyramid(cstring table, cstring geom, cstring fid, cstring config)
```

Parameters

Parameter	Description
table	The name of the spatial geometry data table.
geom	The name of the geometry field.
fid	The name of the element ID field.
config	The fields that are used to create a pyramid. These fields are specified as a JSON string.

The following table describes the fields in the config parameter.

Field	Type	Default value	Description
name	string	Same as the table name	The name of the pyramid.
parallel	int	0	The maximum number of tasks that can run in parallel to create the pyramid. You must also specify the max_prepared_transactions parameter. If you set the parallel parameter to 0, the maximum number of parallel tasks is not limited.
tileSize	int	1024	The tile size of the pyramid. Valid values must be greater than 0 and less than 4096.
tileExtend	int	8	The tile extension size of the pyramid. Valid values must be greater than 0.
userExtent	array[double]	null	The geographic bound that you define. The value of this parameter is a JSON string. This JSON string consists of four fields: minx, miny, maxx, and maxy. If you set this parameter to [], the value is null.
splitSize	int	10000	The maximum number of elements into which an index node can be split. A larger value of this parameter indicates a sparser pyramid.
maxLevel	int	16	The maximum number of layers in the pyramid. Valid values: 0 to 20.
sourceSRS	int	-1	The coordinate system of the source data in the spatial geometry data table. If you do not specify this parameter, the system reads the spatial reference identifiers (SRIDs) in the metadata.
destSRS	int	3857	The EPSG code of the title coordinate system. Only EPSG 3857 and EPSG 4326 are supported.
buildRules	array[object]	null	The rule based on which you want to create the pyramid. You can specify multiple rules. Each rule consists of two parts: level and value.
└level	array[int]	None	The pyramid layers to which the specified rule is applied.

Field	Type	Default value	Description
└value	object	None	The value of the specified rule.
└filter	string	None	The expression that is used to filter data in PostgreSQL.
└attrFields	array[string]	None	The name of the attribute field when the MVT format is used.
└merge	array[string]	None	The filter conditions that are used to add data records to a group.

The following example shows how to set the config parameter:

```
{
   "name" : "hello",
   "parallel": 4,
   "tileSize": 512,
   "tileExtend": 8,
   "userExtent": [-180,-90,180,90],
   "splitSize": 5000,
   "maxLevel": 16,
   "destSRS": 3857,
   "buildRules": [
     {
       "level": [0,1,2],
       "value": {
         "filter": "code!=0",
         "attrFields": ["name","color"],
         "merge":["code=1"]
       }
     }
   ]
}
```

Examples

```
--Create a pyramid for a spatial geometric data table named roads.
select ST_BuildPyramid('roads', 'geom', 'id', '');
st_buildpyramid
----------
t
```

10.3. Functions to delete pyramids

10.3.1. ST_DeletePyramid

This topic describes the ST_DeletePyramid function, which deletes a vector pyramid.

Syntax

```
boolean ST_DeletePyramid(cstring  name)
```

Parameters

Parameter	Description
name	The name of the pyramid that you want to delete.

Examples

```
--Delete a pyramid.
select ST_DeletePyramid('roads');
st_deletepyramid
----------
t
```

10.4. Functions to view pyramids

10.4.1. ST_Tile

This topic describes the ST_Tile function, which generates a MVT-formatted standard binary structure based on a tile ID from a pyramid.

Syntax

```
bytea ST_Tile(cstring name, cstring key);
bytea ST_Tile(cstring name, int x, int y, int z);
```

Parameters

Parameter	Description
name	The name of the pyramid.
key	The ID of the tile.
x	The x value in the tile ID.
y	The y value in the tile ID.
z	The z value in the tile ID.

Description

This function returns a MVT-formatted standard binary structure that is generated based on a tile ID from a pyramid. The tile ID follows the `'z_x_y'` format and the `EPSG:3857` geographic coordinate system.

Examples

```
select ST_Tile('roads', '3_1_6');
st_tile
----------
0xFFAABB8D8A6678...
select ST_Tile('roads', 1, 6, 3);
st_tile
----------
0xFFAABB8D8A6678...
```

10.4.2. ST_AsPng

This topic describes the ST_AsPng function, which converts a specific tile of a pyramid to a PNG image based on the ID of the tile.

Syntax

```
bytea ST_AsPng(cstring name, cstring key, cstring style);
bytea ST_AsPng(cstring name, int x, int y, int z, cstring style);
```

Parameters

Parameter	Description
name	The name of the pyramid.
key	The ID of the tile.
x	The x coordinate in the ID of the tile.
y	The y coordinate in the ID of the tile.
z	The z coordinate in the ID of the tile.
style	The style that is used to render the PNG image. The value of this parameter is a JSON string.

The following table describes the fields in the style parameter.

Field	Type	Default value	Description
background	string	#FFFFFFFF	The RGBA color of the background in the PNG image. The default color is white.
line_color	string	#000000FF	The RGBA color of dots and edges in the PNG image. The default color is black.

Field	Type	Default value	Description
fill_color	string	#F4A460FF	The RGBA color of the tile in the PNG image. The default color is brown.
line_width	int	1	The width per line in the PNG image. Unit: pixels.
point_size	int	10	The size per dot in the PNG image. Unit: pixels. By default, each dot is a circle with a diameter of 10 pixels.
parallel_unit	int	50000	The number of elements that each task running in parallel renders.

The following example shows the settings of the fields in the style parameter:

```
{
  "background": "#FFFFFFFF",
  "line_color": "#000000FF",
  "fill_color": "#F4A460FF",
  "line_width": 1,
  "point_size": 10
}
```

Description

The tile ID that is specified by the key parameter is in the `'z_x_y'` format in compliance with the `EPSG:4326` or `EPSG:3857` coordinate system.

> **Note**
>
> - A title consists of multiple chunks. If the `EPSG:4326` coordinate system is used, the number of chunks on the x axis is twice the number of chunks on the y axis. The z coordinate starts from 1. If the z coordinate is 1, the tile ID can only be `1_0_0` or `1_1_0`.
> - The size of the PNG image that is rendered based on the style parameter is equal to the value of the tileSize parameter that is specified for the pyramid. If you do not specify the style parameter, the default style is used.

Examples

```
select ST_AsPng('roads', '3_1_6', '');
st_aspng
----------
0xFFAABB8D8A6678...
select ST_AsPng('roads', 1, 6, 3, '');
st_aspng
----------
0xFFAABB8D8A6678...
```

11.Map services

11.1. Overview

GeoServer is a Java 2 Platform, Enterprise Edition (J2EE)-based software server in compliance with OpenGIS standards. It allows you to publish, update, delete, and insert geospatial data. You can share spatial information by using GeoServer.

GeoServer conforms to both the Web Map Service (WMS) and Web Feature Service (WFS) standards set forth by Open Geospatial Consortium (OGC). It supports PostgreSQL, Shapefile, ArcSDE, Oracle, VPF, MySQL, MapInfo, and projections in hundreds of formats. GeoServer can produce maps in various formats, such as JPEG, GIF, PNG, SVG, and KML. GeoServer can run in any Java-based or Servlet-based container. It is integrated with OpenLayers, a mapping library that allows you to use asynchronous JavaScript and XML (AJAX) requests to create maps in Map Builder. Additional features are provided for you to explore.

Download and install GeoServer

- GeoServer 2.13.X and GeoTools 19.X
- GeoServer 2.14.X and GeoTools 20.X
- GeoServer 2.15.X and GeoTools 21.X
- GeoServer 2.16.X and GeoTools 22.X
- GeoServer 2.17.X and GeoTools 23.X

> **Note** We recommend that you download GeoServer 2.14.2 or later.

Decompress the GeoServer package to the geoserver/WEB-INFO/lib directory and then start the container.

11.2. Publish geometry data

This topic describes how to publish geometry data in a Ganos database by using GeoServer.

Add a data store

1. Run GeoServer. In the left-side navigation pane, click **Stores**.
2. On the Stores page, click **Add new Store**. On the page that appears, select **PostGIS** in the Vector Data Sources section.
3. Set the connection information about the database.

Use the RESTful API

After publishing geometry data, you can use the RESTful API provided by GeoServer to access the data. For more formation, see GeoServer User Manual.

11.3. Publish raster data

This topic describes how to publish raster data in an ApsaraDB RDS for PostgreSQL or ApsaraDB RDS for PolarDB instance that is powered by Ganos extensions by using GeoServer.

Add a data source

1. Run GeoServer. In the left-side navigation pane, click **Stores**.
2. On the Stores page, click **Add New Store**. In the Raster Data Sources section of the dialog box that appears, select **GanosRaster(PG/PolarDB)**.
3. Configure the connection parameters of the RDS instance.

   The following table describes the parameters.

Parameter	Description	Example
host	The IP address or the endpoint of the RDS instance.	xxxxxxx.pg.rds.aliyuncs.com
port	The port number of the RDS instance.	3432
database	The name of the RDS instance.	rasterdb
username	The username that is used to log on to the RDS instance.	pguser
password	The password that is used to log on to the RDS instance.	123456
schema	The schema of the table.	Default value: public.
table	The name of the table that stores the raster data.	raster_table
column name	The name of the raster column in the table.	raster_column
filter	The condition that is specified by the WHERE clause in the SQL statement to filter raster data. If more than one rasters meet the condition, the first raster is used.	id=1 or name='srtm'
name	The name of the raster that is displayed in GeoServer.	myraster

REST API

- Create a data source.
  - Request URL: http://host:port/geoserver/rest/workspaces/{workspace}/coveragestores
  - Method: POST

- Parameter description:
  - workspace: the name of the created workspace.
  - datastore body: the information about the data store. Keep the default value of the type parameter, and set the url parameter to the connection information of the ApsaraDB RDS for PostgreSQL or ApsaraDB for PolarDB instance in JSON format. The following code snippet provides an example:

