

**PostGIS 3.3.6dev &#xc0ac;&#xc6a9;&#xc790;  
&#xc9c0;&#xce68;&#xc11c;**

---

**DEV (Wed 31 Jan 2024 09:01:06 PM UTC rev.  
3.3.5-29-g173f5cf47 )**

# Contents

|          |                     |          |
|----------|---------------------|----------|
| <b>1</b> | <b>Introduction</b> | <b>1</b> |
| 1.1      | Getting Started     | 1        |
| 1.2      | Installation        | 2        |
| 1.3      | Getting Help        | 2        |
| 1.4      | Reporting Bugs      | 3        |
| <b>2</b> | <b>PostGIS</b>      | <b>6</b> |
| 2.1      | Overview            | 6        |
| 2.2      | Architecture        | 7        |
| 2.2.1    | Architecture        | 8        |
| 2.2.2    | Architecture        | 8        |
| 2.2.3    | Architecture        | 10       |
| 2.2.4    | Architecture        | 12       |
| 2.2.5    | PostGIS Extensions  | 13       |
| 2.2.6    | Architecture        | 15       |
| 2.2.7    | Architecture        | 19       |
| 2.3      | Architecture        | 19       |
| 2.3.1    | Regex::Assemble     | 20       |
| 2.4      | Tiger Geocoder      | 20       |
| 2.4.1    | TIGER               | 21       |
| 2.4.1.1  | TIGER               | 21       |
| 2.4.2    | TIGER               | 24       |
| 2.4.3    | Architecture        | 25       |
| 2.4.4    | Tiger Data          | 25       |
| 2.4.5    | Tiger Geocoder      | 26       |
| 2.5      | Architecture        | 28       |

|          |   |           |
|----------|---|-----------|
| <b>3</b> | <b>PostGIS Administration</b>                                   | <b>29</b> |
| 3.1      | Performance Tuning  | 29        |
| 3.1.1    | Startup   | 29        |
| 3.1.2    | Runtime   | 30        |
| 3.2      | Configuring raster support                                      | 30        |
| 3.3      | Enabling spatial support  | 31        |
| 3.3.1    | Spatially enable database using EXTENSION                       | 31        |
| 3.3.2    | Spatially enable database without using EXTENSION (discouraged) | 31        |
| 3.3.3    | Create a spatially-enabled database from a template             | 32        |
| 3.4      | Upgrading spatial databases                                     | 32        |
| 3.4.1    | Soft upgrade  | 32        |
| 3.4.1.1  | Soft Upgrade 9.1+ using extensions                              | 32        |
| 3.4.1.2  | Soft Upgrade Pre 9.1+ or without extensions                     | 33        |
| 3.4.2    | Hard upgrade  | 34        |
| <b>4</b> | <b>Data Management</b>  | <b>36</b> |
| 4.1      | GIS Data Types  | 36        |
| 4.1.1    | OGC Geometry  | 36        |
| 4.1.1.1  | Point   | 37        |
| 4.1.1.2  | LineString  | 37        |
| 4.1.1.3  | LinearRing  | 37        |
| 4.1.1.4  | Polygon   | 37        |
| 4.1.1.5  | MultiPoint  | 37        |
| 4.1.1.6  | MultiLineString   | 37        |
| 4.1.1.7  | MultiPolygon  | 38        |
| 4.1.1.8  | GeometryCollection  | 38        |
| 4.1.1.9  | PolyhedralSurface   | 38        |
| 4.1.1.10 | Triangle  | 38        |
| 4.1.1.11 | TIN   | 38        |
| 4.1.2    | SQL-MM Part 3   | 38        |
| 4.1.2.1  | CircularString  | 39        |
| 4.1.2.2  | CompoundCurve   | 39        |
| 4.1.2.3  | CurvePolygon  | 39        |
| 4.1.2.4  | MultiCurve  | 40        |
| 4.1.2.5  | MultiSurface  | 40        |
| 4.1.3    | OpenGIS WKB and WKT   | 40        |
| 4.2      | Geometry Data Type  | 41        |
| 4.2.1    | OpenGIS WKB and WKT   | 42        |
| 4.3      | PostGIS Spatial Reference Systems                               | 43        |

---

- 4.3.1 [ST\\_Contains](#) . . . . . 44
- 4.3.2 [PostGIS ST\\_Contains](#) . . . . . 45
- 4.3.3 [ST\\_ContainsProperly](#) . . . . . 47
- 4.3.4 [ST\\_ContainsProperly](#); FAQ . . . . . 48
- 4.4 [Geometry Validation](#) . . . . . 49
  - 4.4.1 [Simple Geometry](#) . . . . . 49
  - 4.4.2 [Valid Geometry](#) . . . . . 51
  - 4.4.3 [Managing Validity](#) . . . . . 53
- 4.5 [SPATIAL\\_REF\\_SYS](#) . . . . . 55
  - 4.5.1 [SPATIAL\\_REF\\_SYS Table](#) . . . . . 55
  - 4.5.2 [SPATIAL\\_REF\\_SYS](#) . . . . . 55
- 4.6 [ST\\_Contains](#) . . . . . 57
  - 4.6.1 [ST\\_Contains](#) . . . . . 57
  - 4.6.2 [The GEOMETRY\\_COLUMNS VIEW](#) . . . . . 58
  - 4.6.3 [geometry\\_columns](#) . . . . . 58
- 4.7 [GIS \(ST\\_Contains\)](#) . . . . . 61
  - 4.7.1 [SQL](#) . . . . . 61
  - 4.7.2 [shp2pgsql: ESRI shapefile](#) . . . . . 62
- 4.8 [ST\\_Contains](#) . . . . . 65
  - 4.8.1 [SQL](#) . . . . . 65
  - 4.8.2 [shp2pgsql: ESRI shapefile](#) . . . . . 67
- 4.9 [ST\\_Contains](#) . . . . . 67
  - 4.9.1 [GiST](#) . . . . . 68
  - 4.9.2 [GiST](#) . . . . . 69
  - 4.9.3 [GiST](#) . . . . . 71
  - 4.9.4 [ST\\_Contains](#) . . . . . 71
- 5 Spatial Queries** . . . . . **73**
  - 5.1 [Determining Spatial Relationships](#) . . . . . 73
    - 5.1.1 [Dimensionally Extended 9-Intersection Model](#) . . . . . 73
    - 5.1.2 [Named Spatial Relationships](#) . . . . . 75
    - 5.1.3 [General Spatial Relationships](#) . . . . . 76
  - 5.2 [Using Spatial Indexes](#) . . . . . 78
  - 5.3 [Examples of Spatial SQL](#) . . . . . 78

|          |   |           |
|----------|---|-----------|
| <b>6</b> | <b>Geometry Constructors</b>              | <b>81</b> |
| 6.1      | Geometry Constructors                     | 81        |
| 6.1.1    | Geometry Constructors                     | 81        |
| 6.1.2    | Geometry Constructors                     | 82        |
| 6.2      | Geometry Constructors                     | 83        |
| 6.3      | Geometry Constructors                     | 83        |
| <b>7</b> | <b>PostGIS Database Integration</b>       |           |
| 7.1      | MapServer                                 | 85        |
| 7.1.1    | MapServer                                 | 85        |
| 7.1.2    | FAQ                                       | 87        |
| 7.1.3    | MapServer                                 | 88        |
| 7.1.4    | MapServer                                 | 89        |
| 7.2      | Java                                      | 91        |
| 7.3      | C   | 93        |
| 7.3.1    | C   | 93        |
| 7.3.2    | C   | 93        |
| <b>8</b> | <b>PostGIS Reference</b>                  | <b>94</b> |
| 8.1      | PostgreSQL PostGIS Geometry/Geography/Box | 94        |
| 8.1.1    | box2d                                     | 94        |
| 8.1.2    | box3d                                     | 95        |
| 8.1.3    | geometry                                  | 96        |
| 8.1.4    | geometry_dump                             | 96        |
| 8.1.5    | geography                                 | 96        |
| 8.2      | Geometry Constructors                     | 97        |
| 8.2.1    | AddGeometryColumn                         | 97        |
| 8.2.2    | DropGeometryColumn                        | 100       |
| 8.2.3    | DropGeometryTable                         | 101       |
| 8.2.4    | Find_SRID                                 | 101       |
| 8.2.5    | Populate_Geometry_Columns                 | 102       |
| 8.2.6    | UpdateGeometrySRID                        | 104       |
| 8.3      | Geometry Constructors (constructor)       | 105       |
| 8.3.1    | ST_GeomCollFromText                       | 105       |
| 8.3.2    | ST_LineFromMultiPoint                     | 107       |
| 8.3.3    | ST_MakeEnvelope                           | 108       |
| 8.3.4    | ST_MakeLine                               | 109       |
| 8.3.5    | ST_MakePoint                              | 110       |

|        |                     |     |
|--------|---------------------|-----|
| 8.3.6  | ST_MakePointM       | 111 |
| 8.3.7  | ST_MakePolygon      | 112 |
| 8.3.8  | ST_Point            | 115 |
| 8.3.9  | ST_Point            | 116 |
| 8.3.10 | ST_Point            | 117 |
| 8.3.11 | ST_Point            | 117 |
| 8.3.12 | ST_Polygon          | 118 |
| 8.3.13 | ST_MakeEnvelope     | 119 |
| 8.3.14 | ST_HexagonGrid      | 119 |
| 8.3.15 | ST_Hexagon          | 122 |
| 8.3.16 | ST_SquareGrid       | 123 |
| 8.3.17 | ST_Square           | 124 |
| 8.3.18 | ST_Letters          | 125 |
| 8.4    | (accessor)          | 126 |
| 8.4.1  |                     | 126 |
| 8.4.2  | ST_Boundary         | 127 |
| 8.4.3  | ST_BoundingDiagonal | 130 |
| 8.4.4  | ST_CoordDim         | 131 |
| 8.4.5  | ST_Dimension        | 132 |
| 8.4.6  | ST_Dump             | 132 |
| 8.4.7  | ST_NumPoints        | 134 |
| 8.4.8  | ST_NumPoints        | 138 |
| 8.4.9  | ST_NRings           | 140 |
| 8.4.10 | ST_EndPoint         | 141 |
| 8.4.11 | ST_Envelope         | 143 |
| 8.4.12 | ST_ExteriorRing     | 144 |
| 8.4.13 | ST_GeometryN        | 145 |
| 8.4.14 | ST_GeometryType     | 148 |
| 8.4.15 | ST_HasArc           | 149 |
| 8.4.16 | ST_InteriorRingN    | 150 |
| 8.4.17 | ST_IsClosed         | 150 |
| 8.4.18 | ST_IsCollection     | 152 |
| 8.4.19 | ST_IsEmpty          | 153 |
| 8.4.20 | ST_IsPolygonCCW     | 155 |
| 8.4.21 | ST_IsPolygonCW      | 155 |
| 8.4.22 | ST_IsRing           | 156 |
| 8.4.23 | ST_IsSimple         | 157 |
| 8.4.24 | ST_M                | 158 |
| 8.4.25 | ST_MemSize          | 159 |

|        |                         |     |
|--------|-------------------------|-----|
| 8.4.26 | ST_NDims                | 160 |
| 8.4.27 | ST_NPoints              | 160 |
| 8.4.28 | ST_NRings               | 161 |
| 8.4.29 | ST_NumGeometries        | 162 |
| 8.4.30 | ST_NumInteriorRings     | 163 |
| 8.4.31 | ST_NumInteriorRing      | 164 |
| 8.4.32 | ST_NumPatches           | 164 |
| 8.4.33 | ST_NumPoints            | 165 |
| 8.4.34 | ST_PatchN               | 166 |
| 8.4.35 | ST_PointN               | 167 |
| 8.4.36 | ST_Points               | 169 |
| 8.4.37 | ST_StartPoint           | 169 |
| 8.4.38 | ST_Summary              | 171 |
| 8.4.39 | ST_X                    | 172 |
| 8.4.40 | ST_Y                    | 173 |
| 8.4.41 | ST_Z                    | 173 |
| 8.4.42 | ST_Zmflag               | 174 |
| 8.5    | (editor)                | 175 |
| 8.5.1  | ST_AddPoint             | 175 |
| 8.5.2  | ST_CollectionExtract    | 176 |
| 8.5.3  | ST_CollectionHomogenize | 177 |
| 8.5.4  | ST_CurveToLine          | 179 |
| 8.5.5  | ST_Scroll               | 181 |
| 8.5.6  | ST_FlipCoordinates      | 182 |
| 8.5.7  | ST_Force2D              | 183 |
| 8.5.8  | ST_Force3D              | 183 |
| 8.5.9  | ST_Force3DZ             | 184 |
| 8.5.10 | ST_Force3DM             | 185 |
| 8.5.11 | ST_Force4D              | 186 |
| 8.5.12 | ST_ForcePolygonCCW      | 187 |
| 8.5.13 | ST_ForceCollection      | 187 |
| 8.5.14 | ST_ForcePolygonCW       | 189 |
| 8.5.15 | ST_ForceSFS             | 189 |
| 8.5.16 | ST_ForceRHR             | 189 |
| 8.5.17 | ST_ForceCurve           | 190 |
| 8.5.18 | ST_LineToCurve          | 191 |
| 8.5.19 | ST_Multi                | 193 |
| 8.5.20 | ST_Normalize            | 193 |
| 8.5.21 | ST_QuantizeCoordinates  | 194 |

---

|          |                                    |     |
|----------|------------------------------------|-----|
| 8.5.22   | ST_RemovePoint                     | 196 |
| 8.5.23   | ST_RemoveRepeatedPoints            | 197 |
| 8.5.24   | ST_Reverse                         | 197 |
| 8.5.25   | ST_Segmentize                      | 198 |
| 8.5.26   | ST_SetPoint                        | 199 |
| 8.5.27   | ST_ShiftLongitude                  | 200 |
| 8.5.28   | ST_WrapX                           | 201 |
| 8.5.29   | ST_SnapToGrid                      | 202 |
| 8.5.30   | ST_Snap                            | 204 |
| 8.5.31   | ST_SwapOrdinates                   | 208 |
| 8.6      | Geometry Validation                | 208 |
| 8.6.1    | ST_IsValid                         | 208 |
| 8.6.2    | ST_IsValidDetail                   | 209 |
| 8.6.3    | ST_IsValidReason                   | 211 |
| 8.6.4    | ST_MakeValid                       | 212 |
| 8.7      | Spatial Reference System Functions | 217 |
| 8.7.1    | ST_SetSRID                         | 217 |
| 8.7.2    | ST_SRID                            | 218 |
| 8.7.3    | ST_Transform                       | 219 |
| 8.8      | Geometry Input                     | 221 |
| 8.8.1    | Well-Known Text (WKT)              | 221 |
| 8.8.1.1  | ST_BdPolyFromText                  | 221 |
| 8.8.1.2  | ST_BdMPolyFromText                 | 222 |
| 8.8.1.3  | ST_GeogFromText                    | 222 |
| 8.8.1.4  | ST_GeographyFromText               | 223 |
| 8.8.1.5  | ST_GeomCollFromText                | 223 |
| 8.8.1.6  | ST_GeomFromEWKT                    | 224 |
| 8.8.1.7  | ST_GeomFromMARC21                  | 226 |
| 8.8.1.8  | ST_GeometryFromText                | 228 |
| 8.8.1.9  | ST_GeomFromText                    | 228 |
| 8.8.1.10 | ST_LineFromText                    | 230 |
| 8.8.1.11 | ST_MLineFromText                   | 231 |
| 8.8.1.12 | ST_MPointFromText                  | 232 |
| 8.8.1.13 | ST_MPolyFromText                   | 232 |
| 8.8.1.14 | ST_PointFromText                   | 233 |
| 8.8.1.15 | ST_PolygonFromText                 | 234 |
| 8.8.1.16 | ST_WKTToSQL                        | 235 |
| 8.8.2    | Well-Known Binary (WKB)            | 236 |
| 8.8.2.1  | ST_GeogFromWKB                     | 236 |

---

|          |                            |     |
|----------|----------------------------|-----|
| 8.8.2.2  | ST_GeomFromEWKB            | 237 |
| 8.8.2.3  | ST_GeomFromWKB             | 238 |
| 8.8.2.4  | ST_LineFromWKB             | 239 |
| 8.8.2.5  | ST_LinestringFromWKB       | 240 |
| 8.8.2.6  | ST_PointFromWKB            | 241 |
| 8.8.2.7  | ST_WKBToSQL                | 242 |
| 8.8.3    | Other Formats              | 243 |
| 8.8.3.1  | ST_Box2dFromGeoHash        | 243 |
| 8.8.3.2  | ST_GeomFromGeoHash         | 243 |
| 8.8.3.3  | ST_GeomFromGML             | 244 |
| 8.8.3.4  | ST_GeomFromGeoJSON         | 247 |
| 8.8.3.5  | ST_GeomFromKML             | 248 |
| 8.8.3.6  | ST_GeomFromTWKB            | 249 |
| 8.8.3.7  | ST_GMLToSQL                | 250 |
| 8.8.3.8  | ST_LineFromEncodedPolyline | 250 |
| 8.8.3.9  | ST_PointFromGeoHash        | 251 |
| 8.8.3.10 | ST_FromFlatGeobufToTable   | 252 |
| 8.8.3.11 | ST_FromFlatGeobuf          | 252 |
| 8.9      | Geometry Output            | 253 |
| 8.9.1    | Well-Known Text (WKT)      | 253 |
| 8.9.1.1  | ST_AsEWKT                  | 253 |
| 8.9.1.2  | ST_AsText                  | 254 |
| 8.9.2    | Well-Known Binary (WKB)    | 255 |
| 8.9.2.1  | ST_AsBinary                | 255 |
| 8.9.2.2  | ST_AsEWKB                  | 257 |
| 8.9.2.3  | ST_AsHEXEWKB               | 258 |
| 8.9.3    | Other Formats              | 259 |
| 8.9.3.1  | ST_AsEncodedPolyline       | 259 |
| 8.9.3.2  | ST_AsFlatGeobuf            | 260 |
| 8.9.3.3  | ST_AsGeobuf                | 260 |
| 8.9.3.4  | ST_AsGeoJSON               | 261 |
| 8.9.3.5  | ST_AsGML                   | 263 |
| 8.9.3.6  | ST_AsKML                   | 267 |
| 8.9.3.7  | ST_AsLatLonText            | 269 |
| 8.9.3.8  | ST_AsMARC21                | 270 |
| 8.9.3.9  | ST_AsMVTGeom               | 272 |
| 8.9.3.10 | ST_AsMVT                   | 273 |
| 8.9.3.11 | ST_AsSVG                   | 274 |
| 8.9.3.12 | ST_AsTWKB                  | 275 |

---

|           |                           |     |
|-----------|---------------------------|-----|
| 8.9.3.13  | ST_AsX3D                  | 277 |
| 8.9.3.14  | ST_GeoHash                | 281 |
| 8.10      | (operator)                | 283 |
| 8.10.1    | Bounding Box Operators    | 283 |
| 8.10.1.1  | &&                        | 283 |
| 8.10.1.2  | &&(geometry,box2df)       | 284 |
| 8.10.1.3  | &&(box2df,geometry)       | 284 |
| 8.10.1.4  | &&(box2df,box2df)         | 285 |
| 8.10.1.5  | &&&                       | 286 |
| 8.10.1.6  | &&&(geometry,gidx)        | 287 |
| 8.10.1.7  | &&&(gidx,geometry)        | 288 |
| 8.10.1.8  | &&&(gidx,gidx)            | 289 |
| 8.10.1.9  | &<                        | 289 |
| 8.10.1.10 | &<                        | 290 |
| 8.10.1.11 | &>                        | 291 |
| 8.10.1.12 | <<                        | 292 |
| 8.10.1.13 | <<                        | 293 |
| 8.10.1.14 | =                         | 294 |
| 8.10.1.15 | >>                        | 295 |
| 8.10.1.16 | @                         | 296 |
| 8.10.1.17 | @(geometry,box2df)        | 297 |
| 8.10.1.18 | @(box2df,geometry)        | 298 |
| 8.10.1.19 | @(box2df,box2df)          | 298 |
| 8.10.1.20 | &>                        | 299 |
| 8.10.1.21 | >>                        | 300 |
| 8.10.1.22 | ~                         | 301 |
| 8.10.1.23 | ~(geometry,box2df)        | 301 |
| 8.10.1.24 | ~(box2df,geometry)        | 302 |
| 8.10.1.25 | ~(box2df,box2df)          | 303 |
| 8.10.1.26 | ~=                        | 304 |
| 8.10.2    | (operator)                | 304 |
| 8.10.2.1  | <->                       | 304 |
| 8.10.2.2  | =                         | 307 |
| 8.10.2.3  | <#>                       | 308 |
| 8.10.2.4  | <<->>                     | 310 |
| 8.10.2.5  | <<#>>                     | 310 |
| 8.11      | Spatial Relationships     | 311 |
| 8.11.1    | Topological Relationships | 311 |
| 8.11.1.1  | ST_3DIntersects           | 311 |

---

|           |                          |     |
|-----------|--------------------------|-----|
| 8.11.1.2  | ST_Contains              | 312 |
| 8.11.1.3  | ST_ContainsProperly      | 316 |
| 8.11.1.4  | ST_CoveredBy             | 317 |
| 8.11.1.5  | ST_Covers                | 318 |
| 8.11.1.6  | ST_Crosses               | 320 |
| 8.11.1.7  | ST_Disjoint              | 322 |
| 8.11.1.8  | ST_Equals                | 323 |
| 8.11.1.9  | ST_Intersects            | 324 |
| 8.11.1.10 | ST_LineCrossingDirection | 325 |
| 8.11.1.11 | ST_OrderingEquals        | 328 |
| 8.11.1.12 | ST_Overlaps              | 329 |
| 8.11.1.13 | ST_Relate                | 332 |
| 8.11.1.14 | ST_RelateMatch           | 334 |
| 8.11.1.15 | ST_Touches               | 335 |
| 8.11.1.16 | ST_Within                | 337 |
| 8.11.2    | Distance Relationships   | 338 |
| 8.11.2.1  | ST_3DDWithin             | 338 |
| 8.11.2.2  | ST_3DDFullyWithin        | 339 |
| 8.11.2.3  | ST_DFullyWithin          | 340 |
| 8.11.2.4  | ST_DWithin               | 341 |
| 8.11.2.5  | ST_PointInsideCircle     | 342 |
| 8.12      | Measurement Functions    | 343 |
| 8.12.1    | ST_Area                  | 343 |
| 8.12.2    | ST_Azimuth               | 345 |
| 8.12.3    | ST_Angle                 | 346 |
| 8.12.4    | ST_ClosestPoint          | 348 |
| 8.12.5    | ST_3DClosestPoint        | 349 |
| 8.12.6    | ST_Distance              | 351 |
| 8.12.7    | ST_3DDistance            | 352 |
| 8.12.8    | ST_DistanceSphere        | 354 |
| 8.12.9    | ST_DistanceSpheroid      | 355 |
| 8.12.10   | ST_FrechetDistance       | 356 |
| 8.12.11   | ST_HausdorffDistance     | 357 |
| 8.12.12   | ST_Length                | 358 |
| 8.12.13   | ST_Length2D              | 360 |
| 8.12.14   | ST_3DLength              | 361 |
| 8.12.15   | ST_LengthSpheroid        | 361 |
| 8.12.16   | ST_LongestLine           | 363 |
| 8.12.17   | ST_3DLongestLine         | 365 |

---

|         |                           |     |
|---------|---------------------------|-----|
| 8.12.18 | ST_MaxDistance            | 366 |
| 8.12.19 | ST_3DMaxDistance          | 367 |
| 8.12.20 | ST_MinimumClearance       | 368 |
| 8.12.21 | ST_MinimumClearanceLine   | 369 |
| 8.12.22 | ST_Perimeter              | 370 |
| 8.12.23 | ST_Perimeter2D            | 372 |
| 8.12.24 | ST_3DPerimeter            | 372 |
| 8.12.25 | ST_Project                | 373 |
| 8.12.26 | ST_ShortestLine           | 374 |
| 8.12.27 | ST_3DShortestLine         | 375 |
| 8.13    | Overlay Functions         | 377 |
| 8.13.1  | ST_ClipByBox2D            | 377 |
| 8.13.2  | ST_Difference             | 377 |
| 8.13.3  | ST_Intersection           | 379 |
| 8.13.4  | ST_MemUnion               | 381 |
| 8.13.5  | ST_Node                   | 382 |
| 8.13.6  | ST_Split                  | 383 |
| 8.13.7  | ST_Subdivide              | 385 |
| 8.13.8  | ST_SymDifference          | 387 |
| 8.13.9  | ST_UnaryUnion             | 389 |
| 8.13.10 | ST_Union                  | 389 |
| 8.14    | Geometric Primitives      | 392 |
| 8.14.1  | ST_Buffer                 | 392 |
| 8.14.2  | ST_BuildArea              | 396 |
| 8.14.3  | ST_Centroid               | 397 |
| 8.14.4  | ST_ChaikinSmoothing       | 399 |
| 8.14.5  | ST_ConcaveHull            | 400 |
| 8.14.6  | ST_ConvexHull             | 403 |
| 8.14.7  | ST_DelaunayTriangles      | 404 |
| 8.14.8  | ST_FilterByM              | 409 |
| 8.14.9  | ST_GeneratePoints         | 410 |
| 8.14.10 | ST_GeometricMedian        | 410 |
| 8.14.11 | ST_LineMerge              | 412 |
| 8.14.12 | ST_MaximumInscribedCircle | 414 |
| 8.14.13 | ST_MinimumBoundingCircle  | 416 |
| 8.14.14 | ST_MinimumBoundingRadius  | 418 |
| 8.14.15 | ST_OrientedEnvelope       | 418 |
| 8.14.16 | ST_OffsetCurve            | 420 |
| 8.14.17 | ST_PointOnSurface         | 423 |

---

|   |     |
|---|-----|
| 8.14.18 ST_Polygonize . . . . .               | 425 |
| 8.14.19 ST_ReducePrecision . . . . .          | 426 |
| 8.14.20 ST_SharedPaths . . . . .              | 428 |
| 8.14.21 ST_Simplify . . . . .                 | 430 |
| 8.14.22 ST_SimplifyPreserveTopology . . . . . | 431 |
| 8.14.23 ST_SimplifyPolygonHull . . . . .      | 432 |
| 8.14.24 ST_SimplifyVW . . . . .               | 434 |
| 8.14.25 ST_SetEffectiveArea . . . . .         | 435 |
| 8.14.26 ST_TriangulatePolygon . . . . .       | 437 |
| 8.14.27 ST_VoronoiLines . . . . .             | 438 |
| 8.14.28 ST_VoronoiPolygons . . . . .          | 439 |
| 8.15 Affine Transformations . . . . .         | 442 |
| 8.15.1 ST_Affine . . . . .                    | 442 |
| 8.15.2 ST_Rotate . . . . .                    | 444 |
| 8.15.3 ST_RotateX . . . . .                   | 445 |
| 8.15.4 ST_RotateY . . . . .                   | 446 |
| 8.15.5 ST_RotateZ . . . . .                   | 446 |
| 8.15.6 ST_Scale . . . . .                     | 448 |
| 8.15.7 ST_Translate . . . . .                 | 449 |
| 8.15.8 ST_TransScale . . . . .                | 450 |
| 8.16 Clustering Functions . . . . .           | 451 |
| 8.16.1 ST_ClusterDBSCAN . . . . .             | 451 |
| 8.16.2 ST_ClusterIntersecting . . . . .       | 454 |
| 8.16.3 ST_ClusterKMeans . . . . .             | 454 |
| 8.16.4 ST_ClusterWithin . . . . .             | 456 |
| 8.17 Bounding Box Functions . . . . .         | 457 |
| 8.17.1 Box2D . . . . .                        | 457 |
| 8.17.2 Box3D . . . . .                        | 458 |
| 8.17.3 ST_EstimatedExtent . . . . .           | 459 |
| 8.17.4 ST_Expand . . . . .                    | 459 |
| 8.17.5 ST_Extent . . . . .                    | 461 |
| 8.17.6 ST_3DExtent . . . . .                  | 462 |
| 8.17.7 ST_MakeBox2D . . . . .                 | 463 |
| 8.17.8 ST_3DMakeBox . . . . .                 | 464 |
| 8.17.9 ST_XMax . . . . .                      | 464 |
| 8.17.10 ST_XMin . . . . .                     | 465 |
| 8.17.11 ST_YMax . . . . .                     | 466 |
| 8.17.12 ST_YMin . . . . .                     | 467 |
| 8.17.13 ST_ZMax . . . . .                     | 468 |

|         |                                 |     |
|---------|---------------------------------|-----|
| 8.17.14 | ST_ZMin                         | 469 |
| 8.18    | (Linear Referencing)            | 470 |
| 8.18.1  | ST_LineInterpolatePoint         | 470 |
| 8.18.2  | ST_LineInterpolatePoint         | 472 |
| 8.18.3  | ST_LineInterpolatePoints        | 473 |
| 8.18.4  | ST_LineLocatePoint              | 474 |
| 8.18.5  | ST_LineSubstring                | 475 |
| 8.18.6  | ST_LocateAlong                  | 477 |
| 8.18.7  | ST_LocateBetween                | 478 |
| 8.18.8  | ST_LocateBetweenElevations      | 480 |
| 8.18.9  | ST_InterpolatePoint             | 480 |
| 8.18.10 | ST_AddMeasure                   | 481 |
| 8.19    | Trajectory Functions            | 482 |
| 8.19.1  | ST_IsValidTrajectory            | 482 |
| 8.19.2  | ST_ClosestPointOfApproach       | 483 |
| 8.19.3  | ST_DistanceCPA                  | 484 |
| 8.19.4  | ST_CPAWithin                    | 485 |
| 8.20    | SFCGAL                          | 485 |
| 8.20.1  | postgis_sfcgal_version          | 485 |
| 8.20.2  | postgis_sfcgal_version          | 486 |
| 8.20.3  | ST_3DArea                       | 486 |
| 8.20.4  | ST_3DConvexHull                 | 487 |
| 8.20.5  | ST_3DIntersection               | 489 |
| 8.20.6  | ST_3DDifference                 | 491 |
| 8.20.7  | ST_3DUnion                      | 492 |
| 8.20.8  | ST_AlphaShape                   | 493 |
| 8.20.9  | ST_ApproximateMedialAxis        | 496 |
| 8.20.10 | ST_ConstrainedDelaunayTriangles | 497 |
| 8.20.11 | ST_Extrude                      | 498 |
| 8.20.12 | ST_ForceLHR                     | 500 |
| 8.20.13 | ST_IsPlanar                     | 500 |
| 8.20.14 | ST_IsSolid                      | 501 |
| 8.20.15 | ST_MakeSolid                    | 501 |
| 8.20.16 | ST_MinkowskiSum                 | 502 |
| 8.20.17 | ST_OptimalAlphaShape            | 504 |
| 8.20.18 | ST_Orientation                  | 506 |
| 8.20.19 | ST_StraightSkeleton             | 507 |
| 8.20.20 | ST_Tesselate                    | 508 |
| 8.20.21 | ST_Volume                       | 510 |

|           |  |            |
|-----------|--|------------|
| 8.21      | (Long Transaction)                         | 511        |
| 8.21.1    | AddAuth                                    | 511        |
| 8.21.2    | CheckAuth                                  | 512        |
| 8.21.3    | DisableLongTransactions                    | 513        |
| 8.21.4    | EnableLongTransactions                     | 513        |
| 8.21.5    | LockRow                                    | 514        |
| 8.21.6    | UnlockRows                                 | 515        |
| 8.22      | Version Functions                          | 515        |
| 8.22.1    | PostGIS_Extensions_Upgrade                 | 515        |
| 8.22.2    | PostGIS_Full_Version                       | 516        |
| 8.22.3    | PostGIS_GEOS_Version                       | 517        |
| 8.22.4    | PostGIS_Liblwgeom_Version                  | 517        |
| 8.22.5    | PostGIS_LibXML_Version                     | 518        |
| 8.22.6    | PostGIS_Lib_Build_Date                     | 518        |
| 8.22.7    | PostGIS_Lib_Version                        | 519        |
| 8.22.8    | PostGIS_PROJ_Version                       | 519        |
| 8.22.9    | PostGIS_Wagyu_Version                      | 520        |
| 8.22.10   | PostGIS_Scripts_Build_Date                 | 520        |
| 8.22.11   | PostGIS_Scripts_Installed                  | 521        |
| 8.22.12   | PostGIS_Scripts_Released                   | 521        |
| 8.22.13   | PostGIS_Version                            | 522        |
| 8.23      | PostGIS GUC(Grand Unified Custom Variable) | 522        |
| 8.23.1    | postgis.backend                            | 522        |
| 8.23.2    | postgis.gdal_datapath                      | 523        |
| 8.23.3    | postgis.gdal_enabled_drivers               | 524        |
| 8.23.4    | postgis.enable_outdb_rasters               | 526        |
| 8.23.5    | postgis.gdal_vsi_options                   | 527        |
| 8.24      | Troubleshooting Functions                  | 527        |
| 8.24.1    | PostGIS_AddBBBox                           | 527        |
| 8.24.2    | PostGIS_DropBBBox                          | 528        |
| 8.24.3    | PostGIS_HasBBBox                           | 529        |
| <b>9</b>  | <b>PostGIS</b>                             | <b>530</b> |
| <b>10</b> | <b>(topology)</b>                          | <b>538</b> |
| 10.1      |  | 539        |
| 10.1.1    | getfaceedges_returntype                    | 539        |
| 10.1.2    | TopoGeometry                               | 539        |
| 10.1.3    | validatetopology_returntype                | 540        |

|         |                                |     |
|---------|--------------------------------|-----|
| 10.2    |                                | 540 |
| 10.2.1  | TopoElement                    | 540 |
| 10.2.2  | TopoElementArray               | 541 |
| 10.3    |                                | 542 |
| 10.3.1  | AddTopoGeometryColumn          | 542 |
| 10.3.2  | DropTopology                   | 543 |
| 10.3.3  | DropTopoGeometryColumn         | 544 |
| 10.3.4  | Populate_Topology_Layer        | 545 |
| 10.3.5  | TopologySummary                | 546 |
| 10.3.6  | ValidateTopology               | 546 |
| 10.3.7  | ValidateTopologyRelation       | 549 |
| 10.3.8  | FindTopology                   | 549 |
| 10.3.9  | FindLayer                      | 550 |
| 10.4    | Topology Statistics Management | 550 |
| 10.5    |                                | 550 |
| 10.5.1  | CreateTopology                 | 550 |
| 10.5.2  | CopyTopology                   | 551 |
| 10.5.3  | ST_InitTopoGeo                 | 552 |
| 10.5.4  | ST_CreateTopoGeo               | 553 |
| 10.5.5  | TopoGeo_AddPoint               | 554 |
| 10.5.6  | TopoGeo_AddLineString          | 554 |
| 10.5.7  | TopoGeo_AddPolygon             | 555 |
| 10.6    |                                | 555 |
| 10.6.1  | ST_AddIsoNode                  | 555 |
| 10.6.2  | ST_AddIsoEdge                  | 556 |
| 10.6.3  | ST_AddEdgeNewFaces             | 557 |
| 10.6.4  | ST_AddEdgeModFace              | 558 |
| 10.6.5  | ST_RemEdgeNewFace              | 558 |
| 10.6.6  | ST_RemEdgeModFace              | 559 |
| 10.6.7  | ST_ChangeEdgeGeom              | 560 |
| 10.6.8  | ST_ModEdgeSplit                | 561 |
| 10.6.9  | ST_ModEdgeHeal                 | 562 |
| 10.6.10 | ST_NewEdgeHeal                 | 562 |
| 10.6.11 | ST_MoveIsoNode                 | 563 |
| 10.6.12 | ST_NewEdgesSplit               | 564 |
| 10.6.13 | ST_RemoveIsoNode               | 565 |
| 10.6.14 | ST_RemoveIsoEdge               | 565 |
| 10.7    |                                | 566 |
| 10.7.1  | GetEdgeByPoint                 | 566 |

|         |                                    |     |
|---------|------------------------------------|-----|
| 10.7.2  | GetFaceByPoint                     | 567 |
| 10.7.3  | GetFaceContainingPoint             | 568 |
| 10.7.4  | GetNodeByPoint                     | 568 |
| 10.7.5  | GetTopologyID                      | 569 |
| 10.7.6  | GetTopologySRID                    | 569 |
| 10.7.7  | GetTopologyName                    | 570 |
| 10.7.8  | ST_GetFaceEdges                    | 571 |
| 10.7.9  | ST_GetFaceGeometry                 | 572 |
| 10.7.10 | GetRingEdges                       | 572 |
| 10.7.11 | GetNodeEdges                       | 573 |
| 10.8    | Polygonize                         | 574 |
| 10.8.1  | Polygonize                         | 574 |
| 10.8.2  | AddNode                            | 575 |
| 10.8.3  | AddEdge                            | 576 |
| 10.8.4  | AddFace                            | 577 |
| 10.8.5  | ST_Simplify                        | 579 |
| 10.8.6  | RemoveUnusedPrimitives             | 580 |
| 10.9    | TopoGeometry                       | 580 |
| 10.9.1  | CreateTopoGeom                     | 580 |
| 10.9.2  | toTopoGeom                         | 582 |
| 10.9.3  | TopoElementArray_Agg               | 584 |
| 10.10   | TopoGeometry                       | 584 |
| 10.10.1 | clearTopoGeom                      | 584 |
| 10.10.2 | TopoGeom_addElement                | 585 |
| 10.10.3 | TopoGeom_remElement                | 585 |
| 10.10.4 | TopoGeom_addTopoGeom               | 586 |
| 10.10.5 | toTopoGeom                         | 586 |
| 10.11   | TopoGeometry                       | 587 |
| 10.11.1 | GetTopoGeomElementArray            | 587 |
| 10.11.2 | GetTopoGeomElements                | 587 |
| 10.11.3 | ST_SRID                            | 588 |
| 10.12   | TopoGeometry                       | 588 |
| 10.12.1 | AsGML                              | 588 |
| 10.12.2 | AsTopoJSON                         | 591 |
| 10.13   | ST_Equals                          | 593 |
| 10.13.1 | ST_Equals                          | 593 |
| 10.13.2 | ST_Equals                          | 594 |
| 10.14   | Importing and exporting Topologies | 594 |
| 10.14.1 | Using the Topology exporter        | 595 |
| 10.14.2 | Using the Topology importer        | 595 |

|           |  |            |
|-----------|--|------------|
| <b>11</b> | <b>&amp;#xb798;&amp;#xc2a4;&amp;#xd130; &amp;#xb370;&amp;#xc774;&amp;#xd130;&amp;#xc758; &amp;#xad00;&amp;#xb9ac;, &amp;#xcffc;&amp;#xb9ac; &amp;#xbc0f; &amp;#xc751;&amp;#xc6a9;</b>                                | <b>596</b> |
| 11.1      | &#xb798;&#xc2a4;&#xd130; &#xb85c;&#xb4dc; &#xbc0f; &#xc0dd;&#xc131; . . . . .  | 596        |
| 11.1.1    | raster2pgsql&#xc744; &#xc774;&#xc6a9;&#xd574; &#xb798;&#xc2a4;&#xd130;&#xb97c; &#xb85c;&#xb4dc;&#xc751;&#xc6a9;  |            |
| 11.1.2    | PostGIS &#xb798;&#xc2a4;&#xd130; &#xd568;&#xc218;&#xb97c; &#xc774;&#xc6a9;&#xd574; &#xb798;&#xc2a4;&#xd130; &#xc0dd;&#xc131;&#xd558;&#xae30; . . . . .   | 602        |
| 11.1.3    | Using "out db" cloud rasters . . . . .   | 603        |
| 11.2      | &#xb798;&#xc2a4;&#xd130; &#xce74;&#xd0c8;&#xb85c;&#xadf8; . . . . .  | 604        |
| 11.2.1    | &#xb798;&#xc2a4;&#xd130; &#xc5f4; &#xce74;&#xd0c8;&#xb85c;&#xadf8; . . . . .   | 604        |
| 11.2.2    | &#xb798;&#xc2a4;&#xd130; &#xc624;&#xbc84;&#xbdf0; . . . . .  | 606        |
| 11.3      | PostGIS &#xb798;&#xc2a4;&#xd130;&#xb97c; &#xc774;&#xc6a9;&#xd558;&#xb294; &#xc0ac;&#xc6a9;&#xc790; &#xc9c0;&#xc815; &#xc751;&#xc6a9; &#xd504;&#xb85c;&#xadf8;&#xb7a8; &#xbe4c;&#xb4dc;&#xd558;&#xae30;               | 608        |
| 11.3.1    | &#xb2e4;&#xb978; &#xb798;&#xc2a4;&#xd130; &#xd568;&#xc218;&#xc640; &#xd568;&#xaed8; ST_AsPNG&#xb97c; &#xc774;&#xc6a9;&#xd574;&#xc11c; PHP &#xc608;&#xc81c;&#xb97c; &#xcd9c;&#xb825;                                  |            |
| 11.3.2    | &#xb2e4;&#xb978; &#xb798;&#xc2a4;&#xd130; &#xd568;&#xc218;&#xc640; &#xd568;&#xaed8; ST_AsPNG&#xb97c; &#xc774;&#xc6a9;&#xd574;&#xc11c; ASP.NET C# &#xc608;&#xc81c;&#xb97c; &#xcd9c;&#xb825;&#xd558;&#xae30; . . . . . | 609        |
| 11.3.3    | &#xb798;&#xc2a4;&#xd130; &#xcffc;&#xb9ac;&#xb97c; &#xc774;&#xbbf8;&#xc9c0; &#xd30c;&#xc77c;&#xb85c;&#xcd9c;&#xb825;&#xd558;&#xb294; Java &#xc5f4;&#xc194; &#xc751;&#xc6a9; &#xd504;&#xb85c;&#xadf8;&#xc751;&#xc6a9;  |            |
| 11.3.4    | PLPython&#xc744; &#xc774;&#xc6a9;&#xd574;&#xc11c; SQL&#xc744; &#xd1b5;&#xd574; &#xc774;&#xbbf8;&#xb364;&#xd504;&#xd558;&#xae30; . . . . .  | 612        |
| 11.3.5    | PSQL&#xc744; &#xc774;&#xc6a9;&#xd574;&#xc11c; &#xb798;&#xc2a4;&#xd130; &#xcd9c;&#xb825;&#xd558;&#xae30;  |            |
| <b>12</b> | <b>&amp;#xb798;&amp;#xc2a4;&amp;#xd2b8; &amp;#xc38;&amp;#xc870;&amp;#xbb38;&amp;#xc11c;</b>  | <b>614</b> |
| 12.1      | &#xb798;&#xc2a4;&#xd130; &#xc9c0;&#xc6d0; &#xb370;&#xc774;&#xd130;&#xd615; . . . . .   | 615        |
| 12.1.1    | geomval . . . . .  | 615        |
| 12.1.2    | addbandarg . . . . .   | 615        |
| 12.1.3    | rastbandarg . . . . .  | 616        |
| 12.1.4    | raster . . . . .   | 616        |
| 12.1.5    | reclassarg . . . . .   | 617        |
| 12.1.6    | summarystats . . . . .   | 618        |
| 12.1.7    | unionarg . . . . .   | 618        |
| 12.2      | &#xb798;&#xc2a4;&#xd130; &#xad00;&#xb9ac; . . . . .  | 619        |
| 12.2.1    | AddRasterConstraints . . . . .   | 619        |
| 12.2.2    | DropRasterConstraints . . . . .  | 621        |
| 12.2.3    | AddOverviewConstraints . . . . .   | 622        |
| 12.2.4    | DropOverviewConstraints . . . . .  | 623        |
| 12.2.5    | PostGIS_GDAL_Version . . . . .   | 624        |
| 12.2.6    | PostGIS_Raster_Lib_Build_Date . . . . .  | 624        |
| 12.2.7    | PostGIS_Raster_Lib_Version . . . . .   | 625        |
| 12.2.8    | ST_GDALDrivers . . . . .   | 625        |
| 12.2.9    | ST_Count . . . . .   | 630        |

---

|         |                        |     |
|---------|------------------------|-----|
| 12.2.10 | ST_MakeEmptyRaster     | 631 |
| 12.2.11 | UpdateRasterSRID       | 632 |
| 12.2.12 | ST_CreateOverview      | 632 |
| 12.3    | ST_Constructor         | 633 |
| 12.3.1  | ST_AddBand             | 633 |
| 12.3.2  | ST_AsRaster            | 636 |
| 12.3.3  | ST_Band                | 639 |
| 12.3.4  | ST_MakeEmptyCoverage   | 641 |
| 12.3.5  | ST_MakeEmptyRaster     | 643 |
| 12.3.6  | ST_Tile                | 644 |
| 12.3.7  | ST_Retile              | 646 |
| 12.3.8  | ST_FromGDALRaster      | 647 |
| 12.4    | ST_Accessor            | 648 |
| 12.4.1  | ST_GeoReference        | 648 |
| 12.4.2  | ST_Height              | 649 |
| 12.4.3  | ST_IsEmpty             | 649 |
| 12.4.4  | ST_MemSize             | 650 |
| 12.4.5  | ST_MetaData            | 651 |
| 12.4.6  | ST_NumBands            | 652 |
| 12.4.7  | ST_PixelHeight         | 652 |
| 12.4.8  | ST_PixelWidth          | 653 |
| 12.4.9  | ST_ScaleX              | 655 |
| 12.4.10 | ST_ScaleY              | 655 |
| 12.4.11 | ST_RasterToWorldCoord  | 656 |
| 12.4.12 | ST_RasterToWorldCoordX | 657 |
| 12.4.13 | ST_RasterToWorldCoordY | 658 |
| 12.4.14 | ST_Rotation            | 659 |
| 12.4.15 | ST_SkewX               | 660 |
| 12.4.16 | ST_SkewY               | 661 |
| 12.4.17 | ST_SRID                | 661 |
| 12.4.18 | ST_Summary             | 662 |
| 12.4.19 | ST_UpperLeftX          | 663 |
| 12.4.20 | ST_UpperLeftY          | 663 |
| 12.4.21 | ST_Width               | 664 |
| 12.4.22 | ST_WorldToRasterCoord  | 665 |
| 12.4.23 | ST_WorldToRasterCoordX | 665 |
| 12.4.24 | ST_WorldToRasterCoordY | 666 |
| 12.5    | ST_BandMetaData        | 667 |
| 12.5.1  | ST_BandMetaData        | 667 |