```
{
  "coverageStore": {
    "name": "<datasource_name>",
    "type": "GanosRaster(PG/PolarDB)",
    "enabled":"true",
    "workspace":"<wokrspace>",
    "url": "{\"column\":\"<raster_colunm>\",\"database\":\"<database_name>\",\"filter
\":\"<raster_filter>\",\"host\":\"<pg_host>\",\"name\":\"<public_name>\",\"password\"
:\"<user_password>\",\"port\":<pg_port>,\"schema\":\"<schema_name>\",\"ssl\":false,\"
table\":\"<raster_table_name>\",\"userName\":\"<user_name>\",\"valid\":true}"
  }
}
```

- Example:

```
{
  "coverageStore": {
    "name": "srtm",
    "type": "GanosRaster(PG/PolarDB)",
    "enabled":"true",
    "workspace":"test",
    "url": "{\"column\":\"rast\",\"database\":\"test_db\",\"filter\":\"name='srtm'\",\"
host\":\"pgm-xxxxxxxx.pg.rds.aliyuncs.com\",\"name\":\"srtm_image\",\"password\":\"xxxx
\",\"port\":3432,\"schema\":\"public\",\"ssl\":true,\"table\":\"raster_table\",\"userNa
me\":\"raster_user\",\"valid\":true}"
  }
}
```

- Obtain the data source.
  - Request URL: http://host:port/geoserver/rest/workspaces/{workspace}/coveragestores
  - Method: GET

- Parameter description:
  - workspace: the name of the created workspace.
  - datastore body: the information about the data store. Keep the default value of the type parameter, and set the url parameter to the connection information about the RDS for PostgreSQL or RDS for PolarDB instance in JSON format. The following code snippet provides an example:

```
{
  "dataStores": {
    "dataStore": [
      {
        "name": "txxxxx_shp_filter",
        "href": "http://11.xxx.xxx.xxx:8080/geoserver/rest/workspaces/tianxun/datasto
res/txxxxx_shp_filter.json"
      },
      {
        "name": "yxxxxx_shape1",
        "href": "http://11.xxx.xxx.xxx:8080/geoserver/rest/workspaces/tianxun/datasto
res/yxxxxx_shape1.json"
      },
      {
        "name": "yxxxxxn_shape2",
        "href": "http://11.xxx.xxx.xxx:8080/geoserver/rest/workspaces/tianxun/datasto
res/yxxxxxn_shape2.json"
      }
    ]
  }
}
```

- Publish a raster layer.
  - Request URL: http://host:port/geoserver/rest/workspaces/{workspace}/coveragestores/{store}/coverages
  - Method: POST
  - Parameter description:
    - workspace: the name of the created workspace.
    - The following code snippet provides an example of the POST request body:

```
{
  "coverage": {
    "abstract": "Digital elevation model for the Spearfish region.\r\n\r\nsfdem is a
Tagged Image File Format with Geographic information",
    "defaultInterpolationMethod": "nearest neighbor",
    "description": "Generated from sfdem",
    "dimensions": {
      "coverageDimension": [
        {
          "description": "GridSampleDimension[-9.999999933815813E36,-9.99999993381581
3E36]",
          "name": "GRAY_INDEX",
          "range": {
            "max": -9.999999933815813e+36,
            "min": -9.999999933815813e+36
```

```
          }
        }
      ]
    },
    "enabled": true,
    "grid": {
      "@dimension": "2",
      "crs": "EPSG:26713",
      "range": {
        "high": "634 477",
        "low": "0 0"
      },
      "transform": {
        "scaleX": 30,
        "scaleY": -30,
        "shearX": 0,
        "shearY": 0,
        "translateX": 589995,
        "translateY": 4927995
      }
    },
    "interpolationMethods": {
      "string": [
        "nearest neighbor",
        "bilinear",
        "bicubic"
      ]
    },
    "keywords": {
      "string": [
        "WCS",
        "sfdem",
        "sfdem",
        "type\\@language=fr\\;\\@vocabulary=test\\;"
      ]
    },
    "latLonBoundingBox": {
      "crs": "EPSG:4326",
      "maxx": -103.62940739432703,
      "maxy": 44.5016011535299,
      "minx": -103.87108701853181,
      "miny": 44.370187074132616
    },
    "metadata": {
      "entry": [
        {
          "@key": "elevation",
          "dimensionInfo": {
            "enabled": false
          }
        },
        {
          "$": "10",
          "@key": "cacheAgeMax"
```

```
        },
        {
          "@key": "time",
          "dimensionInfo": {
            "defaultValue": "",
            "enabled": false
          }
        },
        {
          "$": "true",
          "@key": "cachingEnabled"
        },
        {
          "$": "sfdem_sfdem",
          "@key": "dirName"
        }
      ]
    },
    "name": "sfdem",
    "namespace": {
      "href": "http://localhost:8075/geoserver/restng/namespaces/sf.json",
      "name": "sf"
    },
    "nativeBoundingBox": {
      "crs": {
        "$": "EPSG:26713",
        "@class": "projected"
      },
      "maxx": 609000,
      "maxy": 4928010,
      "minx": 589980,
      "miny": 4913700
    },
    "nativeCRS": {
      "$": "PROJCS[\"NAD27 / UTM zone 13N\", \n  GEOGCS[\"NAD27\", \n    DATUM[\"Nort
h American Datum 1927\", \n      SPHEROID[\"Clarke 1866\", 6378206.4, 294.97869821389
82, AUTHORITY[\"EPSG\",\"7008\"]], \n      TOWGS84[2.478, 149.752, 197.726, 0.526, -0
.498, 0.501, 0.685], \n      AUTHORITY[\"EPSG\",\"6267\"]], \n    PRIMEM[\"Greenwich\
", 0.0, AUTHORITY[\"EPSG\",\"8901\"]], \n    UNIT[\"degree\", 0.017453292519943295],
\n    AXIS[\"Geodetic longitude\", EAST], \n    AXIS[\"Geodetic latitude\", NORTH], \
n    AUTHORITY[\"EPSG\",\"4267\"]], \n  PROJECTION[\"Transverse_Mercator\", AUTHORITY
[\"EPSG\",\"9807\"]], \n  PARAMETER[\"central_meridian\", -105.0], \n  PARAMETER[\"la
titude_of_origin\", 0.0], \n  PARAMETER[\"scale_factor\", 0.9996], \n  PARAMETER[\"fa
lse_easting\", 500000.0], \n  PARAMETER[\"false_northing\", 0.0], \n  UNIT[\"m\", 1.0
], \n  AXIS[\"Easting\", EAST], \n  AXIS[\"Northing\", NORTH], \n  AUTHORITY[\"EPSG\"
,\"26713\"]]",
      "@class": "projected"
    },
    "nativeFormat": "GeoTIFF",
    "nativeName": "sfdem",
    "requestSRS": {
      "string": [
        "EPSG:26713"
      ]
```

```
    },
    "responseSRS": {
      "string": [
        "EPSG:26713"
      ]
    },
    "srs": "EPSG:26713",
    "store": {
      "@class": "coverageStore",
      "href": "http://localhost:8075/geoserver/restng/workspaces/sf/coveragestores/sf
dem.json",
      "name": "sf:sfdem"
    },
    "supportedFormats": {
      "string": [
        "ARCGRID",
        "IMAGEMOSAIC",
        "GTOPO30",
        "GEOTIFF",
        "GIF",
        "PNG",
        "JPEG",
        "TIFF"
      ]
    },
    "title": "Spearfish elevation"
  }
}
```

12.Desktop applications

12.1. Use OpenJump to access Ganos data

OpenJUMP is an open-source geographic information system (GIS) application written in Java. It provides the built-in map visualization feature for spatial analysis, such as buffer analysis, intersection computing, and union computing. In addition, its plug-in system allows you to customize the functionality as needed.

Add a connection to a Ganos database

1. Run OpenJUMP. In the top navigation bar, choose **File > Open**. In the dialog box that appears, click **Data Store Layer** in the left-side navigation pane.
2. In the **Connection Manager** dialog box that appears, click **Add**.
3. In the **Add Connection** dialog box that appears, set connection parameters.

   The following table describes the parameters.

Parameter	Description
Name	The custom name of the connection.
Driver	The driver of the connection. Set this parameter to **PostGIS**.
Server	The IP address of the database or the public network endpoint of the ApsaraDB for Relational Database Service (RDS) or PolarDB instance. You can obtain the endpoint from the Alibaba Cloud console.
Port	The port number of the database. You can obtain the port number from the Alibaba Cloud console.
Database	The name of the database to be connected.
User	The username used to log on to the database.
Password	The password used to log on to the database.

4. After setting the parameters, click **OK**. All layers in the connected Ganos database appear.

View and manage the geospatial data

After OpenJUMP is connected to the Ganos database, you can view and manage the geospatial data. Supported operations include zooming in, zooming out, walkthrough, and symbolic rendering.

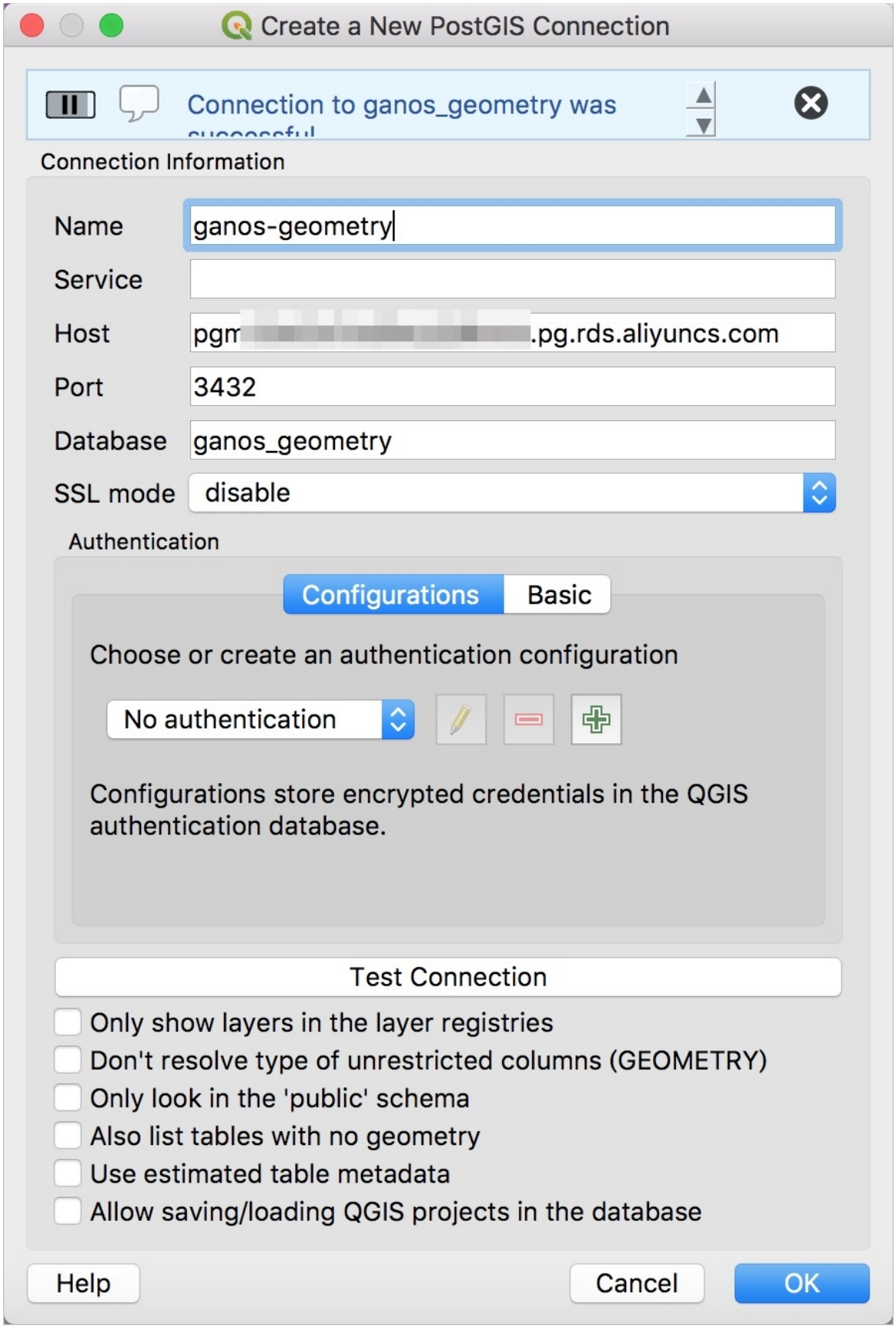
You can also query the attributes of a geometry object.

12.2. Use QGIS to access Ganos data

Quantum GIS (QGIS) is a user-friendly open-source desktop geographic information system (GIS) application. It supports multiple formats of raster images, vector images, and databases as data sources. Using QGIS, you can view, edit, manage, and analyze geospatial data, and create and export maps.

Add a connection to a Ganos database

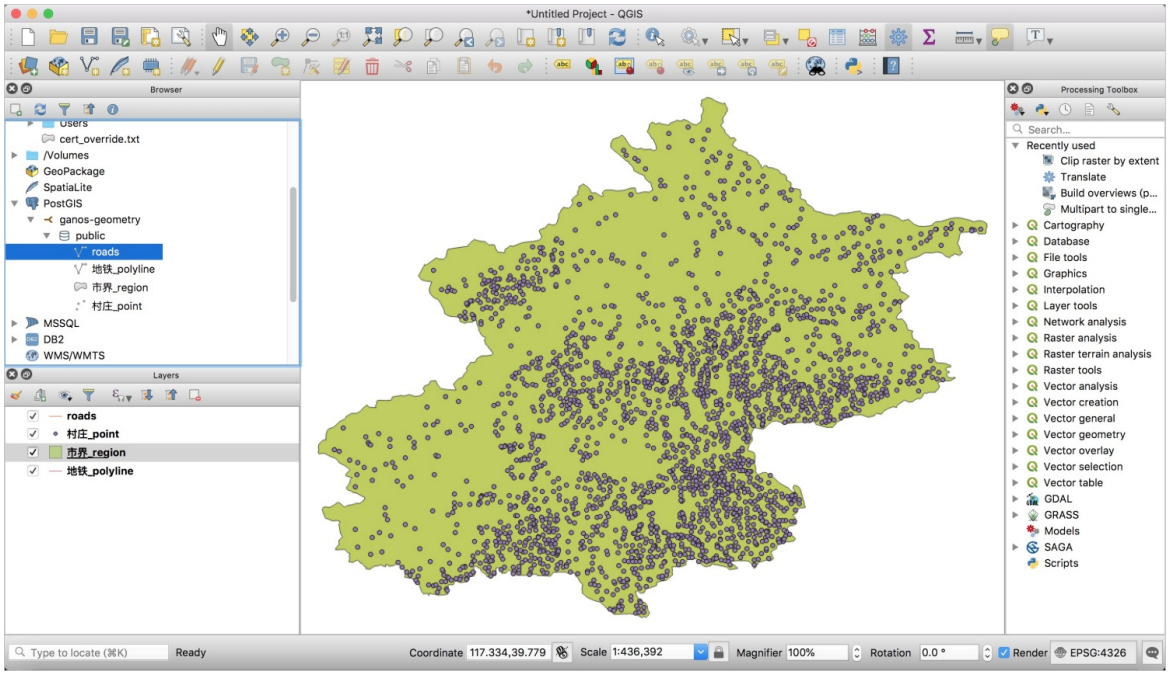
1. Run QGIS. In the Create a New PostGIS Connection dialog box, set connection parameters as required and click **Test Connection**.



The following table describes the parameters.

Parameter	Description
Name	The custom name of the connection.
Host	The IP address of the database or the public network endpoint of the ApsaraDB for Relational Database Service (RDS) or PolarDB instance. You can obtain the endpoint from the Alibaba Cloud console.