---

|         |                       |     |
|---------|-----------------------|-----|
| 12.5.2  | ST_BandNoDataValue    | 669 |
| 12.5.3  | ST_BandIsNoData       | 669 |
| 12.5.4  | ST_BandPath           | 671 |
| 12.5.5  | ST_BandFileSize       | 672 |
| 12.5.6  | ST_BandFileTimestamp  | 672 |
| 12.5.7  | ST_BandPixelType      | 673 |
| 12.5.8  | ST_PixelOfValue       | 673 |
| 12.5.9  | ST_HasNoBand          | 674 |
| 12.6    | ST_PixelAsPolygon     | 675 |
| 12.6.1  | ST_PixelAsPolygons    | 676 |
| 12.6.2  | ST_PixelAsPoint       | 677 |
| 12.6.3  | ST_PixelAsPoints      | 677 |
| 12.6.4  | ST_PixelAsCentroid    | 679 |
| 12.6.5  | ST_PixelAsCentroids   | 679 |
| 12.6.6  | ST_Value              | 680 |
| 12.6.7  | ST_NearestValue       | 684 |
| 12.6.8  | ST_SetSkew            | 686 |
| 12.6.9  | ST_SetSkew            | 687 |
| 12.6.10 | ST_Neighborhood       | 688 |
| 12.6.11 | ST_SetValue           | 691 |
| 12.6.12 | ST_SetValues          | 692 |
| 12.6.13 | ST_DumpValues         | 700 |
| 12.6.14 | ST_PixelOfValue       | 701 |
| 12.6.15 | ST_SetGeoReference    | 703 |
| 12.7    | ST_SetRotation        | 704 |
| 12.7.1  | ST_SetScale           | 705 |
| 12.7.2  | ST_SetSkew            | 706 |
| 12.7.3  | ST_SetSRID            | 707 |
| 12.7.4  | ST_SetUpperLeft       | 708 |
| 12.7.5  | ST_Resample           | 708 |
| 12.7.6  | ST_Rescale            | 710 |
| 12.7.7  | ST_Reskew             | 711 |
| 12.7.8  | ST_SnapToGrid         | 713 |
| 12.7.9  | ST_Resize             | 714 |
| 12.7.10 | ST_Transform          | 716 |
| 12.7.11 | ST_SetBandNoDataValue | 719 |
| 12.8    | ST_SetBandNoDataValue | 719 |
| 12.8.1  |                       |     |

|          |  |     |
|----------|--|-----|
| 12.8.2   | ST_SetBandIsNoData   | 720 |
| 12.8.3   | ST_SetBandPath   | 722 |
| 12.8.4   | ST_SetBandIndex  | 723 |
| 12.9     | &#xb798;&#xc2a4;&#xd130; &#xbc34;&#xb4dc; &#xd1b5;&#xacc4; &#xbc0f; &#xbd84;&#xc11d; | 725 |
| 12.9.1   | ST_Count   | 725 |
| 12.9.2   | ST_CountAgg  | 726 |
| 12.9.3   | ST_Histogram   | 727 |
| 12.9.4   | ST_Quantile  | 729 |
| 12.9.5   | ST_SummaryStats  | 731 |
| 12.9.6   | ST_SummaryStatsAgg   | 733 |
| 12.9.7   | ST_ValueCount  | 735 |
| 12.10    | Raster Inputs  | 738 |
| 12.10.1  | ST_RastFromWKB   | 738 |
| 12.10.2  | ST_RastFromHexWKB  | 739 |
| 12.11    | &#xb798;&#xc2a4;&#xd130; &#xcd9c;&#xb825;  | 739 |
| 12.11.1  | ST_AsBinary/ST_AsWKB   | 739 |
| 12.11.2  | ST_AsHexWKB  | 740 |
| 12.11.3  | ST_AsGDALRaster  | 741 |
| 12.11.4  | ST_AsJPEG  | 742 |
| 12.11.5  | ST_AsPNG   | 744 |
| 12.11.6  | ST_AsTIFF  | 745 |
| 12.12    | &#xb798;&#xc2a4;&#xd130; &#xacf5;&#xac04; &#xcc98;&#xb9ac;                           | 746 |
| 12.12.1  | ST_Clip  | 746 |
| 12.12.2  | ST_ColorMap  | 750 |
| 12.12.3  | ST_Grayscale   | 754 |
| 12.12.4  | ST_Intersection  | 756 |
| 12.12.5  | &#xb0b4;&#xc7a5; &#xb9f5; &#xb300;&#xc218; &#xcf5c;&#xbc31; &#xd568;&#xc218;         | 758 |
| 12.12.6  | ST_MapAlgebraExpr  | 766 |
| 12.12.7  | ST_MapAlgebraExpr  | 769 |
| 12.12.8  | ST_MapAlgebraExpr  | 773 |
| 12.12.9  | ST_MapAlgebraFct   | 778 |
| 12.12.10 | ST_MapAlgebraFct   | 783 |
| 12.12.11 | ST_MapAlgebraFctNgb  | 787 |
| 12.12.12 | ST_Reclass   | 790 |
| 12.12.13 | ST_Union   | 792 |
| 12.13    | &#xb0b4;&#xc7a5; &#xb9f5; &#xb300;&#xc218; &#xcf5c;&#xbc31; &#xd568;&#xc218;         | 794 |
| 12.13.1  | ST_Distinct4ma   | 794 |
| 12.13.2  | ST_InvDistWeight4ma  | 795 |
| 12.13.3  | ST_Max4ma  | 796 |

---

|   |     |
|---|-----|
| 12.13.4 ST_Mean4ma . . . . .  | 797 |
| 12.13.5 ST_Min4ma . . . . .   | 799 |
| 12.13.6 ST_MinDist4ma . . . . .   | 800 |
| 12.13.7 ST_Range4ma . . . . .   | 800 |
| 12.13.8 ST_StdDev4ma . . . . .  | 802 |
| 12.13.9 ST_Sum4ma . . . . .   | 803 |
| 12.14 &#xb798;&#xc2a4;&#xd130; &#xacf5;&#xac04; &#xcc98;&#xb9ac; . . . . .  | 804 |
| 12.14.1 ST_Aspect . . . . .   | 804 |
| 12.14.2 ST_HillShade . . . . .  | 806 |
| 12.14.3 ST_Roughness . . . . .  | 808 |
| 12.14.4 ST_Slope . . . . .  | 809 |
| 12.14.5 ST_TPI . . . . .  | 811 |
| 12.14.6 ST_TRI . . . . .  | 811 |
| 12.15 &#xb798;&#xc2a4;&#xd130;&#xb97c; &#xb3c4;&#xd615;&#xc73c;&#xb85c; . . . . .   | 812 |
| 12.15.1 Box3D . . . . .   | 812 |
| 12.15.2 ST_ConvexHull . . . . .   | 813 |
| 12.15.3 ST_DumpAsPolygons . . . . .   | 814 |
| 12.15.4 ST_Envelope . . . . .   | 816 |
| 12.15.5 ST_MinConvexHull . . . . .  | 816 |
| 12.15.6 ST_Polygon . . . . .  | 817 |
| 12.16 &#xb798;&#xc2a4;&#xd130; &#xc5f0;&#xc0b0;&#xc790; . . . . .   | 819 |
| 12.16.1 && . . . . .  | 819 |
| 12.16.2 &< . . . . .  | 820 |
| 12.16.3 &> . . . . .  | 821 |
| 12.16.4 = . . . . .   | 821 |
| 12.16.5 @ . . . . .   | 822 |
| 12.16.6 ~= . . . . .  | 823 |
| 12.16.7 ~ . . . . .   | 823 |
| 12.17 &#xb798;&#xc2a4;&#xd130; &#xbc0f; &#xb798;&#xc2a4;&#xd130; &#xbc34;&#xb4dc;&#xc758; &#xacf5;&#xac04; &#xad00;&#xacc4;&#xc131; . . . . . | 824 |
| 12.17.1 ST_Contains . . . . .   | 824 |
| 12.17.2 ST_ContainsProperly . . . . .   | 825 |
| 12.17.3 ST_Covers . . . . .   | 826 |
| 12.17.4 ST_CoveredBy . . . . .  | 827 |
| 12.17.5 ST_Disjoint . . . . .   | 828 |
| 12.17.6 ST_Intersects . . . . .   | 830 |
| 12.17.7 ST_Overlaps . . . . .   | 831 |
| 12.17.8 ST_Touches . . . . .  | 832 |
| 12.17.9 ST_SameAlignment . . . . .  | 833 |

---



|           |   |            |
|-----------|---|------------|
| 14.2.14   | Page_Normalize_Address                                      | 880        |
| 14.2.15   | Pprint_Addy   | 883        |
| 14.2.16   | Reverse_Geocode   | 884        |
| 14.2.17   | Topology_Load_Tiger   | 887        |
| 14.2.18   | Set_Geocode_Setting   | 889        |
| <b>15</b> | <b>PostGIS Special Functions Index</b>                      | <b>891</b> |
| 15.1      | PostGIS Aggregate Functions                                 | 891        |
| 15.2      | PostGIS Window Functions                                    | 892        |
| 15.3      | PostGIS SQL-MM Compliant Functions                          | 892        |
| 15.4      | PostGIS Geography Support Functions                         | 898        |
| 15.5      | PostGIS Raster Support Functions                            | 899        |
| 15.6      | PostGIS Geometry / Geography / Raster Dump Functions        | 909        |
| 15.7      | PostGIS Box Functions                                       | 909        |
| 15.8      | PostGIS Functions that support 3D                           | 911        |
| 15.9      | PostGIS Curved Geometry Support Functions                   | 917        |
| 15.10     | PostGIS Polyhedral Surface Support Functions                | 920        |
| 15.11     | PostGIS Function Support Matrix                             | 924        |
| 15.12     | New, Enhanced or changed PostGIS Functions                  | 932        |
| 15.12.1   | PostGIS Functions new or enhanced in 3.3                    | 932        |
| 15.12.2   | PostGIS Functions new or enhanced in 3.2                    | 932        |
| 15.12.3   | PostGIS Functions new or enhanced in 3.1                    | 933        |
| 15.12.4   | PostGIS Functions new or enhanced in 3.0                    | 935        |
| 15.12.5   | PostGIS Functions new or enhanced in 2.5                    | 936        |
| 15.12.6   | PostGIS Functions new or enhanced in 2.4                    | 938        |
| 15.12.7   | PostGIS Functions new or enhanced in 2.3                    | 939        |
| 15.12.8   | PostGIS Functions new or enhanced in 2.2                    | 941        |
| 15.12.9   | PostGIS functions breaking changes in 2.2                   | 941        |
| 15.12.10  | PostGIS Functions new or enhanced in 2.1                    | 942        |
| 15.12.11  | PostGIS functions breaking changes in 2.1                   | 942        |
| 15.12.12  | PostGIS Functions new, behavior changed, or enhanced in 2.0 | 942        |
| 15.12.13  | PostGIS Functions changed behavior in 2.0                   | 944        |
| 15.12.14  | PostGIS Functions new, behavior changed, or enhanced in 1.5 | 944        |
| 15.12.15  | PostGIS Functions new, behavior changed, or enhanced in 1.4 | 945        |
| 15.12.16  | PostGIS Functions new in 1.3                                | 945        |
| <b>16</b> | <b>Reporting Problems</b>                                   | <b>946</b> |
| 16.1      | Reporting Software Bugs                                     | 946        |
| 16.2      | Reporting Documentation Issues                              | 946        |

|  |            |
|--|------------|
| <b>A Appendix</b>                            | <b>947</b> |
| A.1 PostGIS 3.3.5                            | 947        |
| A.1.1 Bug Fixes and Enhancements             | 947        |
| A.2 PostGIS 3.3.4                            | 947        |
| A.2.1 Bug Fixes and Enhancements             | 947        |
| A.3 PostGIS 3.3.3                            | 948        |
| A.3.1 Bug Fixes and Enhancements             | 948        |
| A.4 PostGIS 3.3.2                            | 948        |
| A.4.1 Bug and Security Fixes                 | 949        |
| A.5 PostGIS 3.3.1                            | 949        |
| A.5.1 Bug Fixes                              | 949        |
| A.6 PostGIS 3.3.0                            | 949        |
| A.6.1 New features                           | 950        |
| A.6.2 Breaking Changes                       | 950        |
| A.6.3 Enhancements                           | 950        |
| A.6.4 Bug Fixes                              | 951        |
| A.7 PostGIS 3.3.0rc2                         | 951        |
| A.7.1 Bug Fixes                              | 951        |
| A.8 PostGIS 3.3.0rc1                         | 951        |
| A.8.1 Bug Fixes                              | 952        |
| A.9 PostGIS 3.3.0beta2                       | 952        |
| A.9.1 New Features                           | 952        |
| A.9.2 Enhancements                           | 952        |
| A.9.3 Bug Fixes                              | 952        |
| A.10 PostGIS 3.3.0beta1                      | 953        |
| A.10.1 Enhancements                          | 953        |
| A.10.2 New features                          | 953        |
| A.10.3 Bug Fix                               | 953        |
| A.11 PostGIS 3.3.0alpha1                     | 953        |
| A.11.1 Breaking changes                      | 954        |
| A.11.2 Enhancements                          | 954        |
| A.11.3 New features                          | 954        |
| A.11.4 Bug Fix                               | 954        |
| A.12 PostGIS 3.2.0 (Olivier Courtin Edition) | 955        |
| A.12.1 Breaking changes                      | 955        |
| A.12.2 Enhancements                          | 955        |
| A.12.3 New features                          | 956        |
| A.13 PostGIS 3.2.0beta3                      | 957        |
| A.13.1 Breaking changes / fixes              | 957        |

---

---

|                                       |     |
|---------------------------------------|-----|
| A.14 Release 3.2.0beta2               | 957 |
| A.14.1 Breaking changes / fixes       | 957 |
| A.14.2 Enhancements                   | 958 |
| A.15 Release 3.2.0beta1               | 958 |
| A.15.1 Bug Fixes and Breaking Changes | 958 |
| A.15.2 Enhancements                   | 958 |
| A.16 Release 3.2.0alpha1              | 958 |
| A.16.1 Breaking changes               | 958 |
| A.16.2 Enhancements                   | 959 |
| A.16.3 New features                   | 959 |
| A.17 Release 3.1.0beta1               | 960 |
| A.17.1 Breaking changes               | 960 |
| A.17.2 Enhancements                   | 960 |
| A.18 Release 3.1.0alpha3              | 960 |
| A.18.1 Breaking changes               | 960 |
| A.18.2 New features                   | 961 |
| A.18.3 Enhancements                   | 961 |
| A.18.4 Bug Fixes                      | 961 |
| A.19 Release 3.1.0alpha2              | 961 |
| A.19.1 New Features                   | 962 |
| A.19.2 Enhancements                   | 962 |
| A.19.3 Bug fixes                      | 962 |
| A.20 Release 3.1.0alpha1              | 962 |
| A.20.1 Breaking Changes               | 962 |
| A.20.2 New features                   | 963 |
| A.20.3 Enhancements                   | 963 |
| A.21 Release 3.0.0                    | 963 |
| A.21.1 New Features                   | 963 |
| A.21.2 Breaking Changes               | 964 |
| A.21.3 Enhancements                   | 964 |
| A.22 Release 3.0.0rc2                 | 965 |
| A.22.1 Major highlights               | 966 |
| A.23 Release 3.0.0rc1                 | 966 |
| A.23.1 Major highlights               | 966 |
| A.24 Release 3.0.0beta1               | 966 |
| A.24.1 Major highlights               | 966 |
| A.25 Release 3.0.0alpha4              | 967 |
| A.25.1 Major highlights               | 967 |
| A.26 Release 3.0.0alpha3              | 967 |

---

---

|   |     |
|---|-----|
| A.26.1 Major highlights . . . . .             | 967 |
| A.27 Release 3.0.0alpha2 . . . . .            | 968 |
| A.27.1 Major highlights . . . . .             | 968 |
| A.28 Release 3.0.0alpha1 . . . . .            | 968 |
| A.28.1 New Features . . . . .                 | 968 |
| A.29 Release 2.5.0 . . . . .                  | 968 |
| A.29.1 New Features . . . . .                 | 969 |
| A.29.2 Breaking Changes . . . . .             | 969 |
| A.30 Release 2.4.5 . . . . .                  | 970 |
| A.30.1 Bug Fixes . . . . .                    | 970 |
| A.31 Release 2.4.4 . . . . .                  | 971 |
| A.31.1 Bug Fixes . . . . .                    | 971 |
| A.31.2 Enhancements . . . . .                 | 971 |
| A.32 Release 2.4.3 . . . . .                  | 971 |
| A.32.1 Bug Fixes and Enhancements . . . . .   | 971 |
| A.33 Release 2.4.2 . . . . .                  | 972 |
| A.33.1 Bug Fixes and Enhancements . . . . .   | 972 |
| A.34 Release 2.4.1 . . . . .                  | 972 |
| A.34.1 Bug Fixes and Enhancements . . . . .   | 972 |
| A.35 Release 2.4.0 . . . . .                  | 972 |
| A.35.1 New Features . . . . .                 | 973 |
| A.35.2 Enhancements and Fixes . . . . .       | 973 |
| A.35.3 Breaking Changes . . . . .             | 974 |
| A.36 Release 2.3.3 . . . . .                  | 974 |
| A.36.1 Bug Fixes and Enhancements . . . . .   | 974 |
| A.37 Release 2.3.2 . . . . .                  | 974 |
| A.37.1 Bug Fixes and Enhancements . . . . .   | 974 |
| A.38 Release 2.3.1 . . . . .                  | 975 |
| A.38.1 Bug Fixes and Enhancements . . . . .   | 975 |
| A.39 Release 2.3.0 . . . . .                  | 975 |
| A.39.1 Important / Breaking Changes . . . . . | 975 |
| A.39.2 New Features . . . . .                 | 975 |
| A.39.3 Bug Fixes . . . . .                    | 976 |
| A.39.4 Performance Enhancements . . . . .     | 976 |
| A.40 Release 2.2.2 . . . . .                  | 976 |
| A.40.1 New Features . . . . .                 | 977 |
| A.41 Release 2.2.1 . . . . .                  | 977 |
| A.41.1 New Features . . . . .                 | 977 |
| A.42 Release 2.2.0 . . . . .                  | 978 |

---

|   |     |
|---|-----|
| A.42.1 New Features . . . . .                 | 978 |
| A.42.2 Enhancements . . . . .                 | 979 |
| A.43 Release 2.1.8 . . . . .                  | 980 |
| A.43.1 Bug Fixes . . . . .                    | 980 |
| A.44 Release 2.1.7 . . . . .                  | 980 |
| A.44.1 Bug Fixes . . . . .                    | 980 |
| A.45 Release 2.1.6 . . . . .                  | 980 |
| A.45.1 Enhancements . . . . .                 | 980 |
| A.45.2 Bug Fixes . . . . .                    | 981 |
| A.46 Release 2.1.5 . . . . .                  | 981 |
| A.46.1 Enhancements . . . . .                 | 981 |
| A.46.2 Bug Fixes . . . . .                    | 981 |
| A.47 Release 2.1.4 . . . . .                  | 981 |
| A.47.1 Enhancements . . . . .                 | 981 |
| A.47.2 Bug Fixes . . . . .                    | 982 |
| A.48 Release 2.1.3 . . . . .                  | 982 |
| A.48.1 Important changes . . . . .            | 982 |
| A.48.2 Bug Fixes . . . . .                    | 983 |
| A.49 Release 2.1.2 . . . . .                  | 983 |
| A.49.1 Bug Fixes . . . . .                    | 983 |
| A.49.2 Enhancements . . . . .                 | 983 |
| A.50 Release 2.1.1 . . . . .                  | 984 |
| A.50.1 Important Changes . . . . .            | 984 |
| A.50.2 Bug Fixes . . . . .                    | 984 |
| A.50.3 Enhancements . . . . .                 | 984 |
| A.51 Release 2.1.0 . . . . .                  | 984 |
| A.51.1 Important / Breaking Changes . . . . . | 984 |
| A.51.2 New Features . . . . .                 | 985 |
| A.51.3 Enhancements . . . . .                 | 986 |
| A.51.4 Fixes . . . . .                        | 988 |
| A.51.5 Known Issues . . . . .                 | 989 |
| A.52 Release 2.0.5 . . . . .                  | 989 |
| A.52.1 Bug Fixes . . . . .                    | 989 |
| A.52.2 Important Changes . . . . .            | 989 |
| A.53 Release 2.0.4 . . . . .                  | 989 |
| A.53.1 Bug Fixes . . . . .                    | 989 |
| A.53.2 Enhancements . . . . .                 | 990 |
| A.53.3 Known Issues . . . . .                 | 990 |
| A.54 Release 2.0.3 . . . . .                  | 990 |

---

---

|   |      |
|---|------|
| A.54.1 Bug Fixes . . . . .                    | 991  |
| A.54.2 Enhancements . . . . .                 | 991  |
| A.55 Release 2.0.2 . . . . .                  | 991  |
| A.55.1 Bug Fixes . . . . .                    | 991  |
| A.55.2 Enhancements . . . . .                 | 992  |
| A.56 Release 2.0.1 . . . . .                  | 992  |
| A.56.1 Bug Fixes . . . . .                    | 993  |
| A.56.2 Enhancements . . . . .                 | 994  |
| A.57 Release 2.0.0 . . . . .                  | 994  |
| A.57.1 Testers - Our unsung heroes . . . . .  | 994  |
| A.57.2 Important / Breaking Changes . . . . . | 994  |
| A.57.3 New Features . . . . .                 | 995  |
| A.57.4 Enhancements . . . . .                 | 995  |
| A.57.5 Bug Fixes . . . . .                    | 996  |
| A.57.6 Release specific credits . . . . .     | 996  |
| A.58 Release 1.5.4 . . . . .                  | 996  |
| A.58.1 Bug Fixes . . . . .                    | 996  |
| A.59 Release 1.5.3 . . . . .                  | 997  |
| A.59.1 Bug Fixes . . . . .                    | 997  |
| A.60 Release 1.5.2 . . . . .                  | 997  |
| A.60.1 Bug Fixes . . . . .                    | 997  |
| A.61 Release 1.5.1 . . . . .                  | 998  |
| A.61.1 Bug Fixes . . . . .                    | 998  |
| A.62 Release 1.5.0 . . . . .                  | 998  |
| A.62.1 API Stability . . . . .                | 999  |
| A.62.2 Compatibility . . . . .                | 999  |
| A.62.3 New Features . . . . .                 | 999  |
| A.62.4 Enhancements . . . . .                 | 1000 |
| A.62.5 Bug fixes . . . . .                    | 1000 |
| A.63 Release 1.4.0 . . . . .                  | 1000 |
| A.63.1 API Stability . . . . .                | 1000 |
| A.63.2 Compatibility . . . . .                | 1000 |
| A.63.3 New Features . . . . .                 | 1000 |
| A.63.4 Enhancements . . . . .                 | 1001 |
| A.63.5 Bug fixes . . . . .                    | 1001 |
| A.64 Release 1.3.6 . . . . .                  | 1001 |
| A.65 Release 1.3.5 . . . . .                  | 1001 |
| A.66 Release 1.3.4 . . . . .                  | 1002 |
| A.67 Release 1.3.3 . . . . .                  | 1002 |

---

|  |      |
|--|------|
| A.68 Release 1.3.2                             | 1002 |
| A.69 Release 1.3.1                             | 1002 |
| A.70 Release 1.3.0                             | 1002 |
| A.70.1 Added Functionality                     | 1002 |
| A.70.2 Performance Enhancements                | 1002 |
| A.70.3 Other Changes                           | 1003 |
| A.71 Release 1.2.1                             | 1003 |
| A.71.1 Changes                                 | 1003 |
| A.72 Release 1.2.0                             | 1003 |
| A.72.1 Changes                                 | 1003 |
| A.73 Release 1.1.6                             | 1003 |
| A.73.1 &#xc5c5;&xadf8;&#xb808;&#xc774;&#xb4dc; | 1003 |
| A.73.2 Bug fixes                               | 1004 |
| A.73.3 Other changes                           | 1004 |
| A.74 Release 1.1.5                             | 1004 |
| A.74.1 &#xc5c5;&xadf8;&#xb808;&#xc774;&#xb4dc; | 1004 |
| A.74.2 Bug fixes                               | 1004 |
| A.74.3 New Features                            | 1004 |
| A.75 Release 1.1.4                             | 1004 |
| A.75.1 &#xc5c5;&xadf8;&#xb808;&#xc774;&#xb4dc; | 1005 |
| A.75.2 Bug fixes                               | 1005 |
| A.75.3 Java changes                            | 1005 |
| A.76 Release 1.1.3                             | 1005 |
| A.76.1 &#xc5c5;&xadf8;&#xb808;&#xc774;&#xb4dc; | 1005 |
| A.76.2 Bug fixes / correctness                 | 1005 |
| A.76.3 New functionalities                     | 1006 |
| A.76.4 JDBC changes                            | 1006 |
| A.76.5 Other changes                           | 1006 |
| A.77 Release 1.1.2                             | 1006 |
| A.77.1 &#xc5c5;&xadf8;&#xb808;&#xc774;&#xb4dc; | 1006 |
| A.77.2 Bug fixes                               | 1006 |
| A.77.3 New functionalities                     | 1007 |
| A.77.4 Other changes                           | 1007 |
| A.78 Release 1.1.1                             | 1007 |
| A.78.1 &#xc5c5;&xadf8;&#xb808;&#xc774;&#xb4dc; | 1007 |
| A.78.2 Bug fixes                               | 1007 |
| A.78.3 New functionalities                     | 1007 |
| A.79 Release 1.1.0                             | 1008 |
| A.79.1 Credits                                 | 1008 |

---



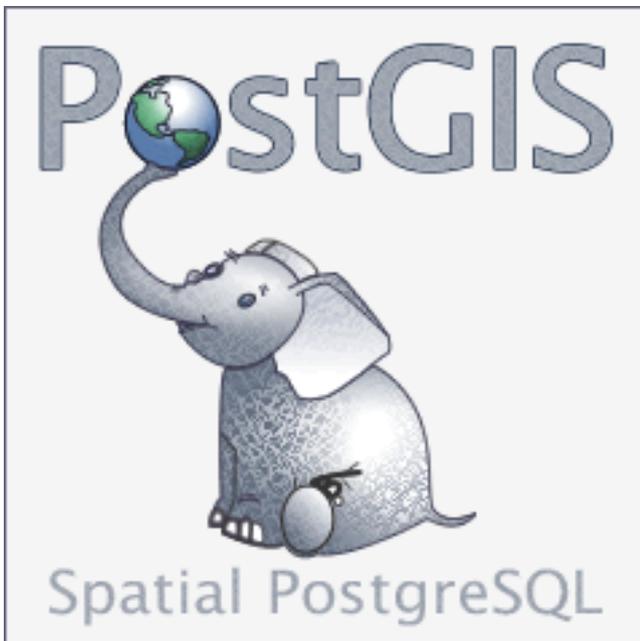
---

|   |      |
|---|------|
| A.87.2 Library changes . . . . .                          | 1014 |
| A.87.3 Scripts changes . . . . .                          | 1014 |
| A.87.4 Other changes . . . . .                            | 1015 |
| A.88 Release 1.0.0RC5 . . . . .                           | 1015 |
| A.88.1 &#xc5c5;&#xadf8;&#xb808;&#xc774;&#xb4dc; . . . . . | 1015 |
| A.88.2 Library changes . . . . .                          | 1015 |
| A.88.3 Other changes . . . . .                            | 1015 |
| A.89 Release 1.0.0RC4 . . . . .                           | 1015 |
| A.89.1 &#xc5c5;&#xadf8;&#xb808;&#xc774;&#xb4dc; . . . . . | 1015 |
| A.89.2 Library changes . . . . .                          | 1015 |
| A.89.3 Scripts changes . . . . .                          | 1016 |
| A.89.4 Other changes . . . . .                            | 1016 |
| A.90 Release 1.0.0RC3 . . . . .                           | 1016 |
| A.90.1 &#xc5c5;&#xadf8;&#xb808;&#xc774;&#xb4dc; . . . . . | 1016 |
| A.90.2 Library changes . . . . .                          | 1016 |
| A.90.3 Scripts changes . . . . .                          | 1016 |
| A.90.4 JDBC changes . . . . .                             | 1017 |
| A.90.5 Other changes . . . . .                            | 1017 |
| A.91 Release 1.0.0RC2 . . . . .                           | 1017 |
| A.91.1 &#xc5c5;&#xadf8;&#xb808;&#xc774;&#xb4dc; . . . . . | 1017 |
| A.91.2 Library changes . . . . .                          | 1017 |
| A.91.3 Scripts changes . . . . .                          | 1017 |
| A.91.4 Other changes . . . . .                            | 1018 |
| A.92 Release 1.0.0RC1 . . . . .                           | 1018 |
| A.92.1 &#xc5c5;&#xadf8;&#xb808;&#xc774;&#xb4dc; . . . . . | 1018 |
| A.92.2 Changes . . . . .                                  | 1018 |

---

## Abstract

PostGIS &#x294; &#xac1d;&#xccb4; &#xad00;&#xacc4;&#xd615; &#xb370;&#xc774;&#xd130;&#xbca0;&#xc774;&#xc2a4; &#xc2dc;&#xc2a4;&#xd15c;&#xc778; PostgreSQL &#xc758; &#xd655;&#xc7a5; &#xd504;&#xb85c;&#xadf8;&#xb7a8;&#xc73c;&#xb370;&#xc774;&#xd130;&#xbca0;&#xc774;&#xc2a4;&#xc5d0; GIS(&#xc9c0;&#xb9ac;&#xc815;&#xbcf4; &#xc2dc;&#xc2a4; &#xac1d;&#xccb4;&#xb97c; &#xc800;&#xc7a5;&#xd560; &#xc218; &#xc788;&#xac8c; &#xd574;&#xc90d;&#xb2c8;&#xb2e4; PostGIS&#xb294; GiST &#xae30;&#xbc18; R-Tree &#xaf5;&#xac04; &#xc778;&#xb371;&#xc2a4;&#xb97c; &#xc9c0;&#xc6d0;&#xc774; GIS &#xac1d;&#xccb4;&#xc758; &#xbd84;&#xc11d; &#xbc0f; &#xaf5;&#xac04; &#xc98;&#xb9ac;&#xb97c; &#xc704;&#xd55c; &#xae30;&#xb2a5;&#xc744; &#xd3ec;&#xd568;&#xd558;&#xace0; &#xc788;&#xc2b5;&#xb2c8;&#xb2e4;.



&#xc774; &#xbb38;&#xc11c;&#xb294; 3.3.6dev &#xbc84;&#xc804;&#xc758; &#xc0ac;&#xc6a9;&#xc790; &#xc9c0;&#xce68;&#xc774;



&#xc774; &#xc800;&#xc791;&#xbb3c;&#xc740; &#xd06c;&#xb9ac;&#xc5d0;&#xc774;&#xd2f0;&#xbe0c; &#xcee4;&#xba3c;&#xc988; &#xc800;&#xc791;&#xc790;&#xd45c;&#xc2dc;-&#xb3d9;&#xc77c;&#xc870;&#xac74;&#xbcc0;&#xc774; 3.0 &#xb77c;&#xc774;&#xc120;&#xc2a4;&#xb97c; &#xb530;&#xb985;&#xb2c8;&#xb2e4;. &#xc0ac;&#xc6a9;&#xc790;&#xac00; &#xc6d0;&#xd558;&#xb294; &#xb300;&#xb85c; &#xc774; &#xbb38;&#xc11c;&#xb97c; &#xc0ac;&#xc6a9;&#xd560; &#xc218; &#xc788;&#xc9c0;&#xb9cc;., &#xc6b0;&#xb9ac;&#xb294; &#xc0ac;&#xc6a9;&#xc790;&#xac00; PostGIS &#xd504;&#xb85c;&#xc800;&#xc791;&#xad8c;&#xc744; &#xba85;&#xc2dc;&#xd558;&#xace0; &#xac00;&#xb2a5;&#xd55c; &#xc704;&#xce58;&#xc774; <http://postgis.net>&#xc73c;&#xb85c; &#xc5f0;&#xacb0;&#xb418;&#xb294; &#xb9c1;&#xd06c;&#xb97c; &#xc0bd;&#xc785;&#xd55c; &#xc694;&#xad6c;&#xd569;&#xb2c8;&#xb2e4;.



## 1.2

**Nicklas Avén** Geometry clustering function additions, other geometry algorithm enhancements, GEOS enhancements and general user support

**Loïc Bartoletti** SFCGAL enhancements and maintenance and ci support

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

**Martin Davis** GEOS enhancements and documentation

**Björn Harrtell** MapBox Vector Tile and GeoBuf functions. Gogs testing and GitLab experimentation.

**Aliaksandr Kalenik** Geometry Processing, PostgreSQL gist, general bug fixing

## 1.3

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

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

**Jorge Arévalo** GDAL integration with GDAL, raster loader, user support, general bug fixing, testing on various OS (Slackware, Mac, Windows, and more)

**Olivier Courtin** XML(KML, GML)/GeoJSON support, 3D support, general bug fixing, testing on various OS (Slackware, Mac, Windows, and more)

**Chris Hodgson** PSC Member. Coordinated bug fixing and maintenance effort, spatial index selectivity and binding, loader/dumper, and Shapefile GUI Loader, integration of new and new function enhancements.

**Mateusz Loskot** PostGIS CI, CMake integration, API improvements, general bug fixing, testing on various OS (Slackware, Mac, Windows, and more)

**Kevin Neufeld** PSC Member. Coordinated bug fixing and maintenance effort, spatial index selectivity and binding, loader/dumper, and Shapefile GUI Loader, integration of new and new function enhancements.

**Dave Blasby** PostGIS CI, CMake integration, API improvements, general bug fixing, testing on various OS (Slackware, Mac, Windows, and more)

**Jeff Lounsbury** shapefile support, general bug fixing, testing on various OS (Slackware, Mac, Windows, and more)

**Mark Leslie** shapefile GUI support, general bug fixing, testing on various OS (Slackware, Mac, Windows, and more)

**&#xd53c;&#xc5d0;&#xb974; &#xb77c;&#xc2e0;(Pierre Racine)** Architect of PostGIS raster implementation. Raster overall architecture, prototyping, programming support

**&#xb2e4;&#xd53c;&#xd2b8; &#xce20;&#xbc14;&#xb974;&#xd06c;(David Zwarg)** &#xb798;&#xc2a4;&#xd130; &#xac1c;&#xb9f5; &#xb300;&#xc218;&#xd559; &#xbd84;&#xc11d; &#xae30;&#xb2a5;&#xb4e4;)&#xc744; &#xb2f4;&#xb2f9;

## 1.4 &#xae30;&#xd0c0; &#xacf5;&#xd5cc;&#xc790;

Alex Bodnaru  
 Alex Mayrhofer  
 Andrea Peri  
 Andreas Forø Tollefsen  
 Andreas Neumann  
 Andrew Gierth  
 Anne Ghisla  
 Antoine Bajolet  
 Arthur Lesuisse  
 Artur Zakirov  
 Barbara Phillipot  
 Ben Jubb  
 Bernhard Reiter  
 Björn Esser  
 Brian Hamlin  
 Bruce Rindahl  
 Bruno Wolff III  
 Bryce L. Nordgren  
 Carl Anderson  
 Charlie Savage  
 Christoph Berg  
 Christoph Moench-Tegeder  
 Dane Springmeyer  
 Dave Fuhry  
 &#xb2e4;&#xd53c;&#xd2b8; &#xce20;&#xbc14;&#xb974;&#xd06c;(David Zwarg)  
 &#xb2e4;&#xd53c;&#xd2b8; &#xce20;&#xbc14;&#xb974;&#xd06c;(David Zwarg)  
 &#xb2e4;&#xd53c;&#xd2b8; &#xce20;&#xbc14;&#xb974;&#xd06c;(David Zwarg)  
 Dmitry Vasilyev  
 Eduin Carrillo  
 Eugene Antimirov  
 Even Rouault  
 Frank Warmerdam  
 George Silva  
 Gerald Fenoy  
 Gino Lucrezi

&#xac1c;&#xc778; &#xacf5;&#xd5cc;&#xc790;

**&#xd6c4;&#xc6d0; &#xae30;&#xc5c5;** PostGIS &#xd504;&#xb85c;&#xc81d;&#xd2b8;&#xc5d0; &#xc9c1;&#xc811;&#xc801;&#xae08;&#xc804;&#xc744; &#xd6c4;&#xc6d0;&#xd558;&#xac70;&#xb098;, &#xac1c;&#xbc1c;&#xc790; &#xc2dc;&#xacf5;&#xb0f; &#xd638;&#xc2a4;&#xd305;&#xc5d0; &#xae30;&#xc5ec;&#xb97c; &#xd55c; &#xae30;&#xc5c5;&#xb4e4;&#xc78

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

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

**&#xd06c;&#xb77c;&#xc6b0;&#xb4dc; &#xd380;&#xb529; &#xcea0;&#xd398;&#xc778;** &#xd06c;&#xb77c;&#xc6b0;&#xb4dc;&#xd380;&#xb529; &#xcea0;&#xd398;&#xc778;&#xc774;&#xb780; &#xc218;&#xb9ce;&#xc740; &#xc0ac;&#xb78c;&#xb4c;&#xc11c;&#xbe44;&#xc2a4;&#xd560; &#xc218; &#xc788;&#xb294;., &#xc6b0;&#xb9ac;&#xac00; &#xac04;&#xc808;&#xd7&#xc6d0;&#xd558;&#xb294; &#xae30;&#xb2a5;&#xb4e4;&#xc744; &#xd6c4;&#xc6d0;&#xbc1b;&#xae30; &#xc704;&#xd5&#xc9c4;&#xd589;&#xd558;&#xb294; &#xcea0;&#xd398;&#xc778;&#xc785;&#xb2c8;&#xb2e4;. &#xac01; &#xcea0;&#xd3&#xd2b9;&#xc815; &#xae30;&#xb2a5; &#xb610;&#xb294; &#xc77c;&#xb828;&#xc758; &#xae30;&#xb2a5;&#xc5d0;&#xd2b9;&#xd654;&#xb418;&#xc5b4; &#xc788;&#xc2b5;&#xb2c8;&#xb2e4;. &#xac01; &#xd6c4;&#xc6d0;&#xc790;&#xb&#xd544;&#xc694;&#xd55c; &#xd380;&#xb529;&#xc758; &#xc791;&#xc740; &#xc77c;&#xbd80;&#xbd84;&#xc744;



## Chapter 2

# PostGIS

PostGIS is a spatial database system that provides the ability to store and retrieve geographic objects directly into a database table. It is an extension of PostgreSQL, which provides the underlying database engine.

### 2.1 Installing PostGIS

This section describes how to install PostGIS on a Linux system. The instructions are for the development version (3.3.6dev) and may vary slightly from the stable release.

```
tar xvfz postgis-3.3.6dev.tar.gz
cd postgis-3.3.6dev
./configure
make
make install
```

PostGIS is installed into the `/usr/local` directory. The installation process will create a `postgis` user and a `postgis` group. You will need to add your user to the `postgis` group and set the `PGDATA` environment variable to the path where the data files are stored.



PostGIS 3.3.6dev

## 2.2.1

PostGIS 3.3.6dev

```
wget http://postgis.net/stuff/postgis-3.3.6dev.tar.gz
tar -xvzf postgis-3.3.6dev.tar.gz
cd postgis-3.3.6dev
```

```
postgis-3.3.6dev (PostGIS)
PostGIS 3.3.6dev
PostGIS 3.3.6dev
```

```
PostGIS 3.3.6dev
PostGIS 3.3.6dev
PostGIS 3.3.6dev
```

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

```
PostGIS 3.3.6dev
PostGIS 3.3.6dev
```

```
./configure
```

## 2.2.2

PostGIS 3.3.6dev

PostGIS 3.3.6dev

- PostgreSQL 11 - 16. A complete installation of PostgreSQL (including server headers) is required. PostgreSQL is available from <http://www.postgresql.org>.

```
PostgreSQL/PostGIS
PostgreSQL/PostGIS
PostgreSQL/PostGIS
```

- GNU C (gcc). PostGIS 3.3.6dev

- GNU Make (gmake, make). PostGIS 3.3.6dev

- Proj reprojection library. Proj 4.9 or above is required. The Proj library is used to provide coordinate reprojection support within PostGIS. Proj is available for download from <https://proj.org/>.

- GEOS geometry library, version 3.6 or greater, but GEOS 3.11+ is required to take full advantage of all the new functions and features. GEOS is available for download from <https://libgeos.org/>.
- LibXML2, version 2.5.x or higher. LibXML2 is currently used in some imports functions (ST\_GeomFromGML and ST\_GeomFromKML). LibXML2 is available for download from <https://gitlab.gnome.org/GNOME/libxml2/-/releases>.
- JSON-C 0.9. JSON-C and ST\_GeomFromGeoJSON. JSON-C is available for download from <https://github.com/json-c/json-c/releases/>.
- GDAL, version 2+ is required 3+ is preferred. This is required for raster support. <https://gdal.org/download.html>.

### Dependencies

- PostgreSQL 9.5 or higher. PostgreSQL 9.5 or higher is required for raster support. <http://trac.osgeo.org/postgis/ticket/635>.
- Shapefile. <http://www.gtk.org/>.
- SFCGAL, version 1.3.1 (or higher), 1.4.1 or higher is recommended. SFCGAL can be used to provide additional 2D and 3D advanced analysis functions to PostGIS cf Section 8.20. And also allow to use SFCGAL rather than GEOS for some 2D functions provided by both backends (like ST\_Intersection or ST\_Area, for instance). A PostgreSQL configuration variable `postgis.backend` allow end user to control which backend he want to use if SFCGAL is installed (GEOS by default). Nota: SFCGAL 1.2 require at least CGAL 4.3 and Boost 1.54 (cf: <https://sfcgal.org> <https://gitlab.com/sfcgal/SFCGAL/>).
- Section 14.1: PCRE. `parseaddress-stcities.h`. `Regex`: `::Assemble`. `CPAN`. Section 14.1: PCRE. `Wagyu`. `--with-pcre-dir=/path/to/pcre`.
- To enable ST\_AsMVT protobuf-c library 1.1.0 or higher (for usage) and the protoc-c compiler (for building) are required. Also, pkg-config is required to verify the correct minimum version of protobuf-c. See [protobuf-c](#). By default, Postgis will use Wagyu to validate MVT polygons faster which requires a c++11 compiler. It will use CXXFLAGS and the same compiler as the PostgreSQL installation. To disable this and use GEOS instead use the `--without-wagyu` during the configure step.
- CUnit(CUnit). <http://cunit.sourceforge.net/>
- DocBook(xsltproc). <http://www.docbook.org/>
- DBLatex(dblatex). PDF. <http://dblatex.sourceforge.net/>
- ImageMagick(convert). <http://www.imagemagick.org/>

## 2.2.3

&#xb300;&#xbd80;&#xbd84;&#xc758; &#xb9ac;&#xb205;&#xc2a4; &#xc124;&#xce58;&#xc5d0;&#xc11c; &#xccab; &#xbc88;&#xb2e8;&#xac4;&#xb294; &#xc18c;&#xc2a4; &#xcf54;&#xb4dc;&#xb97c; &#xbe4c;&#xb4dc;&#xd558;&#xb294; &#xb370; &#xc0ac;&#xc6a9;&#xd560; Makefile&#xc744; &#xc0dd;&#xc131;&#xd558;&#xb294; &#xac83;&#xc785;&#xb2c8;&#xb2e4;. &#xc178; &#xc2a4;&#xd06c;&#xb9bd;&#xd2b8;&#xb97c; &#xc2e4;&#xd589;&#xd574;&#xc11c; Makefile&#xc744; &#xc791;&#xc744;

### ./configure

&#xcd94;&#xac00; &#xd30c;&#xb77c;&#xbbf8;&#xd130; &#xc5c6;&#xc774; &#xc0ac;&#xc6a9;&#xd558;&#xba74;. &#xc774; &#xba85;&#xb839;&#xc740; &#xc790;&#xb3d9;&#xc801;&#xc73c;&#xb85c; PostGIS &#xc18c;&#xc2a4;&#xcf54;&#xb4dc;&#xb97c; &#xc0ac;&#xc6a9;&#xc790; &#xc2dc;&#xc2a4;&#xd15c;&#xc5d0; &#xbe4c;&#xb4dc;&#xd558;&#xb294; &#xb370; &#xd544;&#xc218; &#xad6c;&#xc131; &#xc694;&#xc18c;&#xb4e4;&#xacfc; &#xb77c;&#xc774;&#xbe0c;&#xb7ec;&#xb9ac;&#xc704;&#xce58;&#xb97c; &#xd655;&#xc778;&#xd558;&#xb824; &#xd569;&#xb2c8;&#xb2e4;. **./configure** &#xba85;&#xb839; &#xac00;&#xc7a5; &#xd754;&#xd55c; &#xc0ac;&#xc6a9;&#xbc95;&#xc774;&#xae34; &#xd558;&#xc9c0;&#xb9cc;. &#xc774; &#xc2a4;&#xd06c;&#xb9bd;&#xd2b8;&#xb294; &#be44;&#xd45c;&#xc900;&#xc801;&#xc778; &#xc704;&#xce58;&#xc5d0; &#xc788;&#xb294; &#xd544;&#xc218; &#xb77c;&#xc774;&#xbe0c;&#xb7ec;&#xb9ac;&#xc640; &#xd504;&#xb85c;&#xadf8;&#xb300;&#xd55c; &#xba87;&#xba87; &#xd30c;&#xb77c;&#xbbf8;&#xd130;&#xb97c; &#xbc1b;&#xc2b5;&#xb2c8;&#xb2e4;.