Port	The port number of the database. You can obtain the port number from the Alibaba Cloud console.
Database	The name of the database to be connected.
SSL mode	The Secure Sockets Layer (SSL) encryption mode. Set this parameter to **disable**.

2. Select a connected Ganos database. You can view the existing database and layers under the database. Double-click a layer to browse, view, and manage the geospatial data in the database.

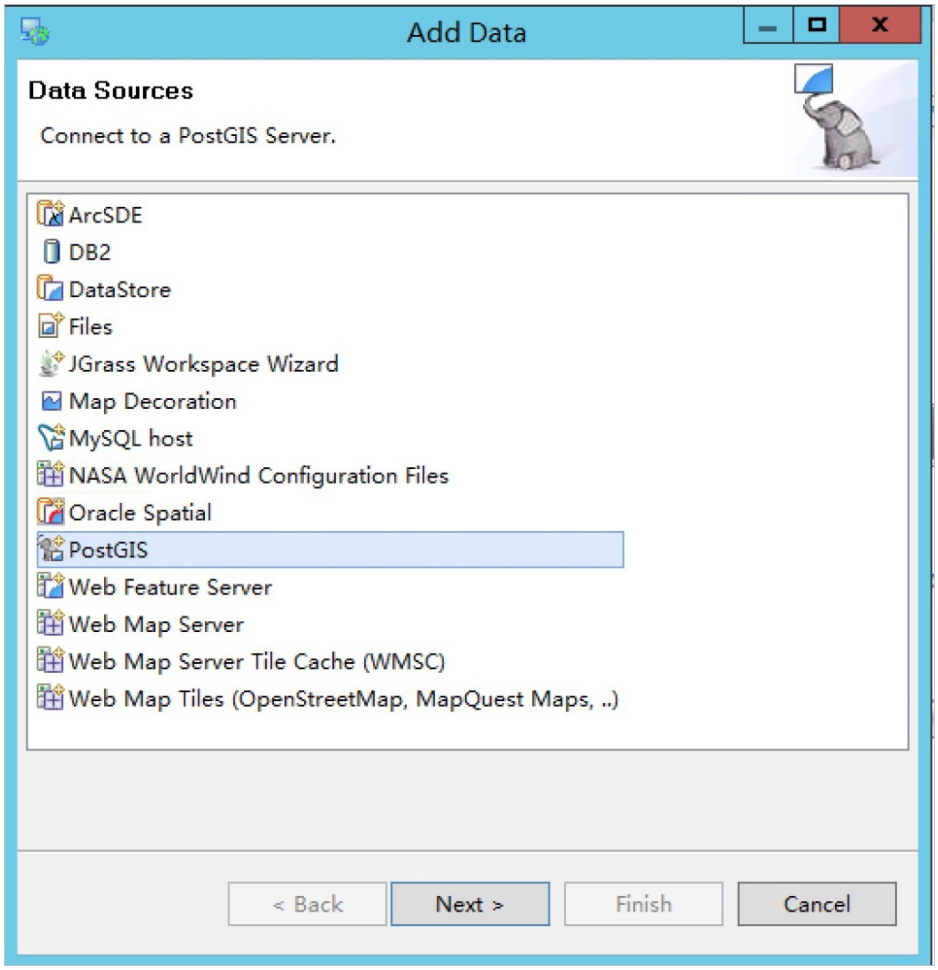


12.3. Use uDig to access Ganos data

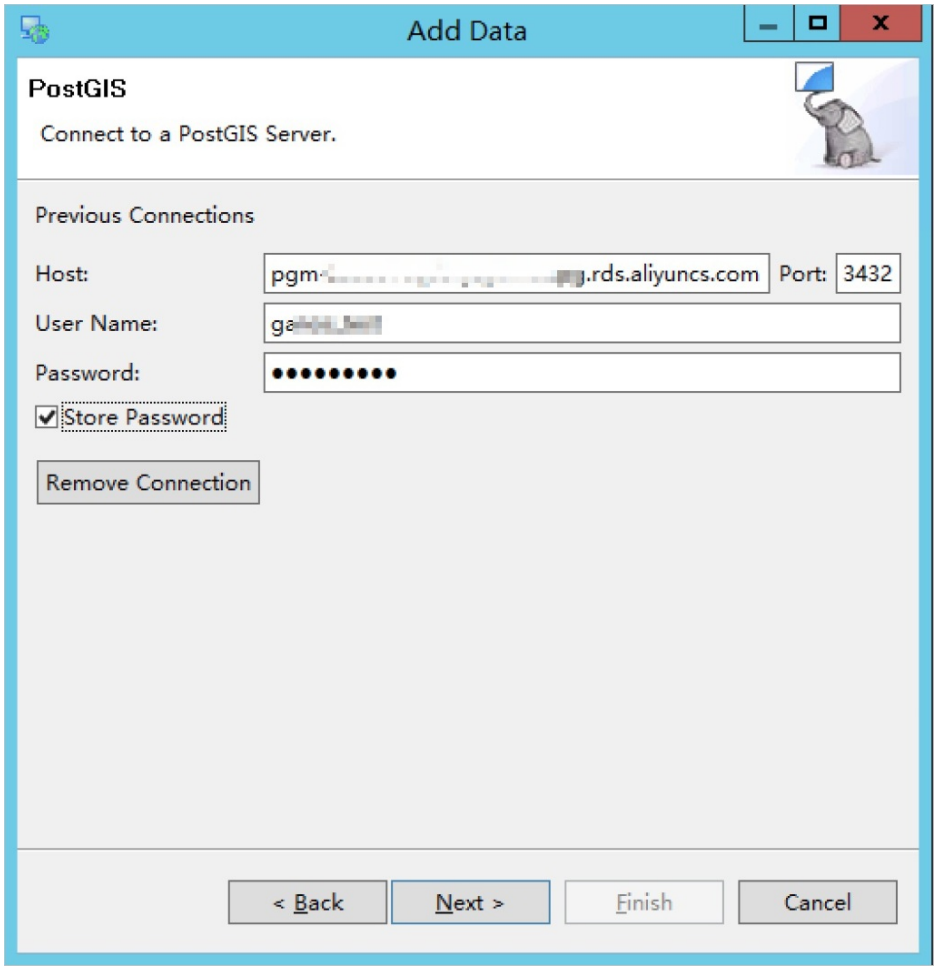
uDig is an open-source desktop geographic information system (GIS) application for editing and viewing geospatial data, such as a map file in the shapefile format. It supports OpenGIS standards and provides advanced Internet GIS, Web Map Service (WMS), and web servers. Based on Java, uDig allows you to use open-source plug-ins to view and manage geospatial data.

Add a connection to a Ganos database

1. Run uDig. In the top navigation bar, choose **Layer > Add Data**. In the dialog box that appears, select PostGIS and click Next.



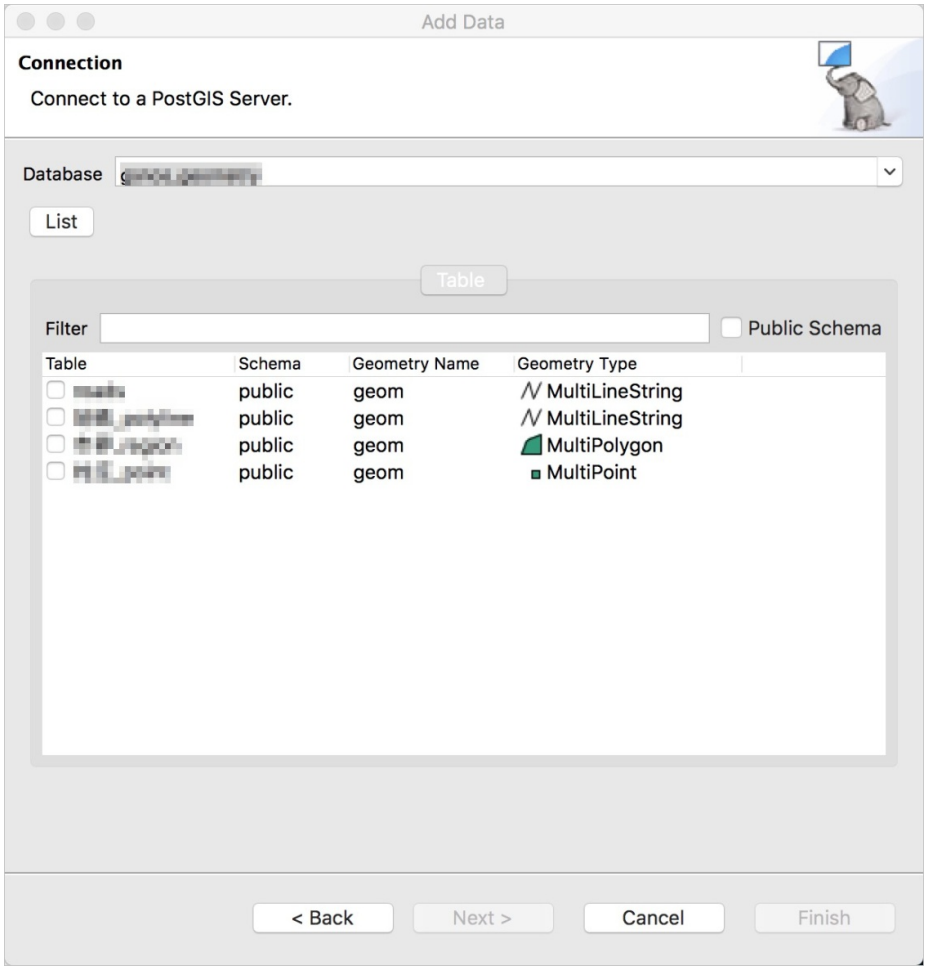
2. Set the connection information about the database.



The following table describes the parameters.

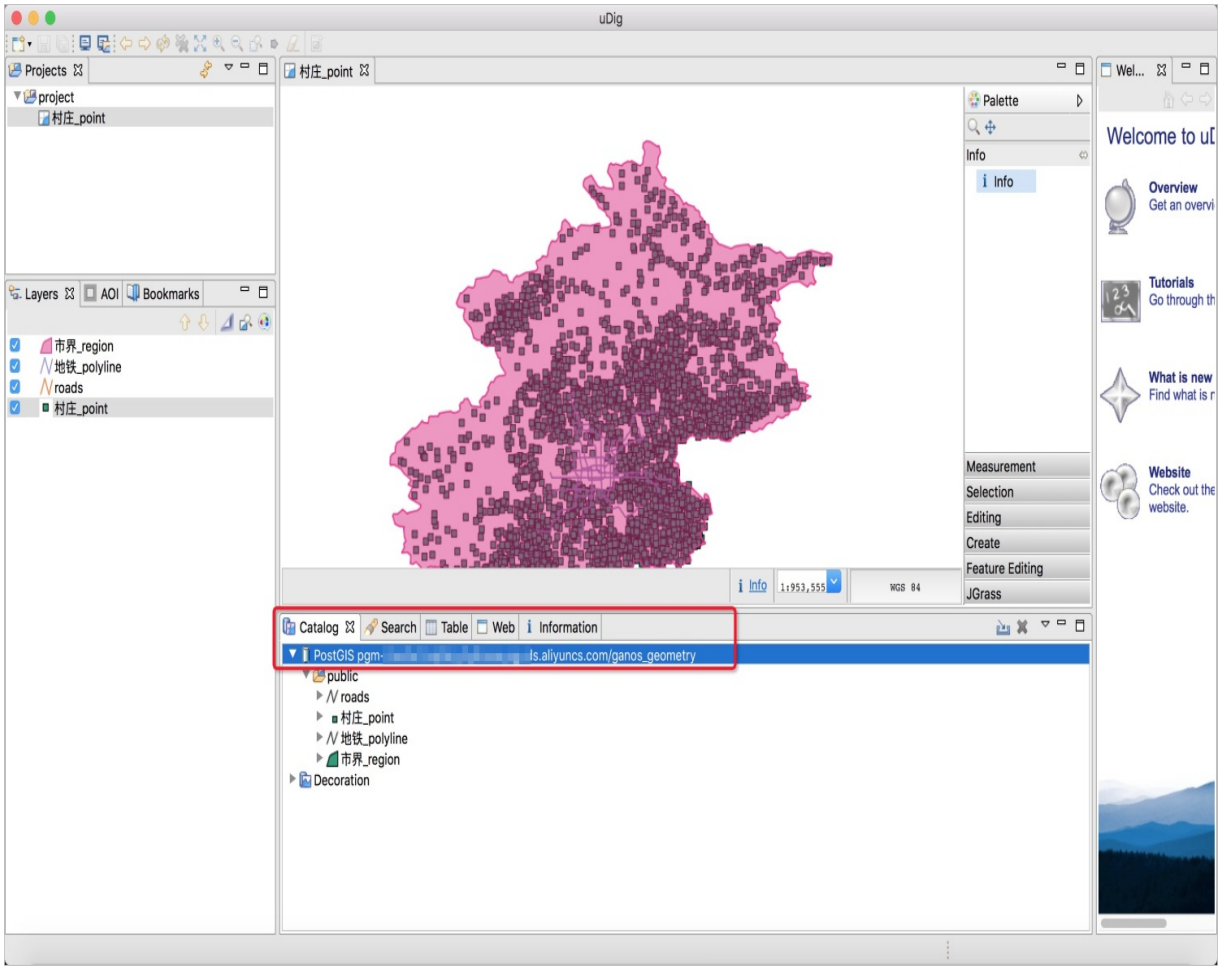
Parameter	Description
Host	The IP address of the database or the public network endpoint of the ApsaraDB for Relational Database Service (RDS) or PolarDB instance. You can obtain the endpoint from the Alibaba Cloud console.
Port	The port number of the database. You can obtain the port number from the Alibaba Cloud console.
User Name	The username used to log on to the database.
Password	The password used to log on to the database.

3. Select a database from the Database drop-down list.



View and manage the geospatial data

After uDig is connected to the Ganos database, you can browse, view, and manage the geospatial data in the Ganos database.


uDig
Projects
project
村庄_point
Layers
AOI
Bookmarks
市界_region
地铁_polyline
roads
村庄_point
Palette
Info
Measurement
Selection
Editing
Create
Feature Editing
JGrass
Catalog
Search
Table
Web
Information
PostGIS pgm-
ls.aliyuncs.com/ganos_geometry
public
roads
村庄_point
地铁_polyline
市界_region
Decoration
Welcome to uD
Overview
Tutorials
What is new
Website

13.FAQ

13.1. Load raster data

This topic describes how to load a raster data file from your computer or an Alibaba Cloud Object Storage Service (OSS) to Ganos by using the ST_ImportFrom function.

Precautions

Your RDS instance is authorized to access the raster data file. If the raster data file is stored in an OSS bucket, your RDS instance resides in the same region as the OSS bucket. For more information, see OSS domain names.

Introduction to the ST_ImportFrom function

The ST_ImportFrom function is used to load a raster data file from your computer or an OSS bucket to Ganos.

> **Note**
> - In addition to the ST_ImportFrom function, Ganos provides other raster-related functions. For more information, see Raster SQL Reference.
> - For more information about the parameters in the ST_ImportFrom function, see ST_ImportFrom.

Syntax