&#xb2e4;&#xc74c;&#xc740; &#xac00;&#xc7a5; &#xb9ce;&#xc774; &#xc0ac;&#xc6a9;&#xb418;&#xb294; &#xd30c;&#xb77c;&#xb098;&#xc5f4;&#xd55c; &#xbaa9;&#xb85d;&#xc785;&#xb2c8;&#xb2e4;. &#xc804;&#xccb4; &#xbaa9;&#xb85d;&#xc740; **--help** &#xb610;&#xb294; **--help=short** &#xd30c;&#xb77c;&#xbbf8;&#xd130;&#xb97c; &#xc0ac;&#xc6a9;&#xd558;&#xc2ed;&#xc740;

**--with-library-minor-version** Starting with PostGIS 3.0, the library files generated by default will no longer have the minor version as part of the file name. This means all PostGIS 3 libs will end in `postgis-3`. This was done to make `pg_upgrade` easier, with downside that you can only install one version PostGIS 3 series in your server. To get the old behavior of file including the minor version: e.g. `postgis-3.0` add this switch to your configure statement.

**--prefix=PREFIX** PostGIS &#xb77c;&#xc774;&#xbe0c;&#xb7ec;&#xb9ac;&#xc640; SQL &#xc2a4;&#xd06c;&#xb9bd;&#xd2b8;&#xc124;&#xce58;&#xb420; &#xacbd;&#xb85c;&#xc785;&#xb2c8;&#xb2e4;. &#xae30;&#xbcf8;&#xc801;&#xc73c;&#xb85c; PostgreSQL&#xac00; &#xc124;&#xce58;&#xb41c; &#xacbd;&#xb85c;&#xc640; &#xac19;&#xc740; &#acf3;&#xc5d0; &#xc124;&#xce58;&#xb429;&#xb2c8;&#xb2e4;.

### Caution



&#xc774; &#xd30c;&#xb77c;&#xbbf8;&#xd130;&#xb294; &#xd604;&#xc7ac; &#xc791;&#xb3d9;&#xb418;&#xc9c0; &#xc54a;&#xc2b5;&#xb2c8;&#xb2e4;. &#xd328;&#xd0a4;&#xc9c0;&#xac00; PostgreSQL &#xc124;&#xce58; &#xacbd;&#xb85c;&#xc5d0;&#xb9cc; &#xc124;&#xce58;&#xb420; &#xac83;&#xc774;&#xae30; &#xb54c;&#xbb38;&#xc785;&#xb2c8;&#xb2e4;. &#xd574;&#xb2f9; &#xbc84;&#xadf8;&#xb97c; &#xcd94;&#xc801;&#xd558;&#xb824;&#xba74; <http://trac.osgeo.org/postgis/ticket/635> &#xb97c; &#xcc38;&#xc870;&#xd558;&#xc2ed;&#xc2dc;&#xc624;.

**--with-pgconfig=FILE** PostgreSQL&#xc740; PostGIS &#xac19;&#xc740; &#xd655;&#xc7a5; &#xd504;&#xb85c;&#xadf8;&#xb7a; PostgreSQL &#xc124;&#xce58; &#xb514;&#xb809;&#xd1a0;&#xb9ac;&#xc758; &#xc704;&#xce58;&#xb97c; &#xd655;&#xd574;&#xc8fc;&#xb294; **pg\_config** &#xb77c;&#xb294; &#xc720;&#xd2f8;&#xb9ac;&#xd2f0;&#xb97c; &#xc81c;&#xacf PostGIS &#xbe4c;&#xb4dc; &#xc2dc; &#xb300;&#xc0c1;&#xc774; &#xb420; &#xd2b9;&#xc815; PostgreSQL &#xc124;&#xb514;&#xb809;&#xd130;&#xb9ac;&#xb97c; &#xc0ac;&#xc6a9;&#xc790;&#xac00; &#xc9c1;&#xc811; &#xc124;&#xc81 &#xc774; &#xd30c;&#xb77c;&#xbbf8;&#xd130;(**--with-pgconfig=/path/to/pg\_config**)&#xb97c; &#xc0ac;&#xc6a9;&#xd558;

**--with-gdalconfig=FILE** GDAL&#xc740; &#xb798;&#xc2a4;&#xd130; &#xc9c0;&#xc6d0;&#xc744; &#xc704;&#xd55c; &#xae30; &#xc81c;&#xacf5;&#xd558;&#xb294; &#xd544;&#xc218; &#xb77c;&#xc774;&#xbe0c;&#xb7ec;&#xb9ac;&#xb85c;. GDAL &#xc124;&#xce58; &#xb514;&#xb809;&#xd130;&#xb9ac;&#xc758; &#xc704;&#xce58;&#xb97c; &#xd655;&#xc778; &#xc18c;&#xd504;&#xd2b8;&#xc6e8;&#xc5b4; &#xc124;&#xce58;&#xb97c; &#xd65c;&#xc131;&#xd654;&#xd558;&#xae3 &#xc704;&#xd55c; **gdal-config**&#xb97c; &#xc9c0;&#xc6d0;&#xd569;&#xb2c8;&#xb2e4;. PostGIS &#xbe4c;&#xb4dc; &#xc2dc; &#xb300;&#xc0c1;&#xc774; &#xb420; &#xd2b9;&#xc815; GDAL &#xc124;&#xce58; &#xb514;&#xb809;&#xd130 &#xc0ac;&#xc6a9;&#xc790;&#xac00; &#xc9c1;&#xc811; &#xc124;&#xc815;&#xd558;&#xb824;&#xba74; &#xc774; &#xd30c;&#xb77c;&#xbbf8;&#xd130;(**--with-gdalconfig=/path/to/gdal-config**)&#xb97c; &#xc0ac;&#xc6a9;&#xd558;&#xc2

- with-geosconfig=FILE** GEOS &#xb294; &#xd544;&#xc218; &#xb3c4;&#xd615; &#xb77c;&#xc774;&#xbe0c;&#xb7ec;&#xb9ac; GEOS &#xc124;&#xce58; &#xb514;&#xb809;&#xd130;&#xb9ac;&#xc758; &#xc704;&#xce58;&#xb97c; &#xd655;&#xc778; &#xc18c;&#xd504;&#xd2b8;&#xc6e8;&#xc5b4; &#xc124;&#xce58;&#xb97c; &#xd65c;&#xc131;&#xd654;&#xd558;&#xae3 &#xc704;&#xd55c; **geos-config**&#xb77c;&#xb294; &#xc720;&#xd2f8;&#xb9ac;&#xd2f0;&#xb97c; &#xc81c;&#xacf5;&#xd5 PostGIS &#xbe4c;&#xb4dc; &#xc2dc; &#xb300;&#xc0c1;&#xc774; &#xb420; &#xd2b9;&#xc815; GEOS &#xc124;&#xce58; &#xb514;&#xb809;&#xd130;&#xb9ac;&#xb97c; &#xc0ac;&#xc6a9;&#xc790;&#xac00; &#xc9c1;&#xc811; &#xc124;&#xc81 &#xc774; &#xd30c;&#xb77c;&#xbbf8;&#xd130;(**--with-geosconfig=/path/to/geos-config**)&#xb97c; &#xc0ac;&#xc6a9;&#xd558;&#xc704;&#xd574; &#xd544;&#xc694;&#xd55c; &#xb77c;&#xc774;&#xbe0c;&#xb7ec;&#xb9ac;&#xc785;&#xb2c8;&#xb2e &#xc77c;&#xbc18;&#xc801;&#xc73c;&#xb85c; libxml2 &#xc744; &#xc124;&#xce58;&#xd558;&#xba74; &#xcc3e;&#xc744; &#xc218; &#xc788;&#xc9c0;&#xb9cc; &#xc124;&#xce58;&#xd558;&#xc9c0; &#xc54a;&#xc558;&#xac70;&#xb098; &#xd2b9;&#xc815; &#xbc84;&#xc804;&#xc744; &#xc0ac;&#xc6a9;&#xd558;&#xae30; &#xbc14;&#xb784; &#xacbd;&#xc6 LibXML2 &#xc124;&#xce58; &#xb514;&#xb809;&#xd130;&#xb9ac;&#xc758; &#xc704;&#xce58;&#xb97c; &#xd655;&#xc77 &#xc18c;&#xd504;&#xd2b8;&#xc6e8;&#xc5b4; &#xc124;&#xce58;&#xb97c; &#xd65c;&#xc131;&#xd654;&#xd558;&#xae3 &#xc704;&#xd574; xml2-config &#xb77c;&#xb294; &#xc124;&#xc815; &#xd30c;&#xc77c;&#xc5d0; PostGIS &#xc758; &#xc704;&#xce58;&#xb97c; &#xc9c0;&#xc815;&#xd574;&#xc57c; &#xd569;&#xb2c8;&#xb2e4;. PostGIS &#xbe4c;&#xb4dc &#xc2dc; &#xb300;&#xc0c1;&#xc774; &#xb420; &#xd2b9;&#xc815; LibXML2 &#xc124;&#xce58; &#xb514;&#xb809;&#xd1 &#xc0ac;&#xc6a9;&#xc790;&#xac00; &#xc9c1;&#xc811; &#xc124;&#xc815;&#xd558;&#xb824;&#xba74; &#xc774; &#xd30c;&#xb77c;&#xbbf8;&#xd130;(**--with-xml2config=/path/to/xml2-config**)&#xb97c; &#xc0ac;&#xc6a9;&#xd558;&#xc704;&#xd574; &#xd544;&#xc694;&#xd55c; &#xb77c;&#xc774;&#xbe0c;&#xb7ec;&#xb9ac;&#xc785;&#xb2c8;&#xb2e4;. PostGIS &#xbe4c;&#xb4dc &#xc2dc; &#xb300;&#xc0c1;&#xc774; &#xb420; &#xd2b9;&#xc815; Proj4 &#xc124;&#xce58; &#xb514;&#xb809;&#xd130;&#xb9ac;&#xb97c; &#xc0ac;&#xc6a9;&#xc790;&#xac00; &#xc9c1;&#xc811; &#xc124;&#xc815;&#xd558;&#xb824;&#xba74; &#xc774; &#xd30c;&#xb77c;&#xbbf8;&#xd130;(**--with-projdir=/path/to/projdir**)&#xb97c; &#xc0ac;&#xc6a9;&#xd558;&#xc704;&#xd574; &#xd544;&#xc218; &#xc7ac;&#xd22c;&#xc601; &#xb77c;&#xc774;&#xbe0c;&#xb7ec;&#xb9ac;&#xc785;&#xb2c8;&#xb2e4;. PostGIS &#xbe4c;&#xb4dc; &#xc2dc; &#xb300;&#xc0c1;&#xc774; &#xb420; &#xd2b9;&#xc815; Proj4 &#xc124;&#xce58; &#xb514;&#xb809;&#xd130;&#xb9ac;&#xb97c; &#xc0ac;&#xc6a9;&#xc790;&#xac00; &#xc9c1;&#xc811; &#xc124;&#xc815;&#xd558;&#xb824;&#xba74; &#xc774; &#xd30c;&#xb77c;&#xbbf8;&#xd130;(**--with-projdir=/path/to/projdir**)&#xb97c; &#xc0ac;&#xc6a9;&#xd558;&#xc704;&#xd574; &#xd544;&#xc218; &#xc7ac;&#xd22c;&#xc601; &#xb77c;&#xc774;&#xbe0c;&#xb7ec;&#xb9ac;&#xc785;&#xb2c8;&#xb2e4;. PostGIS &#xbe4c;&#xb4dc; &#xc2dc; &#xb300;&#xc0c1;&#xc774; &#xb420; &#xd2b9;&#xc815; JSON-C &#xc124;&#xce58; &#xb514;&#xb809;&#xd130;&#xb9ac;&#xb97c; &#xc0ac;&#xc6a9;&#xc790;&#xac00; &#xc9c1;&#xc811; &#xc124;&#xc815;&#xd558;&#xb824;&#xba74; &#xc774; &#xd30c;&#xb77c;&#xbbf8;&#xd130;(**--with-jsondir=/path/to/jsondir**)&#xb97c; &#xc0ac;&#xc6a9;&#xd558;&#xc704;&#xd574; &#xd544;&#xc218; &#xc7ac;&#xd22c;&#xc601; &#xb77c;&#xc774;&#xbe0c;&#xb7ec;&#xb9ac;&#xc785;&#xb2c8;&#xb2e4;. PostGIS &#xbe4c;&#xb4dc; &#xc2dc; &#xb300;&#xc0c1;&#xc774; &#xb420; &#xd2b9;&#xc815; JSON-C &#xc124;&#xce58; &#xb514;&#xb809;&#xd130;&#xb9ac;&#xb97c; &#xc0ac;&#xc6a9;&#xc790;&#xac00; &#xc9c1;&#xc811; &#xc124;&#xc815;&#xd558;&#xb824;&#xba74; &#xc774; &#xd30c;&#xb77c;&#xbbf8;&#xd130;(**--with-jsondir=/path/to/jsondir**)&#xb97c; &#xc0ac;&#xc6a9;&#xd558;&#xc704;&#xd574; &#xd544;&#xc218; &#xc7ac;&#xd22c;&#xc601; &#xb77c;&#xc774;&#xbe0c;&#xb7ec;&#xb9ac;&#xc785;&#xb2c8;&#xb2e4;. PostGIS &#xbe4c;&#xb4dc; &#xc2dc; &#xb300;&#xc0c1;&#xc774; &#xb420; &#xd2b9;&#xc815; PCRE &#xc124;&#xce58; &#xb514;&#xb809;&#xd130;&#xb9ac;&#xb97c; &#xc0ac;&#xc6a9;&#xc790;&#xac00; &#xc9c1;&#xc811; &#xc124;&#xc815;&#xd558;&#xb824;&#xba74; &#xc774; &#xd30c;&#xb77c;&#xbbf8;&#xd130;(**--with-pcredir=/path/to/pcredir**)&#xb97c; &#xc0ac;&#xc6a9;&#xd558;&#xc704;&#xd574; &#xd544;&#xc218; &#xc7ac;&#xd22c;&#xc601; &#xb77c;&#xc774;&#xbe0c;&#xb7ec;&#xb9ac;&#xc785;&#xb2c8;&#xb2e4;. PostGIS &#xbe4c;&#xb4dc; &#xc2dc; &#xb300;&#xc0c1;&#xc774; &#xb420; &#xd2b9;&#xc815; GUI &#xc774;&#xd30c;&#xc77c;(GTK+2.0 &#xd544;&#xc6 shp2pgsql-gui &#xc758; shp2pgsql &#xc5d0; &#xb300;&#xd55c; &#xadf8;&#xb798;&#xd53d; &#xc778;&#xd14c;&#xd398;&#xc7 &#xc0dd;&#xc131;&#xd569;&#xb2c8;&#xb2e4;. &#xb798;&#xc2a4;&#xd130; &#xc9c0;&#xc6d0; &#xc124;&#xce58; Disable topology support. There is no corresponding library as all logic needed for topology is in postgis-3.3.6dev library. &#xae30;&#xbcf8;&#xc801;&#xc73c;&#xb85c; PostGIS &#xb294; gettext &#xc9c0;&#xc6d0;&#xc744; &#xac10 &#xd568;&#xaed8; &#xcef4;&#xd30c;&#xc77c;&#xd558;&#xc9c0;&#xb9cc; &#xb85c;&#xb354;&#xc758; &#xd30c;&#xc19 &#xc57c;&#xae30;&#xd558;&#xb294; &#xbe44;&#xd638;&#xd658;&#xc131; &#xbb38;&#xc81c;&#xac00; &#xbc1c;&#xc0c &#xacbd;&#xc6b0; &#xc774; &#xba85;&#xb839;&#xc5b4;&#xb85c; gettext &#xc9c0;&#xc6d0;&#xc744; &#xc644;&#xc804; &#xbe44;&#xd65c;&#xc131;&#xd654;&#xc2dc;&#xd0ac; &#xc218; &#xc788;&#xc2b5;&#xb2c8;&#xb2e4;. &#xc774;&#xb7 &#xbc29;&#xbc95;&#xc73c;&#xb85c; &#xc124;&#xc815;&#xc744; &#xbcc0;&#xacbd;&#xd574;&#xc11c; &#xbb38;&#xc81 &#xd574;&#xcacb0;&#xd558;&#xb294; &#xc608;&#xb294; &#xbc84;&#xadf8; &#xd2f0;&#xc13; <http://trac.osgeo.org/postgis/ticket/748> &#xc744; &#xc38;&#xc870;&#xd558;&#xc2ed;&#xc2dc;&#xc624;. &#xc8fc;&#xc758;. gettext &#xc9c0;&#xb048;&#xb2e4;&#xace0; &#xd574;&#xc11c; &#xbcc4;&#xb2e4;&#xb978; &#xbb38;&#xc81c;&#xb294; &#xc5c6;&#xc2 gettext &#xc9c0;&#xc6d0;&#xc740; &#xc544;&#xc9c1; &#xbb38;&#xc11c;&#xd654;&#xb418;&#xc9c0;&#xb3c4; &#xc54a;





```
share/extension &#xd3f4;&#xb354;&#xc758;&#xb8e8;&#xd2b8;&#xc5d0;&#xbcf5;&#xc0ac;&#xb418;&#xc5b4;&#xc57c;
&#xd568;&#xc5d0;&#xc8fc;&#xc758;&#xd558;&#xc2ed;&#xc2dc;&#xc624;. extensions/postgis/sql/*.sql,
extensions/postgis_topology/sql/*.sql
```

```
&#xc774;&#xb807;&#xac8c;&#xd55c;&#xb2e4;&#xc74c;,&#xc0ac;&#xc6a9;&#xc790;&#xb294;&#xc0ac;&#xc6a9;&#xac00;&#xc0ac;
extensions&#xc73c;&#xb85c;postgis,postgis_topology&#xc744;PgAdmin->extensions&#xc5d0;&#xc11c;&#xbcf5;
&#xc218;&#xc788;&#xc744;&#xac83;&#xc785;&#xb2c8;&#xb2e4;.
```

```
&#xb9cc;&#xc57d;psql&#xc744;&#xc774;&#xc6a9;&#xc911;&#xc774;&#xb77c;&#xba74;&#xb2e4;&#xc74c;&#xc758;
&#xcffc;&#xb9ac;&#xb97c;&#xc2e4;&#xd589;&#xd568;&#xc73c;&#xb85c;&#xc368;&#xd655;&#xc778;&#xd560;&#xc218;
&#xc788;&#xc2b5;&#xb2c8;&#xb2e4;.
```

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

| name                         | default_version | installed_version |
|------------------------------|-----------------|-------------------|
| address_standardizer         | 3.3.6dev        | 3.3.6dev          |
| address_standardizer_data_us | 3.3.6dev        | 3.3.6dev          |
| postgis                      | 3.3.6dev        | 3.3.6dev          |
| postgis_sfcgal               | 3.3.6dev        |                   |
| postgis_tiger_geocoder       | 3.3.6dev        | 3.3.6dev          |
| postgis_topology             | 3.3.6dev        |                   |

(6 rows)

```
&#xb9cc;&#xc57d;&#xc0ac;&#xc6a9;&#xc790;&#xac00;&#xcffc;&#xb9ac;&#xd558;&#xb294;&#xb370;&#xc774;&#xd130;&#xc0ac;
extension&#xc774;&#xc124;&#xc5e8;&#xb418;&#xc5b4;&#xc788;&#xb2e4;&#xba74;,&#xc0ac;&#xc6a9;&#xc790;&#xb294;
installed_version&#xc5ec;&#xb7fc;&#xc5d0;&#xc11c;&#xc774;&#xb984;&#xc744;&#xbcf5;&#xc218;&#xc788;&#xc2b5;
&#xb9cc;&#xc57d;&#xc544;&#xbb34;&#xb808;&#xcf54;&#xb4dc;&#xb3c4;&#xc5c6;&#xb2e4;&#xba74;&#xc11c;&#xbc84;&#xc0ac;
postgis extension&#xc774;&#xc804;&#xd600;&#xc124;&#xc5e8;&#xb418;&#xc5b4;&#xc788;&#xc9c0;&#xc54a;&#xc74c;&#xc0ac;
&#xb73b;&#xd569;&#xb2c8;&#xb2e4;. PgAdmin III 1.14&#xc774;&#xc0c1;&#xbc84;&#xc804;&#xc5d0;&#xc11c;&#xb294;
&#xb370;&#xc774;&#xd130;&#xbca0;&#xc774;&#xc2a4;&#xd0d0;&#xc0c9;&#xd2b8;&#xb9ac;&#xc758;extensions&#xc5d0;
&#xb9c8;&#xc6b0;&#xc2a4;&#xc624;&#xb978;&#xcabd;&#xbc84;&#xd2bc;&#xd074;&#xb9ad;&#xc744;&#xd1b5;&#xd574;
&#xc5c5;&#xadf8;&#xb808;&#xc774;&#xb4dc;&#xb610;&#xb294;&#xc0ad;&#xc81c;&#xb97c;&#xd5c8;&#xc6a9;&#xd569;&#xc0ac;
extension&#xc774;&#xc774;&#xc6a9;&#xac00;&#xb2a5;&#xd55c;&#xc0c1;&#xd0dc;&#xb77c;&#xba74;pgAdmin exten-
sion&#xc778;&#xd130;&#xd398;&#xc774;&#xc2a4;&#xb610;&#xb294;&#xb2e4;&#xc74c;&#xc758;sql&#xba85;&#xb839;&#xc0ac;
&#xc2e4;&#xd589;&#xd568;&#xc73c;&#xb85c;&#xc368;&#xc120;&#xd0dd;&#xd55c;&#xb370;&#xc774;&#xd130;&#xbca0;&#xc0ac;
&#xc548;&#xc5d0;postgis extension&#xc744;&#xc124;&#xc5e8;&#xd560;&#xc218;&#xc788;&#xc2b5;&#xb2c8;&#xb2e4;.
```

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

```
PSQL&#xc5d0;&#xc11c;&#xb2e4;&#xc74c;&#xba85;&#xb839;&#xc5b4;&#xb97c;&#xc0ac;&#xc6a9;&#xd558;&#xba74;
&#xc5b4;&#xb5a4;&#xbc84;&#xc804;&#xc744;&#xc124;&#xc5e8;&#xd588;&#xb294;&#xc9c0;,&#xc5b4;&#xb5a4;&#xc2a4;&#xc0ac;
&#xc124;&#xc5e8;&#xd588;&#xb294;&#xc9c0;&#xc54c;&#xc218;&#xc788;&#xc2b5;&#xb2c8;&#xb2e4;.
```