```
INSERT INTO <raster_table_name>(<raster_column_name>) VALUES (ST_IMPORTFROM('<chunk_table_name>','<path_to_raster_file>'))
```

Examples

1. Create the ganos_raster extension.

   ```
   CREATE EXTENSION Ganos_Raster CASCADE
   ```

2. Load a raster data file from your computer to Ganos.

   ```
   INSERT INTO raster_table(name, rast) VALUES ('local_file', ST_ImportFrom('chunk_table', '/home/beijing.tif'));
   ```

3. Load a raster data file from an OSS bucket to Ganos.

   ```
   INSERT INTO raster_table(name, rast) VALUES ('oss_file', ST_ImportFrom('chunk_table', 'oss://ak:as@bucket/data/beijing.tif'));
   ```

13.2. Load vector data

This topic describes how to load a vector data file into Ganos by using shp2pgsql, ogr2ogr, or QGIS.

Preparations

Before you load a vector data file, run the following command on your AnalyticDB for PostgreSQL instance to create the ganos_geometry extension:

```
CREATE EXTENSION ganos_geometry CASCADE
```

Load vector data by using shp2pgsql

The shp2pgsql command line tool is used to convert Esri shapefiles into SQL files. For more information, visit shp2pgsql. Then, you can load the SQL files into Ganos.

- Syntax

```
shp2pgsql -s <srid> -c -W <charset> <path_to_shpfile> <schema>.<table_name> | psql -d <db
name> -h <host> -U <user_name> -p <port>
```

- Parameters

Parameter	Description	Example
-s <srid>	The spatial reference system identifier (SRID) of the vector data file.	4490
-c	Specifies to create a table. You can also use the following parameters: ○ -a: appends data to a table. ○ -d: deletes a table before data is loaded. ○ -p: creates a schema in SQL without writing spatial data.	None
-W <charset>	The character set that is used by the shapefile.	GBK or UTF8
<path_to_shpfile>	The storage path of the shapefile. The storage path must be accessible to the shp2pgsql command line tool.	/home/roads.shp
<schema>.<table_name>	The name of the geometry table that you created.	public.roads
-d <dbname>	The name of the database.	mydb
-h <host>	The endpoint of the instance on which to create the database.	pgm-xxxxxxx.pg.rds.aliyuncs.com
-U <user_name>	The username that is used to connect to the database.	my_name
-p <port>	The port number that is used to connect to the database.	3433

- Example

```
shp2pgsql -s 4490 -c -W "GBK" /home/roads.shp public.roads | psql -d mydb -h pgm-xxxxxxx.
pg.rds.aliyuncs.com -U my_name -p 3433
```

Note If you want to import more than one vector data file at a time by using a script, run the `export PGPASSWORD=my_pass` command to set the PGPASSWORD environment variable to my_pass.

Load vector data by using ogr2ogr

The ogr2ogr command line tool is provided by GDAL/OGR to convert data. It supports common vector data types, such as Esri ShapFile, MapInfo, and FileGDB. For more information about ogr2ogr, visit ogr2ogr. For more information anout vector data types, visit Vector drivers.

Note Before you load vector data by using the ogr2ogr command line tool, make sure that GDAL supports the PostGIS vector driver.

- Syntax

```
ogr2ogr -nln <table_name> -f PostgreSQL PG:"dbname='<dbname>' host='<host>' port='<port>'
user='<user_name>' password='<password>'" -lco SPATIAL_INDEX=GIST -lco FID=<FID_NAME> -ov
erwrite "<PATH_TO_FILE>" -nlt GEOMETRY
```

- Parameters

Parameter	Description	Example
-nln <table_name>	The name of the vector data file.	roads
-f PostgreSQL PG:"dbname='<dbname>' host='<host>' port='<port>' user='<user_name>' password='<password>'"	The information that is used to connect to the AnalyticDB for PostgreSQL instance.	dbname='mydb' host='pgm-xxxxxxx.pg.rds.aliyuncs.com' port='3433' user='my_name' password='my_pass'
-lco SPATIAL_INDEX=GIST	The GiST spatial index that you want to create.	None
-lco FID=<FID_NAME>	The ID of the specified feature.	fid
-overwrite	The rewrite mode that you want to use. You can also select the -append mode.	None
<PATH_TO_FILE>	The storage path of the vector data file. Make sure that the storage path is accessible to the ogr2ogr command line tool.	/home/roads.shp /home/land.tab

Parameter	Description	Example
-nlt GEOMETRY	Specifies the geometry data type. If the vector data file contains polygon data, we recommend that you specify this parameter. This allows you to avoid errors that occur if the vector data file contains not only data of the MultiPolygon type but also data of other types.	None

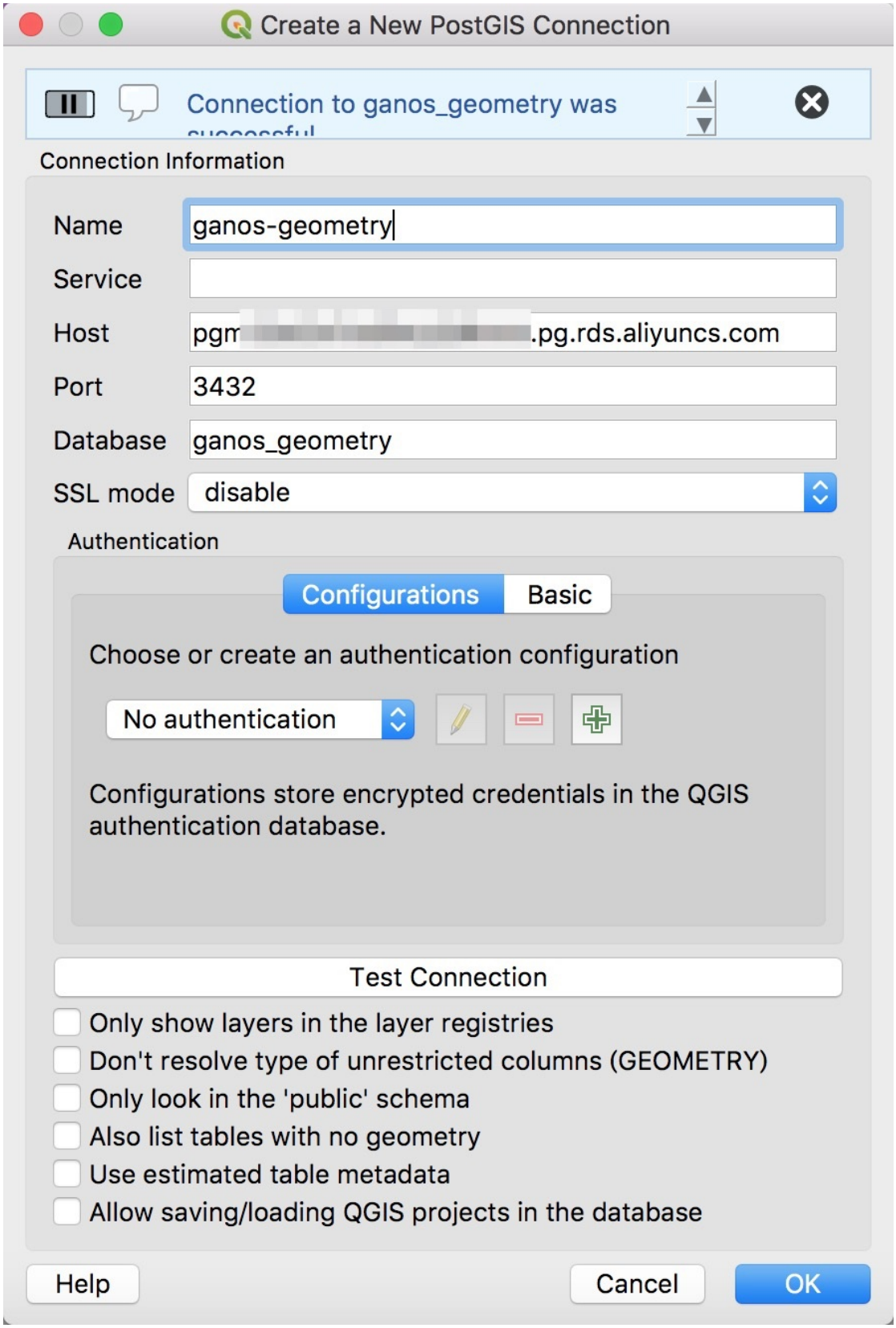
- Example