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

```
List of installed extensions
```

```
-[ RECORD 1 ]-----
-
Name          | postgis
```

```

Version      | 3.3.6dev
Schema       | public
Description  | PostGIS geometry, geography, and raster spat..
-[ RECORD 2 ]-----
-
Name         | postgis_tiger_geocoder
Version      | 3.3.6dev
Schema       | tiger
Description  | PostGIS tiger geocoder and reverse geocoder
-[ RECORD 3 ]-----
-
Name         | postgis_topology
Version      | 3.3.6dev
Schema       | topology
Description  | PostGIS topology spatial types and functions

```

**Warning**


 spatial\_ref\_sys, layer, topology &#xd655;&#xc7a5; &#xd14c;&#xc774;&#xbe14;&#xc740;
 &#xbc31;&#xc5c5;&#xb418;&#xc9c0; &#xc54a;&#xc2b5;&#xb2c8;&#xb2e4;. &#xc774;&#xac83;&#xb4e4;&#xc740;
 postgis &#xb610;&#xb294; postgis\_topology extension&#xc774; &#xbc31;&#xc5c5;&#xc774; &#xb420;
 &#xacbd;&#xc6b0;&#xc5d0;&#xb9cc; &#xbc31;&#xc5c5;&#xc774; &#xac00;&#xb2a5;&#xd569;&#xb2c8;&#xb2e4;.
 &#xc774;&#xb294; &#xc804;&#xc0cb4; &#xb370;&#xc774;&#xd130;&#xbca0;&#xc774;&#xc2a4;&#xac00;
 &#xbc31;&#xc5c5;&#xb420; &#xb54c;&#xc5d0;&#xb9cc; &#xbc1c;&#xc0dd;&#xd55c;&#xb2e4;&#xace0;
 &#xbcf; &#xc218; &#xc788;&#xc2b5;&#xb2c8;&#xb2e4;. PostGIS 2.0.1&#xc5d0;&#xc11c;&#xb294;
 &#xb370;&#xc774;&#xd130;&#xbca0;&#xc774;&#xc2a4; &#xbc31;&#xc5c5;&#xc2dc; srid
 &#xb808;&#xc5f4;&#xb4dc;&#xb9cc;&#xc774; &#xbc31;&#xc5c5;&#xb429;&#xb2c8;&#xb2e4;. &#xc774;&#xc640;
 &#xad00;&#xd55c; &#xbb38;&#xc81c;&#xb97c; &#xbc1c;&#xacac;&#xd558;&#xba74; trac &#xd2f0;&#xcf13;&#xc744;
 &#xbc1c;&#xd589;&#xd574;&#xc8fc;&#xc2ed;&#xc2dc;&#xc624;. extension &#xd14c;&#xc774;&#xbe14;&#xc758;
 &#xad6c;&#xc870;&#xb4e4;&#xc740; CREATE EXTENSION&#xacfc; &#xd568;&#xae8;
 &#xc0dd;&#xc131;&#xb418;&#xae30; &#xb54c;&#xbb38;&#xc5d0; &#xbc31;&#xc5c5;&#xb418;&#xc9c0;
 &#xc54a;&#xc2b5;&#xb2c8;&#xb2e4;. &#xc774;&#xb7ec;&#xd55c; &#bc29;&#xc2dd;&#xc740; PostgreSQL
 extension &#xbaa8;&#xb378;&#xc5d0; &#xc801;&#xc6a9;&#xb418;&#xae30; &#xb54c;&#xbb38;&#xc5d0;
 &#xc870;&#xc5e8;&#xb97c; &#xcde8;&#xd560; &#xc218; &#xc788;&#xb294; &#bc29;&#bc95;&#xc774;
 &#xc5c6;&#xc2b5;&#xb2c8;&#xb2e4;.

&#xc6b0;&#xb9ac;&#xc758; &#xba4b;&#xc9c4; &#xd655;&#xc7a5; &#xd504;&#xb85c;&#xadf8;&#xb7a8; &#xc2dc;&#xc2a4;&#xc5c6;&#xc774;
 3.3.6dev&#xc744; &#xc124;&#xc5e8;&#xd588;&#xb2e4;&#xba74; &#xba3c;&#xc800; &#xb2e4;&#xc74c; &#xc5c5;&#xadf8;
 &#xb808;&#xc774;&#xb4dc; &#xc2a4;&#xd06c;&#xb9bd;&#xd2b8;&#xb97c; &#xc2e4;&#xd589;&#xd574;&#xd655;&#xc7a5;
 &#xd504;&#xb85c;&#xadf8;&#xb7a8; &#xae30;&#bc18; &#xcd5c;&#xc2e0; &#bc84;&#xc804;&#xc73c;&#xbcc0;&#xacbd;
 &#xd560; &#xc218; &#xc788;&#xc2b5;&#xb2c8;&#xb2e4;. postgis\_upgrade\_22\_minor.sql,raster\_upgrade\_22\_minor.sql,topology\_upgrade\_22\_minor.sql.

```

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

```

**2.2.6 &#xd14c;&#xc2a4;&#xd2b8;**

&#xb9cc;&#xc57d; PostGIS &#xbe4c;&#xb4dc;&#xb97c; &#xd14c;&#xc2a4;&#xd2b8;&#xd558;&#xace0; &#xc2f6;&#xb2e4;&#xc2e4;
 &#xd589;&#xd558;&#xc2ed;&#xc2dc;&#xc624;.

**make check**

&#xc704; &#xba85;&#xb839;&#xc5b4;&#xb294; &#xd65c;&#xc131; PostgreSQL &#xb370;&#xc774;&#xd130;&#xbca0;&#xc774;
 &#bc14;&#xd0d5;&#xc73c;&#xb85c; &#xc0dd;&#xc131;&#xb41c; &#xb77c;&#xc774;&#xbe0c;&#xb7ec;&#xb9ac;&#xb97c;
 &#xc774;&#xc6a9;&#xd558;&#xc5ec; &#xb2e4;&#xc591;&#xd55c; &#xd655;&#xc778;&#xacfc; &#xd68c;&#xad0; &#xd14c;&#xc2e4;
 &#xd589;&#xd560; &#xac83;&#xc785;&#xb2c8;&#xb2e4;.





```

PostGIS &#xc758;: &#xc774;&#xbbf8; PostGIS &#xcf54;&#xb4dc; &#xd3f4;&#xb354;&#xc758; &#xb8e8;&#xd2b8;&#xc5d0;&#xc
make install&#xc744; &#xc2e4;&#xd589;&#xd588;&#xb2e4;&#xba74; &#xb2e4;&#xc2dc; &#xc2e4;&#xd589;&#xd560; &#xd544;&#xc
&#xc5c6;&#xc2b5;&#xb2c8;&#xb2e4;.

```

```
address_standardizer&#xc758; &#xacbd;&#xc6b0;:
```

```

cd extensions/address_standardizer
make install
make installcheck

```

```
&#xb2e4;&#xc74c;&#xacfc; &#xac19;&#xc740; &#xacb0;&#xacfc;&#xac00; &#xb098;&#xc640;&#xc57c; &#xd569;&#xb2c8;&#xc7
```

```

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

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

```

```

TIGER &#xc9c0;&#xc624;&#xcf54;&#xb529; &#xb3c4;&#xad6c;&#xc758; &#xacbd;&#xc6b0;, &#xc0ac;&#xc6a9;&#xc790;&#xc7
PostgreSQL &#xc778;&#xc2a4;&#xd134;&#xc2a4; &#xc548;&#xc5d0;&#xc11c; PostGIS &#xbc0f; fuzzystmatch &#xd655;&#xc7a
&#xd504;&#xb85c;&#xadf8;&#xb7a8;&#xc744; &#xc774;&#xc6a9;&#xd560; &#xc218; &#xc788;&#xb294;&#xc9c0; &#xd655;&#xc
address_standardizer &#xc9c0;&#xc6d0;&#xc774; &#xb418;&#xb3c4;&#xb85d; PostGIS&#xb97c; &#xbe4c;&#xb4dc;&#xd588;&#xc
address_standardizer &#xd14c;&#xc2a4;&#xd2b8;&#xb3c4; &#xd568;&#xaed8; &#xc2e4;&#xd589;&#xb420; &#xac83;&#xc785;&#xc

```

```

cd extensions/postgis_tiger_geocoder
make install
make installcheck

```

```
&#xb2e4;&#xc74c;&#xacfc; &#xac19;&#xc740; &#xacb0;&#xacfc;&#xac00; &#xb098;&#xc640;&#xc57c; &#xd569;&#xb2c8;&#xc7
```

```

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

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

```

## 2.2.7

PostGIS

```
make install
```

```
--prefix /bin/
PostGIS
--prefix /lib/
```

- ```
(loader)
/bin/
```
- ```
postgis.sql
--prefix /share/contrib/
```
- ```
PostGIS
--prefix /lib/
```

```
postgis_comments.sql, raster_comments.sql
--prefix /share/contrib/
make comments
--prefix /lib/
```

### make comments-install



#### Note

```
xsiltproc
--prefix /lib/
postgis_comments.sql, raster_comments.sql,
topology_comments.sql
```

## 2.3

address\_standardizer

```
PostGIS 2.2
--prefix /lib/
Section 14.1
```

Normalize\_Address

```
PostGIS
TIGER
--prefix /lib/
Section 2.4.3
--prefix /lib/
Section 2.2.3
--prefix /lib/
```

```
PCRE
--prefix /lib/
http://www.pcre.org
--prefix /lib/
Section 2.2.3
--prefix /lib/
```







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

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

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

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

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

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

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

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

Alternatively you can load just these tables after loading state data using the [Loader\\_Generate\\_Census\\_Script](#)

12. For each state you want to load data for, generate a state script [Loader\\_Generate\\_Script](#).



#### Warning

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

- 13.
- ```
psql -c "SELECT Loader_Generate_Script(ARRAY['MA'], 'debbie')" -d geocoder -tA > /
gisdata/ma_load.sh
```

- 14.

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

- 15.

```
SELECT install_missing_indexes();
vacuum (analyze, verbose) tiger.addr;
vacuum (analyze, verbose) tiger.edges;
vacuum (analyze, verbose) tiger.faces;
vacuum (analyze, verbose) tiger.featnames;
vacuum (analyze, verbose) tiger.place;
vacuum (analyze, verbose) tiger.cousub;
vacuum (analyze, verbose) tiger.county;
```





```
tiger
Drop_State_Tables_Generate_Script
Drop_Nation_Tables_Generate_Script
Loader_Generate_Nation_Script
```

```
PostGIS
shp2pgsql
wget
http://www.7-zip.org/
http://gnuwin32.sourceforge.net/packages/wget.htm
```

- Unix
- PostGIS
- wget

```
tiger_2010
Drop_Nation_Tables_Generate_Script
Loader_Generate_Nation_Script
(2010)
install_missing_indexes
```

```
SELECT install_missing_indexes();
```

```
Geocode
```

## 2.4.5 Tiger Geocoder

```
PostGIS 2.0
TIGER
tar
http://www.7-zip.org/
http://gnuwin32.sourceforge.net/packages/wget.htm
```



## 2.5 检查数据库版本

检查数据库版本，可以通过以下命令实现：

1. PostgreSQL 11 版本检查
 

在 Linux 系统上，可以通过以下命令检查 PostgreSQL 11 的版本：

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

或者，可以通过以下 SQL 查询检查 PostgreSQL 11 的版本：

```
SELECT version();
```

```
SELECT version();
```

RPM 包管理器可以通过以下命令检查 PostgreSQL 11 的版本：

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

2. PostgreSQL 11 版本检查
 

在 Linux 系统上，可以通过以下命令检查 PostgreSQL 11 的版本：

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

```
SELECT postgis_full_version();
```

检查数据库版本，可以通过以下命令实现：

1. PostgreSQL 11 版本检查
 

在 Linux 系统上，可以通过以下命令检查 PostgreSQL 11 的版本：

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

## Chapter 3

# PostGIS Administration

### 3.1 Performance Tuning

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

For general details about optimizing PostgreSQL, refer to [Tuning your PostgreSQL Server](#).

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

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

In addition to the Postgres settings, PostGIS has some custom settings which are listed in [Section 8.23](#).

#### 3.1.1 Startup

These settings are configured in `postgresql.conf`:

##### `constraint_exclusion`

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

##### `shared_buffers`

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

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

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

### 3.1.2 Runtime

**work\_mem** - sets the size of memory used for sort operations and complex queries

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

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

**maintenance\_work\_mem** - the memory size used for VACUUM, CREATE INDEX, etc.

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

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

#### **max\_parallel\_workers\_per\_gather**

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

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

## 3.2 Configuring raster support

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

As of PostGIS 2.1.3, out-of-db rasters and all raster drivers are disabled by default. In order to re-enable these, you need to set the following environment variables `POSTGIS_GDAL_ENABLED_DRIVERS` and `POSTGIS_ENABLE_OUTDB_RASTERS` in the server environment. For PostGIS 2.2, you can use the more cross-platform approach of setting the corresponding Section 8.23.

If you want to enable offline raster:

```
POSTGIS_ENABLE_OUTDB_RASTERS=1
```

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

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

```
POSTGIS_GDAL_ENABLED_DRIVERS=ENABLE_ALL
```

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

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

**Note**

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

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

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

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

## 3.3 Spatially enable database using EXTENSION

### 3.3.1 Spatially enable database using EXTENSION

If you are using PostgreSQL 9.1+ and have compiled and installed the extensions/postgis modules, you can turn a database into a spatial one using the EXTENSION mechanism.

Core postgis extension includes geometry, geography, spatial\_ref\_sys and all the functions and comments. Raster and topology are packaged as a separate extension.

Run the following SQL snippet in the database you want to enable spatially:

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

### 3.3.2 Spatially enable database without using EXTENSION (discouraged)

**Note**

This is generally only needed if you cannot or don't want to get PostGIS installed in the PostgreSQL extension directory (for example during testing, development or in a restricted environment).

Adding PostGIS objects and function definitions into your database is done by loading the various sql files located in `[prefix]/share/contrib` as specified during the build phase.

The core PostGIS objects (geometry and geography types, and their support functions) are in the `postgis.sql` script. Raster objects are in the `rtpostgis.sql` script. Topology objects are in the `topology.sql` script.

For a complete set of EPSG coordinate system definition identifiers, you can also load the `spatial_ref_sys.sql` definitions file and populate the `spatial_ref_sys` table. This will permit you to perform `ST_Transform()` operations on geometries.

If you wish to add comments to the PostGIS functions, you can find them in the `postgis_comments.sql` script. Comments can be viewed by simply typing `\dd [function_name]` from a `psql` terminal window.

Run the following Shell commands in your terminal:

```
DB=[yourdatabase]
SCRIPTSDIR=`pg_config --sharedir`/contrib/postgis-3.2/

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

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

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

### 3.3.3 Create a spatially-enabled database from a template

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

From the shell:

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

From SQL:

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

## 3.4 Upgrading spatial databases

Upgrading existing spatial databases can be tricky as it requires replacement or introduction of new PostGIS object definitions. Unfortunately not all definitions can be easily replaced in a live database, so sometimes your best bet is a dump/reload process. PostGIS provides a SOFT UPGRADE procedure for minor or bugfix releases, and a HARD UPGRADE procedure for major releases.

Before attempting to upgrade PostGIS, it is always worth to backup your data. If you use the `-Fc` flag to `pg_dump` you will always be able to restore the dump with a HARD UPGRADE.

### 3.4.1 Soft upgrade

If you installed your database using extensions, you'll need to upgrade using the extension model as well. If you installed using the old sql script way, you are advised to switch your install to extensions because the script way is no longer supported.

#### 3.4.1.1 Soft Upgrade 9.1+ using extensions

If you originally installed PostGIS with extensions, then you need to upgrade using extensions as well. Doing a minor upgrade with extensions, is fairly painless.

If you are running PostGIS 3 or above, then you should use the [PostGIS\\_Extensions\\_Upgrade](#) function to upgrade to the latest version you have installed.

```
SELECT postgis_extensions_upgrade();
```

If you are running PostGIS 2.5 or lower, then do the following:

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

If you have multiple versions of PostGIS installed, and you don't want to upgrade to the latest, you can explicitly specify the version as follows:

```
ALTER EXTENSION postgis UPDATE TO "3.3.6dev";
ALTER EXTENSION postgis_topology UPDATE TO "3.3.6dev";
```

If you get an error notice something like:

```
No migration path defined for ... to 3.3.6dev
```

Then you'll need to backup your database, create a fresh one as described in Section 3.3.1 and then restore your backup on top of this new database.

If you get a notice message like:

```
Version "3.3.6dev" of extension "postgis" is already installed
```

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

```
ALTER EXTENSION postgis UPDATE TO "3.3.6devnext";
ALTER EXTENSION postgis_topology UPDATE TO "3.3.6devnext";
```



#### Note

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



#### Note

If you are upgrading PostGIS extension from a version prior to 3.0.0, you will have a new extension `postgis_raster` which you can safely drop, if you don't need raster support. You can drop as follows:

```
DROP EXTENSION postgis_raster;
```

### 3.4.1.2 Soft Upgrade Pre 9.1+ or without extensions

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

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

NOTE: if you are moving from PostGIS 1.\* to PostGIS 2.\* or from PostGIS 2.\* prior to r7409, you cannot use this procedure but would rather need to do a **HARD UPGRADE**.

After compiling and installing (make install) you should find a set of `*_upgrade.sql` files in the installation folders. You can list them all with:

```
ls `pg_config --sharedir`/contrib/postgis-3.3.6dev/*_upgrade.sql
```

Load them all in turn, starting from `postgis_upgrade.sql`.

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

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

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

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

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

You are advised to switch to an extension based install by running

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



#### Note

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

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

## 3.4.2 Hard upgrade

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

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

Supplementary instructions for windows users are available at [Windows Hard upgrade](#).

The Procedure is as follows:

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

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

2. Do a fresh install of PostGIS in a new database -- we'll refer to this database as `newdb`. Please refer to [Section 3.3.2](#) and [Section 3.3.1](#) for instructions on how to do this.

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

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

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

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

Errors may arise in the following cases:

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

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

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

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

If you are upgrading an old database containing french **IGN** cartography, you will have probably SRIDs out of range and you will see, when importing your database, issues like this :

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

In this case, you can try following steps : first throw out completely the IGN from the sql which is resulting from `postgis_restore.pl`. So, after having run :

```
perl utils/postgis_restore.pl "/somepath/olddb.backup" > olddb.sql
```

run this command :

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

Create then your newdb, activate the required Postgis extensions, and insert properly the french system IGN with : [this script](#) After these operations, import your data :

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

## Chapter 4

# Data Management

### 4.1 GIS

#### 4.1.1 OGC Geometry

The Open Geospatial Consortium (OGC) developed the *Simple Features Access* standard (SFA) to provide a model for geospatial data. It defines the fundamental spatial type of **Geometry**, along with operations which manipulate and transform geometry values to perform spatial analysis tasks. PostGIS implements the OGC Geometry model as the PostgreSQL data types **geometry** and **geography**.

Geometry is an *abstract* type. Geometry values belong to one of its *concrete* subtypes which represent various kinds and dimensions of geometric shapes. These include the **atomic** types **Point**, **LineString**, **LinearRing** and **Polygon**, and the **collection** types **MultiPoint**, **MultiLineString**, **MultiPolygon** and **GeometryCollection**. The *Simple Features Access - Part 1: Common architecture v1.2.1* adds subtypes for the structures **PolyhedralSurface**, **Triangle** and **TIN**.

Geometry models shapes in the 2-dimensional Cartesian plane. The **PolyhedralSurface**, **Triangle**, and **TIN** types can also represent shapes in 3-dimensional space. The size and location of shapes are specified by their **coordinates**. Each coordinate has a **X** and **Y ordinate** value determining its location in the plane. Shapes are constructed from points or line segments, with points specified by a single coordinate, and line segments by two coordinates.

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

Geometry values are associated with a **spatial reference system** indicating the coordinate system in which it is embedded. The spatial reference system is identified by the geometry **SRID** number. The units of the **X** and **Y** axes are determined by the spatial reference system. In **planar** reference systems the **X** and **Y** coordinates typically represent easting and northing, while in **geodetic** systems they represent longitude and latitude. **SRID 0** represents an infinite Cartesian plane with no units assigned to its axes. See Section 4.5.

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

A geometry value may be **empty**. Empty values contain no vertices (for atomic geometry types) or no elements (for collections).

An important property of geometry values is their spatial **extent** or **bounding box**, which the OGC model calls **envelope**. This is the 2 or 3-dimensional box which encloses the coordinates of a geometry. It is an efficient way to represent a geometry's extent in coordinate space and to check whether two geometries interact.

The geometry model allows evaluating topological spatial relationships as described in Section 5.1.1. To support this the concepts of **interior**, **boundary** and **exterior** are defined for each geometry type. Geometries are topologically closed, so they always contain their boundary. The boundary is a geometry of dimension one less than that of the geometry itself.

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

#### 4.1.1.1 Point

A Point is a 0-dimensional geometry that represents a single location in coordinate space.

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

#### 4.1.1.2 LineString

A LineString is a 1-dimensional line formed by a contiguous sequence of line segments. Each line segment is defined by two points, with the end point of one segment forming the start point of the next segment. An OGC-valid LineString has either zero or two or more points, but PostGIS also allows single-point LineStrings. LineStrings may cross themselves (self-intersect). A LineString is **closed** if the start and end points are the same. A LineString is **simple** if it does not self-intersect.