```
ogr2ogr -nln roads -f PostgreSQL PG:"dbname='mydb' host='pgm-xxxxxxx.pg.rds.aliyuncs.com'
port='3433' user='my_name' password='my_pass'" -lco SPATIAL_INDEX=GIST -lco FID=fid -over
write "/home/roads.shp" -nlt GEOMETRY
```

Load vector data by using QGIS

QGIS was known as Quantum GIS. It is a user-friendly, open-source desktop tool that supports various vector and raster formats. It allows you to use databases as data sources. In addition, it allows you to visualize, manage, and analyze data and to prepare maps. For more information, see QGIS.

1. Establish a connection to Ganos.
   i. In the right-side **Browser** pane, create a PostGIS connection.
   ii. Configure the required parameters.



The following table describes the parameters.

Parameter	Description
Name	The name of the connection.
Host	The IP address of your ECS instance or the public endpoint of your ApsaraDB RDS instance or PolarDB cluster. You can obtain this information from the Alibaba Cloud Management Console.
Port	The port number of your ECS instance, ApsaraDB RDS instance, or PolarDB cluster. You can obtain this information from the Alibaba Cloud Management Console.
Database	The name of the database.
SSL Mode	Specifies whether to establish an encrypted connection based on SSL.

iii. Click **Test Connection**. Then, wait until the connection is established.

2. Load the vector data file.
   i. In the menu bar, choose **Database > DB Manager**.
   ii. In the dialog box that appears, select the data source from which you want to import the vector data file, and click **Import Layer/File...**.
   iii. In the dialog box that appears, click the vector data file, configure the required parameters, and then click **OK**.

13.3. Trajectory FAQ

This topic provides answers to the commonly asked questions about trajectory. This helps you use the Ganos engine in AnalyticDB for PostgreSQL.

How do I convert the data of coordinate points into a trajectory object?

You can use the ST_makeTrajectory constructor to convert the data of coordinate points into a trajectory object. You can execute the following statements:

```
-- Create an extension.
create extension ganos_trajectory cascade;
-- Create a point table.
create table points(id integer, x float8, y float8, t timestamp, speed float8);
insert into points values(1, 128.1, 28.1, '2019-01-01 00:00:00', 100);
insert into points values(2, 128.2, 28.2, '2019-01-01 00:00:01', 101);
insert into points values(3, 128.3, 28.3, '2019-01-01 00:00:02', 102);
insert into points values(4, 128.4, 28.4, '2019-01-01 00:00:04', 103);
-- Create a trajectory table.
create table traj(id integer, traj trajectory);
-- Insert data.
insert into traj(id, traj)
select 1,
    ST_MakeTrajectory('STPOINT'::leaftype, x, y, 4326, t, ARRAY['speed'], NULL, s, NULL)
FROM (select array_agg(x order by id) as x,
      array_agg(y order by id) as y,
      array_agg(t order by id) as t,
      array_agg(speed order by id) as s
  From points) a;
```

What shall I do if the built-in ST_makeTrajectory constructor does not meet my requirements?

If the built-in ST_makeTrajectory constructor does not meet your requirements, you can customize one with required attributes. For example, to create a trajectory object with five attributes, including two of the int8 type, two of the float4 type, and one of the timestamp type, you can customize the following constructor:

```
CREATE OR REPLACE FUNCTION  ST_MakeTrajectory(type leaftype, x float8[], y float8[] ,
               srid integer, timespan timestamp[],attrs_name cstring[], attr1 int8[],
               attr2 int8[], attr3 float4[], attr4 float4[], attr5 timestamp[])
    RETURNS trajectory
    AS '$libdir/libpg-trajectory16','sqltr_traj_make_all_array'
    LANGUAGE 'c' IMMUTABLE Parallel SAFE;
```

In this constructor, the first six parameters are fixed, and the last five parameters are customized attributes. You can directly use this constructor with the customized attributes as a user-defined overloaded function.

How do I append trajectory points to a trajectory object?

You can use the ST_append constructor to append trajectory points to an existing trajectory object. The following statements describe the syntax of the ST_append constructor.

```
trajectory ST_append(trajectory traj, geometry spatial, timestamp[] timespan, text str_them
e_json) ;
trajectory ST_append(trajectory traj, trajectory tail) ;
```

The following example shows how to append trajectory points to a trajectory object based on a point table.

```
-- Create an extension.
```

```
create extension ganos_trajectory cascade;
-- Create a point table.
create table points(id integer, x float8, y float8, t timestamp, speed float8);
insert into points values(1, 128.1, 28.1, '2019-01-01 00:00:00', 100);
insert into points values(2, 128.2, 28.2, '2019-01-01 00:00:01', 101);
insert into points values(3, 128.3, 28.3, '2019-01-01 00:00:02', 102);
insert into points values(4, 128.4, 28.4, '2019-01-01 00:00:04', 103);
-- Create a trajectory table.
create table traj(id integer, traj trajectory);
-- Insert data.
insert into traj(id, traj)
select 1,
    ST_MakeTrajectory('STPOINT'::leaftype, x, y, 4326, t, ARRAY['speed'], NULL, s, NULL)
FROM (select array_agg(x order by id) as x,
      array_agg(y order by id) as y,
      array_agg(t order by id) as t,
      array_agg(speed order by id) as s
  From points) a;
-- Insert trajectory points.
insert into points values(5, 128.5, 28.5, '2019-01-01 00:00:05', 105);
insert into points values(6, 128.6, 28.6, '2019-01-01 00:00:06', 106);
insert into points values(7, 128.7, 28.7, '2019-01-01 00:00:07', 107);
-- Update the trajectory object based on a single trajectory point.
With point_traj as (
 select ST_MakeTrajectory('STPOINT'::leaftype, x, y, 4326, t, ARRAY['speed'], NULL, s, NULL
) AS traj
 FROM (select array_agg(x order by id) as x,
      array_agg(y order by id) as y,
      array_agg(t order by id) as t,
      array_agg(speed order by id) as s
      From points WHERE ID = 5) a
)
Update traj
set traj = ST_append(traj.traj, a.traj)
From point_traj a
WHERE traj.ID=1;
-- Use a generated SQL script to update the trajectory object.
With point_traj as (
 select ST_MakeTrajectory('STPOINT'::leaftype, ARRAY[128.5::float8],
      ARRAY[28.5::float8], 4326, ARRAY['2019-01-01 00:00:05'::timestamp],
      ARRAY['speed'], NULL, ARRAY[106::float8], NULL) AS traj
)
Update traj
set traj = ST_append(traj.traj, a.traj)
From point_traj a
WHERE traj.ID =1;
-- Update the trajectory object based on multiple trajectory points.
With point_traj as (
 select ST_MakeTrajectory('STPOINT'::leaftype, x, y, 4326, t, ARRAY['speed'], NULL, s, NULL
) AS traj
 FROM (select array_agg(x order by id) as x,
      array_agg(y order by id) as y,
      array_agg(t order by id) as t,
      array_agg(speed order by id) as s
      From points WHERE ID > 5) a
```

```
      From points WHERE ID > 5 ) a
)
Update traj
set traj = ST_append(traj.traj, a.traj)
From point_traj a
WHERE traj.ID =1;
```

How do I enable advanced compression?

LZ4 is an advanced compression algorithm, which has a higher compression ratio and execution speed. To enable and disable the LZ4 compression algorithm, execute the following statements:

```
-- Enable LZ4 compression.
Set toast_compression_use_lz4 = true;
-- Disable LZ4 compression and use the default PostgreSQL compression algorithm.
Set toast_compression_use_lz4 = false;
```

To enable and disable the LZ4 compression algorithm for the entire database by default, execute the following statements:

```
-- Enable LZ4 compression for the database.
Alter database dbname Set toast_compression_use_lz4 = true;
-- Disable LZ4 compression and use the default compression algorithm for the database.
Alter database dbname Set toast_compression_use_lz4 = false;
```

How do I set a default length for string-type attribute fields?

You can use the GUC variable ganos.trajectory.attr_string_length to set a default length for string-type attribute fields. You can execute the following statement:

```
Set ganos.trajectory.attr_string_length = 32;
```

How do I calculate the maximum, minimum, or average value of an attribute field?

To calculate the maximum, minimum, or average value of an attribute field, execute the following statement:

```
With traj AS (
  select '{"trajectory":{"version":1,"type":"STPOINT","leafcount":2,"start_time":"2010-01-0
1 11:30:00","end_time":"2010-01-01 12:30:00","spatial":"SRID=4326;LINESTRING(1 1,3 5)","tim
eline":["2010-01-01 11:30:00","2010-01-01 12:30:00"],"attributes":{"leafcount":2,"velocity"
:{"type":"integer","length":4,"nullable":true,"value":[1,100]},"speed":{"type":"float","len
gth":8,"nullable":true,"value":[null,1.0]},"angel":{"type":"string","length":64,"nullable":
true,"value":["test",null]}, "tngel2":{"type":"timestamp","length":8,"nullable":true,"value
":["2010-01-01 12:30:00",null]},"bearing":{"type":"bool","length":1,"nullable":true,"value"
:[null,true]}}}}'::trajectory a)
select avg(v) from
(
Select unnest(st_trajAttrsAsInteger(a, 'velocity')) as v from traj
) t;
```

13.4. How do I update the Ganos plug-ins?

Ganos is a spatio-temporal database engine that is developed by Alibaba Cloud. Ganos provides a wide range of data types, functions, and stored procedures. This topic describes how to update the Ganos plug-ins on an ApsaraDB RDS instance.

Each Ganos plug-in is identified by the word Ganos in the description of the Ganos plug-in. To find and update the Ganos plug-ins that you installed, perform the following steps:

1. Use the PostgreSQL CLI to connect to your RDS instance.
2. Run the `\dx` command to view all plug-ins that you installed.

   Information similar to the following command output is displayed:

```
                                                                          List of insta
lled extensions
                Name                 | Version |   Schema   |
Description
------------------------------------+---------+------------+--------------------------
-------------------------------------------------------------------------------------
---
 address_standardizer               | 2.5.4   | public     | Ganos PostGIS+ address sta
ndardizer
 address_standardizer_data_us       | 2.5.4   | public     | Ganos PostGIS+ address sta
ndardizer data us
 ganos_address_standardizer         | 4.1     | public     | Used to parse an address i
nto constituent elements. Generally used to support geocoding address normalization ste
p.
 ganos_address_standardizer_data_us | 4.1     | public     | Address Standardizer US da
taset example
 ganos_geometry                     | 4.1     | public     | Ganos geometry extension f
or PostgreSQL
 ganos_geometry_sfcgal              | 4.1     | public     | Ganos geometry SFCGAL func
tions extension for PostgreSQL
 ganos_geometry_topology            | 4.1     | topology   | Ganos geometry topology sp
atial types and functions extension for PostgreSQL
 ganos_networking                   | 4.1     | public     | Ganos networking extension
for PostgreSQL
 ganos_spatialref                   | 4.1     | public     | Ganos spatial reference ex
tension for PostgreSQL
 ganos_tiger_geocoder               | 4.1     | tiger      | Ganos tiger geocoder and r
everse geocoder
 plpgsql                            | 1.0     | pg_catalog | PL/pgSQL procedural langua
ge
 postgis                            | 2.5.4   | public     | Ganos PostGIS+
 postgis_sfcgal                     | 2.5.4   | public     | Ganos PostGIS+
 postgis_tiger_geocoder             | 2.5.4   | public     | Ganos PostGIS+ tiger geoco
der
 postgis_topology                   | 2.5.4   | public     | Ganos PostGIS+ topology
```

3. View the information about each plug-in in the **Description** column of the command output to find the Ganos plug-ins that you want to update.

   > **Note** If the description of a plug-in includes the word Ganos, the plug-in is a Ganos plug-in.

4. Update the Ganos plug-ins.
   - If the Ganos plug-ins are in version 3.1 or later, execute the following statement to update all Ganos plug-ins:

     ```
     SELECT ganos_update();
     ```

   - If the Ganos plug-ins are in versions earlier than 3.1, create a function to update all Ganos plug-ins. The following code snippet is an example:

     ```
     CREATE OR REPLACE FUNCTION ganos_update()
         RETURNS text AS
     $$
     DECLARE
         rec RECORD;
         sql text;
     BEGIN
         FOR rec IN
             SELECT extname
             FROM pg_extension
             WHERE extname like 'ganos_%'
             LOOP
                 sql = 'ALTER EXTENSION '
                     || rec.extname
                     || ' UPDATE ';
                 RAISE NOTICE '%', sql;
                 EXECUTE sql;
             END LOOP;
         return 'All Ganos extensions have updated to latest version';
     END
     $$ LANGUAGE 'plpgsql' volatile STRICT;
     ```

   > **Note** The method of updating the Ganos plug-ins is different from the method of updating other plug-ins. To update a different plug-in, you can execute the following statement:
   >
   > ```
   > ALTER EXTENSION <Plug-in name> UPDATE;
   > ```

14.Best practice

14.1. Trajectory best practices

Use appropriate spatio-temporal indexes

You can create appropriate indexes based on application requirements to accelerate queries. Trajectory data supports the following index types:

- Spatial index: the index on space, which applies when you query only the spatial data of a trajectory.
- Temporal index: the index on time, which applies when you query only the time range of a trajectory.
- Spatio-temporal composite index: the composite index on both space and time, which applies when you query both the spatial data and time range of a trajectory.

```
-- Create a function-based spatial index to accelerate the filtering of spatial data.
create index tr_spatial_geometry_index on trajtab using gist (st_trajectoryspatial(traj));
-- Create a function-based temporal index to accelerate the filtering of time.
create index tr_timespan_time_index on trajtab using gist (st_timespan(traj));
-- Create function-based indexes on the start time and end time of a trajectory object.
create index tr_starttime_index on trajtab using btree (st_starttime(traj));
create index tr_endtime_index on trajtab using btree (st_endtime(traj));
-- Create a btree_gist extension.
create extension btree_gist;
-- Use btree_gist to create a spatio-temporal composite index on the start time, end time,
and spatial data of a trajectory object.
create index tr_traj_test_stm_etm_sp_index on traj_test using gist (st_starttime(traj),st_e
ndtime(traj),st_trajectoryspatial(traj));
```

Use appropriate partitioned tables

The amount of trajectory data in a database keeps increasing along with the continuous use of the database. As a result, a larger number of database indexes are created and data queries slow down. In this case, you can use partitioned tables to decrease the data size of a single table.

For more information about how to use partitioned tables, see Table partitioning.

Reduce the use of string-type attribute fields

A large number of string-type attribute fields in trajectory data lead to a waste of the storage space and performance deterioration.

- If string-type attribute fields have fixed values, they can be converted into integers. We recommend that you convert the data type in code.
- If string-type attribute fields are required, you can set a default length for the fields to save space.

  To set a default length for string-type attribute fields, execute the following statement:

  ```
  -- Set the default length of string-type attribute fields to 32.
  Set ganos.trajectory.attr_string_length = 32;
  ```

Use multiple trajectory points to generate a trajectory object

We recommend that you use multiple trajectory points to generate a trajectory object and avoid appending trajectory points one by one.

Use the advanced compression mode

LZ4 is an advanced compression algorithm, with a higher compression ratio and execution speed. To enable and disable the LZ4 compression algorithm, execute the following statements:

```
-- Enable LZ4 compression.
Set toast_compression_use_lz4 = true;
-- Disable LZ4 compression and use the default PostgreSQL compression algorithm.
Set toast_compression_use_lz4 = false;
```

To enable and disable the LZ4 compression algorithm for the entire database by default, execute the following statements:

```
-- Enable LZ4 compression for the database.
Alter database dbname Set toast_compression_use_lz4 = true;
-- Disable LZ4 compression and use the default compression algorithm for the database.
Alter database dbname Set toast_compression_use_lz4 = false;
```