```
LINESTRING(0 0,1 1,1 2)
```

#### 4.1.1.3 LinearRing

A LinearRing is a LineString which is both closed and simple. The first and last points must be equal, and the line must not self-intersect.

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

#### 4.1.1.4 Polygon

A Polygon is a 2-dimensional planar region, delimited by an exterior boundary (the shell) and zero or more interior boundaries (holes). Each boundary is a [LinearRing](#).

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

#### 4.1.1.5 MultiPoint

A MultiPoint is a collection of Points.

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

#### 4.1.1.6 MultiLineString

A MultiLineString is a collection of LineStrings. A MultiLineString is closed if each of its elements is closed.

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







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

```
&#xc608;&#xb97c;&#xb4e4;&#xc5b4; OGC &#xacf5;&#xac04; &#xac1d;&#xccb4;&#xb97c; &#xc0dd;&#xc131;&#xd558;&#xace
&#xc0bd;&#xc785;&#xd558;&#xae30; &#xc704;&#xd55c; &#xc720;&#xd6a8;&#xd55c; &#xc0bd;&#xc785; &#xad6c;&#xbb38;&#
&#xb2e4;&#xc74c;&#xacfc; &#xac19;&#xc744; &#xac83;&#xc785;&#xb2c8;&#xb2e4;;
```

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

Well-Known Binary (WKB) provides a portable, full-precision representation of spatial data as binary data (arrays of bytes). Examples of the WKB representations of spatial objects are:

- POINT(0 0)  
WKB: 010100000000000000000000F03F000000000000F03
- LINestring(0 0,1 1,1 2)  
WKB: 0102000000020000000000000000004000000000000040000000000022400000000000002240

Input and output of WKB is provided by the functions `ST_AsBinary` and `ST_GeomFromWKB`:

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

```
&#xc608;&#xb97c;&#xb4e4;&#xc5b4; OGC &#xacf5;&#xac04; &#xac1d;&#xccb4;&#xb97c; &#xc0dd;&#xc131;&#xd558;&#xace
&#xc0bd;&#xc785;&#xd558;&#xae30; &#xc704;&#xd55c; &#xc720;&#xd6a8;&#xd55c; &#xc0bd;&#xc785; &#xad6c;&#xbb38;&#
&#xb2e4;&#xc74c;&#xacfc; &#xac19;&#xc744; &#xac83;&#xc785;&#xb2c8;&#xb2e4;;
```

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

## 4.2 Geometry Data Type

PostGIS implements the OGC Simple Features model by defining a PostgreSQL data type called `geometry`. It represents all of the geometry subtypes by using an internal type code (see `&#xb3c4;&#xd615;` `&#xc720;&#xd615;` and `ST_GeometryType`). This allows modelling spatial features as rows of tables defined with a column of type `geometry`.

The `geometry` data type is *opaque*, which means that all access is done via invoking functions on geometry values. Functions allow creating geometry objects, accessing or updating all internal fields, and compute new geometry values. PostGIS supports all the functions specified in the OGC *Simple feature access - Part 2: SQL option* (SFS) specification, as well many others. See Chapter 8 for the full list of functions.



### Note

PostGIS follows the SFA standard by prefixing spatial functions with "ST\_". This was intended to stand for "Spatial and Temporal", but the temporal part of the standard was never developed. Instead it can be interpreted as "Spatial Type".

```
OpenGIS &#xc0ac;&#xc591;&#xc11c;&#xb294; &#xacf5;&#xac04; &#xac1d;&#xccb4;&#xc758; &#xb0b4;&#xbd80; &#xc800;&#
&#xd615;&#xc2dd;&#xc774; &#xacf5;&#xac04; &#xcc38;&#xc870; &#xc2dc;&#xc2a4;&#xd15c; &#xc2dd;&#xbcc4;&#xc790;(SR
&#xd3ec;&#xd568;&#xd558;&#xb3c4;&#xb85d; &#xc694;&#xad6c;&#xd569;&#xb2c8;&#xb2e4;. &#xb370;&#xc774;&#xd130;&#
&#xc0bd;&#xc785;&#xb420; &#xacf5;&#xac04; &#xac1d;&#xccb4; &#xc0dd;&#xc131;&#xc2dc; SRID&#xac00; &#xd544;&#xc69
```

To make querying geometry efficient PostGIS defines various kinds of spatial indexes, and spatial operators to use them. See Section 4.9 and Section 5.2 for details.



```
&#xd53c;&#xccd0;&#xb4e4;&#xc758; &#xacf5;&#xac04; &#xc624;&#xbe0c;&#xc81d;&#xd2b8;&#xb4e4;&#xc758; &#xd14d;&#x
&#xbb38;&#xc790;&#xc5f4;&#xd45c;&#xd604;&#xb4e4;(WKT) &#xc758; &#xc608;&#xb4e4;&#xb85c;&#xb294; &#xb2e4;&#xc7
&#xac19;&#xc740; &#xac83;&#xb4e4;&#xc774; &#xc788;&#xc2b5;&#xb2c8;&#xb2e4;:
```

- POINT(0 0 0) -- XYZ
- SRID=32632;POINT(0 0) -- SRID &#xcd94;&#xac00; XY
- POINTM(0 0 0) -- XYM
- POINT(0 0 0 0) -- XYZM
- SRID=4326;MULTIPOINTM(0 0 0,1 2 1) -- SRID &#xcd94;&#xac00; XYM
- MULTILINESTRING((0 0 0,1 1 0,1 2 1),(2 3 1,3 2 1,5 4 1))
- POLYGON((0 0 0,4 0 0,4 4 0,0 4 0,0 0 0),(1 1 0,2 1 0,2 2 0,1 2 0,1 1 0))
- MULTIPOLYGON(((0 0 0,4 0 0,4 4 0,0 4 0,0 0 0),(1 1 0,2 1 0,2 2 0,1 2 0,1 1 0)),((-1 -1 0,-1 -2 0,-2 -2 0,-2 -1 0,-1 -1 0)))
- GEOMETRYCOLLECTIONM( POINTM(2 3 9), LINESTRINGM(2 3 4, 3 4 5) )
- MULTICURVE( (0 0, 5 5), CIRCULARSTRING(4 0, 4 4, 8 4) )
- POLYHEDRALSURFACE( ((0 0 0, 0 0 1, 0 1 1, 0 1 0, 0 0 0)), ((0 0 0, 0 1 0, 1 1 0, 1 0 0, 0 0 0)), ((0 0 0, 1 0 0, 1 0 1, 0 0 1, 0 0 0)), ((1 1 0, 1 1 1, 1 0 1, 1 0 0, 1 1 0)), ((0 1 0, 0 1 1, 1 1 1, 1 1 0, 0 1 0)), ((0 0 1, 1 0 1, 1 1 1, 0 1 1, 0 0 1)) )
- TRIANGLE ((0 0, 0 9, 9 0, 0 0))
- TIN( ((0 0 0, 0 0 1, 0 1 0, 0 0 0)), ((0 0 0, 0 1 0, 1 1 0, 0 0 0)) )

```
&#xb2e4;&#xc74c;&#xacfc; &#xac19;&#xc740; &#xc778;&#xd130;&#xd398;&#xc774;&#xc2a4;&#xb97c; &#xc774;&#xc6a9;&#xc7
&#xc774; &#xd615;&#xc2dd;&#xc744; &#xc785;&#xb825;/&#xcd9c;&#xb825;&#xd560; &#xc218; &#xc788;&#xc2b5;&#xb2c8;&
```

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

```
&#xc608;&#xb97c;&#xb4e4;&#xc5b4; PostGIS &#xacf5;&#xac04; &#xac1d;&#xccb4;&#xb97c; &#xc0dd;&#xc131;&#xd558;&#xa
&#xc0bd;&#xc785;&#xd558;&#xae30; &#xc704;&#xd55c; &#xc720;&#xd6a8;&#xd55c; &#xc0bd;&#xc785; &#xad6c;&#xbb38;&#xc7
&#xb2e4;&#xc74c;&#xacfc; &#xac19;&#xc744; &#xac83;&#xc785;&#xb2c8;&#xb2e4;:
```

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

### 4.3 PostGIS &#xc9c0;&#xb9ac;&#xd615; &#xc720;&#xd615;

```
&#xc9c0;&#xb9ac;&#xd615; &#xc720;&#xd615;&#xc740; (&#xc885;&#xc885; "&#xce21;&#xc9c0;" &#xc88c;&#xd45c;,
&#xb610;&#xb294; "&#xc704;&#xb3c4;/&#xacbd;&#xb3c4;"&#xb098; "&#xacbd;&#xb3c4;/&#xc704;&#xb3c4;"&#xb77c;&#xacef;
&#xbd88;&#xb9ac;&#xb294;)"&#xc9c0;&#xb9ac;" &#xc88c;&#xd45c;&#xb85c; &#xd45c;&#xd604;&#xb418;&#xb294; &#xacf5;
&#xd53c;&#xc98;&#xb97c; &#xc790;&#xccb4;&#xc801;&#xc73c;&#xb85c; &#xc9c0;&#xc6d0;&#xd569;&#xb2c8;&#xb2e4;.
&#xc9c0;&#xb9ac; &#xc88c;&#xd45c;&#xb294; &#xac01;&#xb3c4; &#xb2e8;&#xc704;(&#xb3c4;)&#xb97c; &#xc0ac;&#xc6a9;&#xc7
&#xad6c;&#xba74;(&#x7403;&#x9762;) &#xc88c;&#xd45c;&#xc785;&#xb2c8;&#xb2e4;.
```

```
PostGIS &#xb3c4;&#xd615; &#xc720;&#xd615;&#xc740; &#xd3c9;&#xba74;&#xc744; &#xae30;&#xbc18;&#xc73c;&#xb85c;
&#xd569;&#xb2c8;&#xb2e4;. &#xd3c9;&#xba74;&#xc0c1;&#xc5d0;&#xc11c; &#xb450; &#xd3ec;&#xc778;&#xd2b8; &#xc0ac;&#xc7
&#xac00;&#xc7a5; &#xc9e7;&#xc740; &#xacbd;&#xb85c;&#xb294; &#xc9c1;&#xc120;&#xc785;&#xb2c8;&#xb2e4;. &#xc989;
&#xb370;&#xce74;&#xb974;&#xd2b8; &#xc218;&#xd559;&#xacfc; &#xc9c1;&#xc120; &#xbca1;&#xd130;&#xb97c; &#xc774;&#xc7
```

PostGIS geography data type. The geography data type is associated with a spatial reference system via a spatial reference system identifier (SRID). Any geodetic (long/lat based) spatial reference system defined in the `spatial_ref_sys` table can be used. (Prior to PostGIS 2.2, the geography type supported only WGS 84 geodetic (SRID:4326)). You can add your own custom geodetic spatial reference system as described in Section 4.5.2.

For all spatial reference systems the units returned by measurement functions (e.g. `ST_Distance`, `ST_Length`, `ST_Perimeter`, `ST_Area`) and for the distance argument of `ST_DWithin` are in meters.

Like the geometry data type, geography data is associated with a spatial reference system via a spatial reference system identifier (SRID). Any geodetic (long/lat based) spatial reference system defined in the `spatial_ref_sys` table can be used. (Prior to PostGIS 2.2, the geography type supported only WGS 84 geodetic (SRID:4326)). You can add your own custom geodetic spatial reference system as described in Section 4.5.2.

For all spatial reference systems the units returned by measurement functions (e.g. `ST_Distance`, `ST_Length`, `ST_Perimeter`, `ST_Area`) and for the distance argument of `ST_DWithin` are in meters.

For all spatial reference systems the units returned by measurement functions (e.g. `ST_Distance`, `ST_Length`, `ST_Perimeter`, `ST_Area`) and for the distance argument of `ST_DWithin` are in meters.

### 4.3.1 Creating a Geography Column

You can create a table to store geography data using the `CREATE TABLE` SQL statement with a column of type `geography`. The following example creates a table with a geography column storing 2D LineStrings in the WGS84 geodetic coordinate system (SRID 4326):

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

The geography type supports two optional type modifiers:

- the `POINT` modifier restricts the spatial reference system SRID to a particular number. If omitted, the SRID defaults to 4326 (WGS84 geodetic), and all calculations are performed using WGS84.
- the `LINESTRING`, `POLYGON`, `MULTIPOINT`, `MULTILINESTRING`, `MULTIPOLYGON`, `Z`, `M`, `ZM`, `LINESTRINGM`, `POLYGONM`, `MULTIPOINTM`, `MULTILINESTRINGM`, `MULTIPOLYGONM` modifiers restrict the spatial reference system SRID to a particular number. If omitted, the SRID defaults to 4326 (WGS84 geodetic), and all calculations are performed using WGS84.

Examples of creating tables with geography columns:









```

&#xb3c4;&#xd615;&#xc758; &#xacbd;&#xc6b0;, &#xb300;&#xc6a9;&#xb7c9; &#xd3f4;&#xb9ac;&#xace4;&#xc744;
&#xb300;&#xc0c1;&#xc73c;&#xb85c; &#xc881;&#xc740; &#xc9c0;&#xc5ed;&#xc5d0; &#xb300;&#xd55c; &#xcffc;&#xb9a
&#xd560; &#xb54c; &#xc0ac;&#xc6a9;&#xc790; &#xb3c4;&#xd615; &#xb370;&#xc774;&#xd130;&#xb97c; &#xb354;
&#xc791;&#xc740; &#xb369;&#xc5b4;&#xb9ac;&#xb4e4;&#xb85c; "&#xbe44;&#xc815;&#xaddc;&#xd654;"&#xd574;&#xc
&#xc778;&#xb371;&#xc2a4;&#xac00; &#xd6a8;&#xc728;&#xc801;&#xc73c;&#xb85c; &#xac1d;&#xcceb4;&#xc758;
&#xc77c;&#xbd80;&#xbd84;&#xc744; &#xd558;&#xc704; &#xcffc;&#xb9ac;(subquery)&#xd560; &#xc218; &#xc788;&#xb3
&#xb9cc;&#xb4e4;&#xc5b4; &#xcffc;&#xb9ac; &#xc2dc; &#xb9e4;&#xbc88; &#xc804;&#xcceb4; &#xac1d;&#xcceb4;&#xb97
&#xc77d;&#xc5b4;&#xc62c; &#xd544;&#xc694;&#xac00; &#xc5c6;&#xb3c4;&#xb85d; &#xd558;&#xb294; &#xd3b8;&#xc7
&#xc88b;&#xc2b5;&#xb2c8;&#xb2e4;. &#xc720;&#xb7fd; &#xc804;&#xcceb4;&#xb97c; &#xd3f4;&#xb9ac;&#xace4;
&#xd55c; &#xac1c;&#xb85c; &#xc800;&#xc7a5;&#xd560; &#xc218; &#xc788;&#xb2e4;&#xace0; &#xd574;&#xc11c;
&#xaf2d; &#xadf8;&#xb807;&#xac8c; &#xd574;&#xc57c; &#xd55c;&#xb2e4;&#xb294; &#xb73b;&#xc740; &#xc544;&#xb2

```

## 4.4 Geometry Validation

PostGIS is compliant with the Open Geospatial Consortium's (OGC) Simple Features specification. That standard defines the concepts of geometry being *simple* and *valid*. These definitions allow the Simple Features geometry model to represent spatial objects in a consistent and unambiguous way that supports efficient computation. (Note: the OGC SF and SQL/MM have the same definitions for simple and valid.)

### 4.4.1 Simple Geometry

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

```

POINT &#xb780; 0&#xc28;&#xc6d0; &#xb3c4;&#xd615; &#xac1d;&#xcceb4;&#xb85c;&#xc11c; &#xc0c1;&#xc18d;&#xc801;&#xc
&#xb2e8;&#xc21c;&#xd615; &#xc785;&#xb2c8;&#xb2e4;.

```

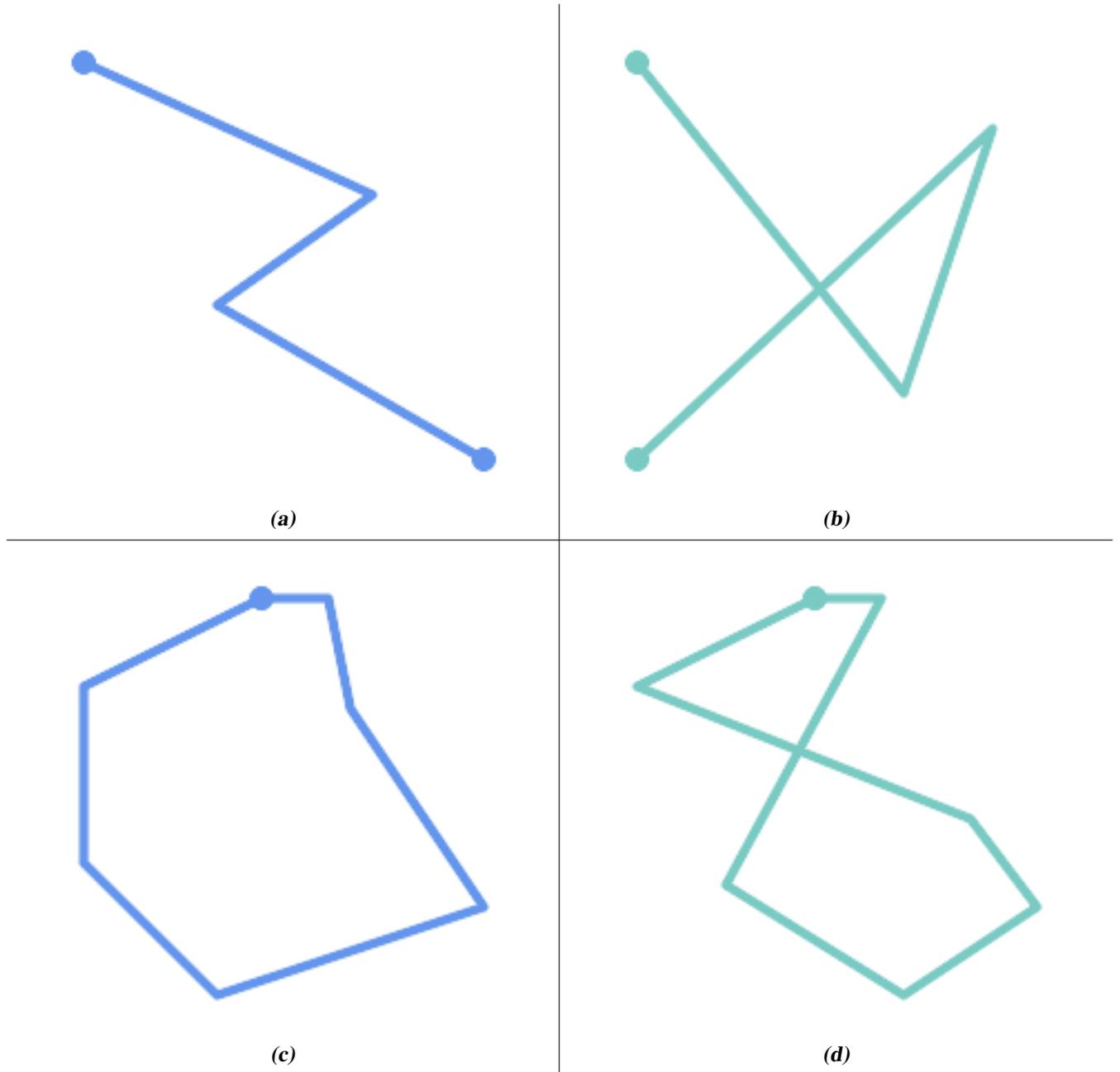
```

MULTIPOINT &#xb294; &#xc5b4;&#xb5a4; &#xb450; &#xc88c;&#xd45c;(POINT)&#xb3c4; &#xb3d9;&#xc77c;&#xd558;&#xc9c
&#xc54a;&#xc740; (&#xb3d9;&#xc77c;&#xd55c; &#xc88c;&#xd45c;&#xb97c; &#xacf5;&#xc720;&#xd558;&#xc9c0; &#xc54a;&#xc
&#xb2e8;&#xc21c;&#xd615; &#xc785;&#xb2c8;&#xb2e4;.

```

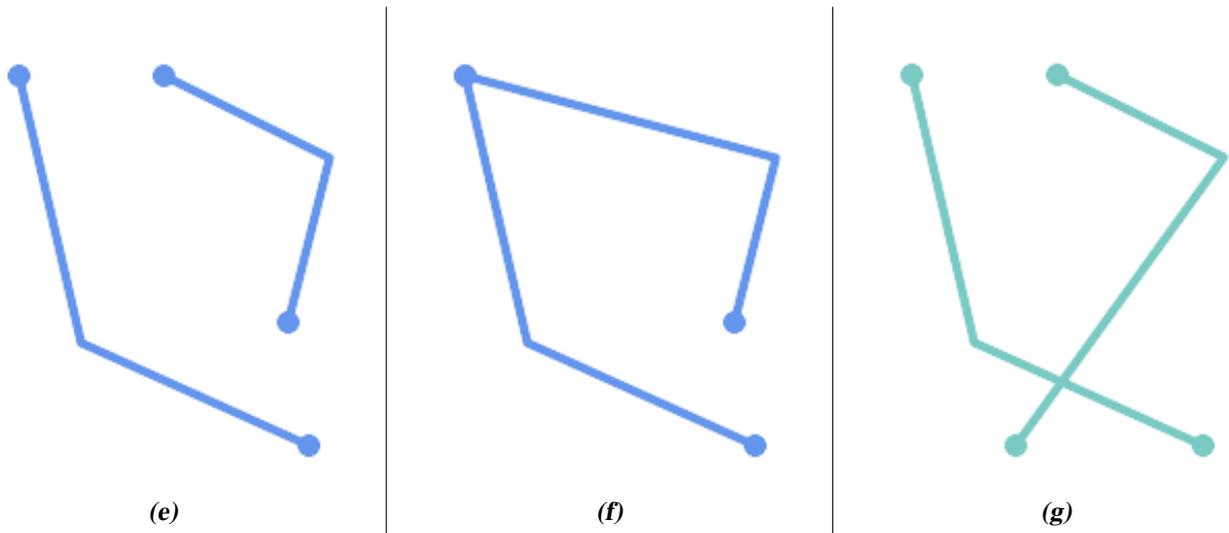
A *LINestring* is *simple* if it does not pass through the same point twice, except for the endpoints. If the endpoints of a simple LineString are identical it is called *closed* and referred to as a Linear Ring.

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



A `MULTILINESTRING` is *simple* only if all of its elements are simple and the only intersection between any two elements occurs at points that are on the boundaries of both elements.

*(e) and (f) are simple MULTILINESTRINGs. (g) is not simple.*



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

To test if a geometry is simple use the [ST\\_IsSimple](#) function:

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

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

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

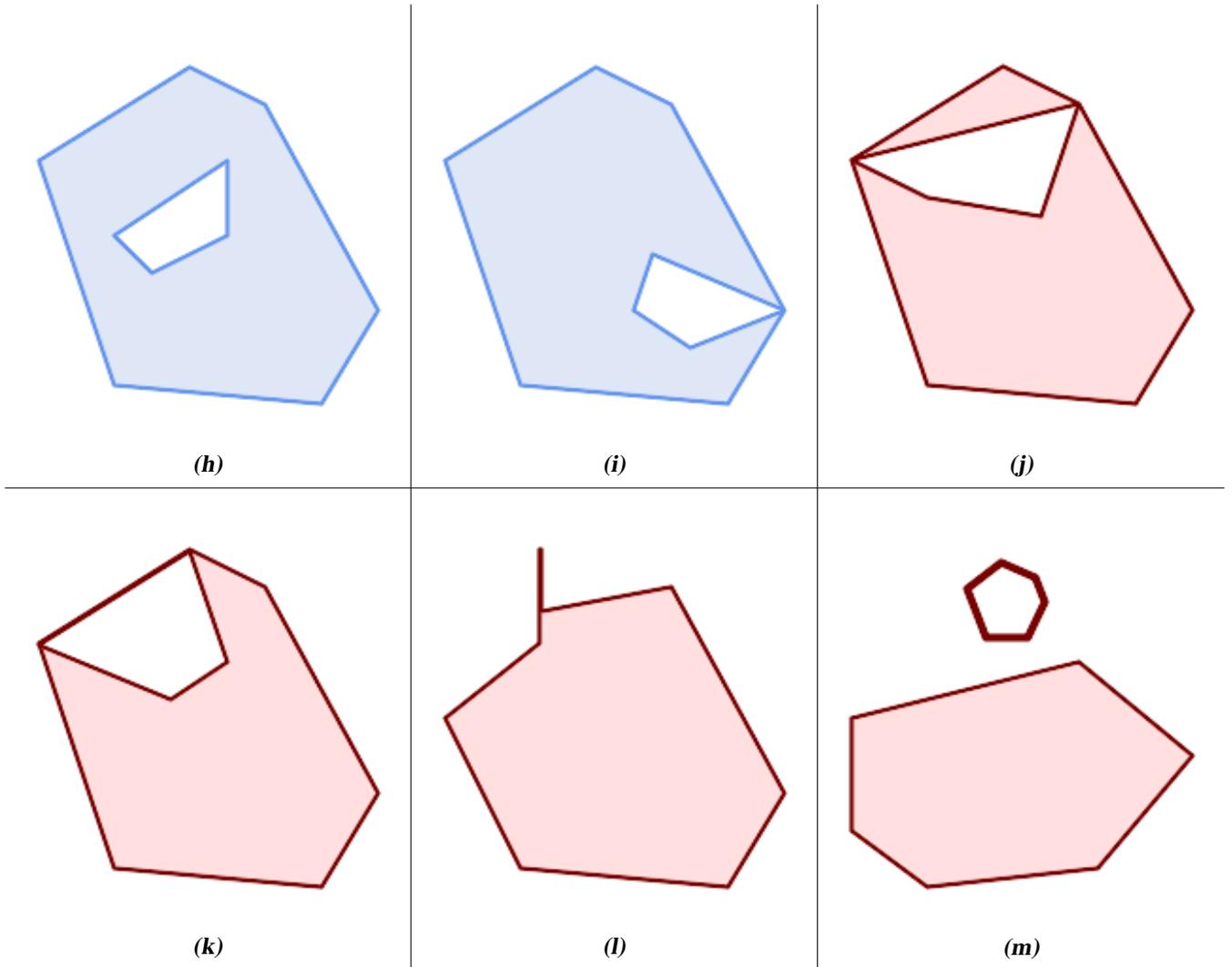
#### 4.4.2 Valid Geometry

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

A POLYGON is *valid* if:

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

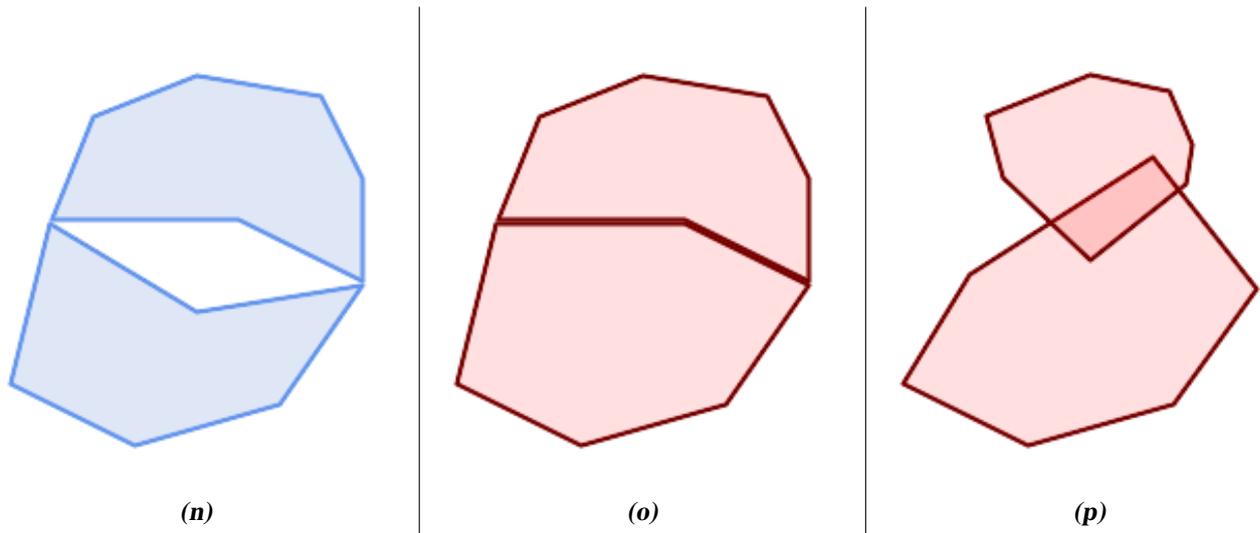
**(h)** and **(i)** are valid POLYGONS. **(j-m)** are invalid. **(j)** can be represented as a valid MULTIPOLYGON.



A MULTIPOLYGON is *valid* if:

1. its element POLYGONS are valid
2. elements do not overlap (i.e. their interiors must not intersect)
3. elements touch only at points (i.e. not along a line)

*(n)* is a *valid* MULTIPOLYGON. *(o)* and *(p)* are *invalid*.



These rules mean that valid polygonal geometry is also *simple*.

For linear geometry the only validity rule is that `LINESTRINGS` must have at least two points and have non-zero length (or equivalently, have at least two distinct points.) Note that non-simple (self-intersecting) lines are valid.

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

| len_nonzero | len_zero | self_int |
|-------------|----------|----------|
| t           | f        | t        |

`POINT` and `MULTIPOINT` geometries have no validity rules.

#### 4.4.3 Managing Validity

PostGIS allows creating and storing both valid and invalid Geometry. This allows invalid geometry to be detected and flagged or fixed. There are also situations where the OGC validity rules are stricter than desired (examples of this are zero-length linestrings and polygons with inverted holes.)

Many of the functions provided by PostGIS rely on the assumption that geometry arguments are valid. For example, it does not make sense to calculate the area of a polygon that has a hole defined outside of the polygon, or to construct a polygon from a non-simple boundary line. Assuming valid geometric inputs allows functions to operate more efficiently, since they do not need to check for topological correctness. (Notable exceptions are that zero-length lines and polygons with inversions are generally handled correctly.) Also, most PostGIS functions produce valid geometry output if the inputs are valid. This allows PostGIS functions to be chained together safely.

If you encounter unexpected error messages when calling PostGIS functions (such as "GEOS Intersection() threw an error!"), you should first confirm that the function arguments are valid. If they are not, then consider using one of the techniques below to ensure the data you are processing is valid.



#### Note

If a function reports an error with valid inputs, then you may have found an error in either PostGIS or one of the libraries it uses, and you should report this to the PostGIS project. The same is true if a PostGIS function returns an invalid geometry for valid input.

To test if a geometry is valid use the `ST_IsValid` function:

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

Information about the nature and location of an geometry invalidity are provided by the [ST\\_IsValidDetail](#) function:

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

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

In some situations it is desirable to correct invalid geometry automatically. Use the [ST\\_MakeValid](#) function to do this. ([ST\\_MakeValid](#) is a case of a spatial function that *does* allow invalid input!)

By default, PostGIS does not check for validity when loading geometry, because validity testing can take a lot of CPU time for complex geometries. If you do not trust your data sources, you can enforce a validity check on your tables by adding a check constraint:

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

## 4.5 SPATIAL\_REF\_SYS

A [Spatial Reference System](#) (SRS) (also called a Coordinate Reference System (CRS)) defines how geometry is referenced to locations on the Earth's surface. There are three types of SRS:

- A **geodetic** SRS uses angular coordinates (longitude and latitude) which map directly to the surface of the earth.
- A **projected** SRS uses a mathematical projection transformation to "flatten" the surface of the spheroidal earth onto a plane. It assigns location coordinates in a way that allows direct measurement of quantities such as distance, area, and angle. The coordinate system is Cartesian, which means it has a defined origin point and two perpendicular axes (usually oriented North and East). Each projected SRS uses a stated length unit (usually metres or feet). A projected SRS may be limited in its area of applicability to avoid distortion and fit within the defined coordinate bounds.
- A **local** SRS is a Cartesian coordinate system which is not referenced to the earth's surface. In PostGIS this is specified by a SRID value of 0.

There are many different spatial reference systems in use. Common SRSes are standardized in the European Petroleum Survey Group [EPSG database](#). For convenience PostGIS (and many other spatial systems) refers to SRS definitions using an integer identifier called a SRID.

A geometry is associated with a Spatial Reference System by its SRID value, which is accessed by [ST\\_SRID](#). The SRID for a geometry can be assigned using [ST\\_SetSRID](#). Some geometry constructor functions allow supplying a SRID (such as [ST\\_Point](#) and [ST\\_MakeEnvelope](#)). The [EWKT](#) format supports SRIDs with the `SRID=n;` prefix.

Spatial functions processing pairs of geometries (such as [overlay](#) and [relationship](#) functions) require that the input geometries are in the same spatial reference system (have the same SRID). Geometry data can be transformed into a different spatial reference system using [ST\\_Transform](#). Geometry returned from functions has the same SRS as the input geometries.

## 4.5.1 SPATIAL\_REF\_SYS Table

The `SPATIAL_REF_SYS` table used by PostGIS is an OGC-compliant database table that defines the available spatial reference systems. It holds the numeric SRIDs and textual descriptions of the coordinate systems.

`SPATIAL_REF_SYS`

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

`auth_name` The name of the authority that defines the coordinate system.

`srid` The SRID of the coordinate system. For example, 4326 is the SRID for the NAD83 datum.

`auth_name` The name of the authority that defines the coordinate system. For example, "EPSG" is the authority for the EPSG coordinate system.

`auth_srid` The ID of the Spatial Reference System as defined by the Authority cited in the `auth_name`. In the case of EPSG, this is the EPSG code.

`srtext` The Well-Known Text (WKT) representation of the coordinate system.

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

For a discussion of SRS WKT, see the OGC standard [Well-known text representation of coordinate reference systems](#).

`proj4text` The Proj4 text representation of the coordinate system. For example, `+proj=utm +zone=10 +ellps=clrk66 +datum=NAD27 +units=m`

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

For more information, see the <http://trac.osgeo.org/postgis/wiki/SpatialReferenceSystems> page.







```
&#xc788;&#xb294;&#xb370;, SQL &#xbdf0; &#xadf8;&#xb9ac;&#xace0; &#xb300;&#xaddc;&#xbaa8; &#xc0bd;&#xc785;(bulk
insert)&#xc758; &#xacbd;&#xc6b0;&#xc785;&#xb2c8;&#xb2e4;. &#xc774;&#xb7f0; &#xacbd;&#xc6b0;,&#xd574;&#xb2f9;
&#xc5f4;&#xc5d0; &#xc81c;&#xc57d; &#xc870;&#xac74;&#xc744; &#xac78;&#xc5b4;&#xc11c; geometry_columns &#xd14c;&#x
&#xb4f1;&#xb85d;&#xc744; &#xbc14;&#xb85c;&#xc7a1;&#xc744; &#xc218; &#xc788;&#xc2b5;&#xb2c8;&#xb2e4;. Post-
GIS 2.0 &#xc774;&#xc0c1; &#xbc84;&#xc804;&#xc5d0;&#xc11c;&#xb294;,&#xc0ac;&#xc6a9;&#xc790; &#xc5f4;&#xc774;
typmod &#xae30;&#xbc18;&#xc774;&#xb77c;&#xba74; &#xc0dd;&#xc131; &#xacfc;&#xc815; &#xc911;&#xc5d0; &#xc815;&#xd
&#xb4f1;&#xb85d;&#xd560; &#xac83;&#xc774;&#xae30; &#xb54c;&#xbb38;&#xc5d0; &#xc544;&#xbb34;&#xac83;&#xb3c4;
&#xd560; &#xd544;&#xc694;&#xac00; &#xc5c6;&#xb2e4;&#xb294; &#xc810;&#xc744; &#xae30;&#xc5b5;&#xd558;&#xc2ed;&#x
```

```
-- &#xc774;&#xb807;&#xac8c; &#xc0dd;&#xc131;&#xb41c; &#xbdf0;&#xac00; ←
&#xc788;&#xb2e4;&#xace0; &#xd569;&#xc2dc;&#xb2e4;.
CREATE VIEW public.vwmytablemercator AS
    SELECT gid, ST_Transform(geom,3395) As geom, f_name
    FROM public.mytable;

-- PostGIS 2.0 &#xc774;&#xc0c1; &#xbc84;&#xc804;&#xc5d0; &#xc815;&#xd655;&#xd558;&#xac8c; ←
&#xb4f1;&#xb85d;&#xd558;&#xb824;&#xba74;
-- &#xb3c4;&#xd615;&#xc744; &#xd615;&#xbcc0;&#xd658;&#xd574;&#xc57c; ←
&#xd569;&#xb2c8;&#xb2e4;.
--
DROP VIEW public.vwmytablemercator;
CREATE VIEW public.vwmytablemercator AS
    SELECT gid, ST_Transform(geom,3395)::geometry(Geometry, 3395) As geom, f_name
    FROM public.mytable;

-- &#xb3c4;&#xd615; &#xc720;&#xd615;&#xc774; 2D &#xd3f4;&#xb9ac;&#xace4;&#xc774;&#xb780; ←
&#xc0ac;&#xc2e4;&#xc744; &#xd655;&#xc2e4;&#xd788; &#xc54c;&#xace0; &#xc788;&#xc744; ←
&#xacbd;&#xc6b0; &#xb2e4;&#xc74c;&#xacfc; &#xac19;&#xc774; &#xd560; &#xc218; ←
&#xc788;&#xc2b5;&#xb2c8;&#xb2e4;.
DROP VIEW public.vwmytablemercator;
CREATE VIEW public.vwmytablemercator AS
    SELECT gid, ST_Transform(geom,3395)::geometry(Polygon, 3395) As geom, f_name
    FROM public.mytable;
```

```
-- &#xb300;&#xaddc;&#xbaa8; &#xc0bd;&#xc785; &#xc791;&#xc5c5;&#xc744; &#xd1b5;&#xd574; ←
&#xd30c;&#xc0dd; &#xd14c;&#xc774;&#xbel4;&#xc744; ←
&#xc0dd;&#xc131;&#xd588;&#xb2e4;&#xace0; &#xd569;&#xc2dc;&#xb2e4;.
SELECT poi.gid, poi.geom, citybounds.city_name
INTO myschema.my_special_pois
FROM poi INNER JOIN citybounds ON ST_Intersects(citybounds.geom, poi.geom);

-- &#xc0c8; &#xd14c;&#xc774;&#xbel4;&#xc5d0; 2D &#xc778;&#xb371;&#xc2a4;&#xb97c; ←
&#xc0dd;&#xc131;&#xd569;&#xb2c8;&#xb2e4;.
CREATE INDEX idx_myschema_myspecialpois_geom_gist
ON myschema.my_special_pois USING gist(geom);

-- &#xc0ac;&#xc6a9;&#xc790; &#xd3ec;&#xc778;&#xd2b8;&#xac00; 3D &#xb610;&#xb294; 3M ←
&#xd3ec;&#xc778;&#xd2b8;&#xc77c; &#xacbd;&#xc6b0;,
-- 2D &#xc778;&#xb371;&#xc2a4;&#xac00; &#xc544;&#xb2c8;&#xb77c; nD ←
&#xc778;&#xb371;&#xc2a4;&#xb97c; &#xc0dd;&#xc131;&#xd558;&#xb294; &#xd3b8;&#xc774; ←
&#xc88b;&#xc744; &#xc218;&#xb3c4; &#xc788;&#xc2b5;&#xb2c8;&#xb2e4;.
-- &#xb2e4;&#xc74c;&#xcc98;&#xb7fc; &#xb9d0;&#xc774;&#xc8e0;.
CREATE INDEX my_special_pois_geom_gist_nd
ON my_special_pois USING gist(geom gist_geometry_ops_nd);

-- &#xc774; &#xc0c8; &#xd14c;&#xc774;&#xbel4;&#xc758; &#xb3c4;&#xd615; &#xc5f4;&#xc744; ←
geometry_columns &#xd14c;&#xc774;&#xbel4;&#xc5d0; &#xc9c1;&#xc811; ←
&#xb4f1;&#xb85d;&#xd558;&#xb294;
-- &#xb2e4;&#xc74c; &#xbc29;&#xbc95;&#xc740; PostGIS 2.0 &#xc774;&#xd6c4; &#xbc84;&#xc804; ←
&#xbcf0; 1.4 &#xc774;&#xd6c4; &#xbc84;&#xc804; &#xbaa8;&#xb450;&#xc5d0;&#xc11c; ←
&#xb3d9;&#xc791;&#xd569;&#xb2c8;&#xb2e4;.
```

```

-- PostGIS 2.0 &#x0ac;&#xc6a9;&#xc790; &#xc9c0;&#xce68;&#xc11c;
&#xad6c;&#xc2dd; &#xc81c;&#xc57d;&#xc870;&#xac74; &#xae30;&#xbc18; &#xc815;&#xc758; &#xb3d9;&#xc791;&#xc774; &#xd544;&#xc694;&#xd55c; &#xacbd;&#xc6b0;
&#xbaa8;&#xb4e0; &#xc790;&#xc2dd; &#xac1d;&#xccb4;&#xac00; &#xb3d9;&#xc77c;&#xd55c; &#xc720;&#xd615;&#xacfc; SRID&#xac00; &#xc544;&#xb2cc; &#xc0c1;&#xc18d; &#xd14c;&#xc774;&#xbe14;&#xc758; &#xacbd;&#xc6b0; &#xb4f1;)
&#xc120;&#xd0dd;&#xc801;&#xc778; &#xc0c8; use_typmod argument
&#xd30c;&#xb77c;&#xbbf8;&#xd130;&#xb97c; &#xac70;&#xc9d3;&#xc73c;&#xb85c; &#xc124;&#xc815;&#xd558;&#xc2ed;&#xc2dc;&#xc624;.
SELECT populate_geometry_columns('myschema.my_special_pois'::regclass);

-- PostGIS 2.0 &#xc84;&#xc804;&#xc758; &#xacbd;&#xc6b0; &#xc5f4;&#xc744; typmod &#xae30;&#xbc18;&#xc73c;&#xb85c; &#xb9cc;&#xb4e4;&#xae30; &#xc704;&#xd574;
&#xd14c;&#xc774;&#xbe14;&#xc758; &#xae30;&#xc800; &#xad6c;&#xc870;&#xb97c; &#xbcc0;&#xacbd;&#xd560; &#xc83;&#xc785;&#xb2c8;&#xb2e4;.
-- PostGIS 2.0 &#xc774;&#xc804; &#xc84;&#xc804;&#xc758; &#xacbd;&#xc6b0;, &#xb3d9;&#xc77c;&#xd55c; &#xc29;&#xc95;&#xc73c;&#xb85c; &#xbdf0;&#xb97c; &#xb4f1;&#xb85d;&#xd560; &#xc218;&#xb3c4; &#xc788;&#xc2b5;&#xb2c8;&#xb2e4;.
SELECT populate_geometry_columns('myschema.my_special_pois'::regclass);

-- PostGIS 2.0 &#xc84;&#xc804;&#xc744; &#xc0ac;&#xc6a9;&#xc911;&#xc774;&#xace0; &#xc5b4;&#xb5a4; &#xc774;&#xc720;&#xc5d0;&#xc11c;&#xb4e0;
&#xad6c;&#xc2dd; &#xc81c;&#xc57d;&#xc870;&#xac74; &#xae30;&#xbc18; &#xc815;&#xc758; &#xb3d9;&#xc791;&#xc774; &#xd544;&#xc694;&#xd55c; &#xacbd;&#xc6b0;
(&#xbaa8;&#xb4e0; &#xc790;&#xc2dd; &#xac1d;&#xccb4;&#xac00; &#xb3d9;&#xc77c;&#xd55c; &#xc720;&#xd615;&#xacfc; SRID&#xac00; &#xc544;&#xb2cc; &#xc0c1;&#xc18d; &#xd14c;&#xc774;&#xbe14;&#xc758; &#xacbd;&#xc6b0; &#xb4f1;)
&#xc120;&#xd0dd;&#xc801;&#xc778; &#xc0c8; use_typmod argument
&#xd30c;&#xb77c;&#xbbf8;&#xd130;&#xb97c; &#xac70;&#xc9d3;&#xc73c;&#xb85c; &#xc124;&#xc815;&#xd558;&#xc2ed;&#xc2dc;&#xc624;.
SELECT populate_geometry_columns('myschema.my_special_pois'::regclass, false);

```

```

&#xad6c;&#xc2dd; &#xc81c;&#xc57d;&#xc870;&#xac74; &#xae30;&#xbc18; &#xc815;&#xc758; &#xb3d9;&#xc791;&#xc774; &#xd544;&#xc694;&#xd55c; &#xacbd;&#xc6b0;
&#xbaa8;&#xb4e0; &#xc790;&#xc2dd; &#xac1d;&#xccb4;&#xac00; &#xb3d9;&#xc77c;&#xd55c; &#xc720;&#xd615;&#xacfc; SRID&#xac00; &#xc544;&#xb2cc; &#xc0c1;&#xc18d; &#xd14c;&#xc774;&#xbe14;&#xc758; &#xacbd;&#xc6b0; &#xb4f1;)
&#xc120;&#xd0dd;&#xc801;&#xc778; &#xc0c8; use_typmod argument
&#xd30c;&#xb77c;&#xbbf8;&#xd130;&#xb97c; &#xac70;&#xc9d3;&#xc73c;&#xb85c; &#xc124;&#xc815;&#xd558;&#xc2ed;&#xc2dc;&#xc624;.
SELECT populate_geometry_columns('myschema.my_special_pois'::regclass, false);

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

```

PSQL

```
\d pois_ny;
```

```

&#xb450; &#xc5f4;&#xc774; &#xc11c;&#xb85c; &#xb2e4;&#xb974;&#xac8c; &#xc815;&#xc758;&#xb418;&#xc5c8;&#xb2e4;&#xc0ac;&#xc2e4;&#xc744; &#xc54c; &#xc218; &#xc788;&#xc2b5;&#xb2c8;&#xb2e4;. &#xd558;&#xb098;&#xb294; typmod, &#xb2e4;&#xb978; &#xd558;&#xb098;&#xb294; &#xc81c;&#xc57d;&#xc870;&#xac74;&#xc73c;&#xb85c; &#xc815;&#xc758;

```

```

Table "public.pois_ny"
Column | Type | Modifiers
-----+-----+-----
gid | integer | not null default nextval('pois_ny_gid_seq'::regclass)
poi_name | text |
cat | character varying(20) |
geom | geometry(Point,4326) |
geom_2160 | geometry |

```

Indexes:

```
"pois_ny_pkey" PRIMARY KEY, btree (gid)
```

Check constraints:

```
"enforce_dims_geom_2160" CHECK (st_ndims(geom_2160) = 2)
```

```
"enforce_geotype_geom_2160" CHECK (geometrytype(geom_2160) = 'POINT'::text
OR geom_2160 IS NULL)
```

```
"enforce_srid_geom_2160" CHECK (st_srid(geom_2160) = 2160)
```

```

&#xb458; &#xb2e4; geometry_columns &#xd14c;&#xc774;&#xbe14;&#xc5d0; &#xc815;&#xd655;&#xd558;&#xac8c; &#xb4f1;&#xc774;

```



SQL `CREATE TABLE roads (road_id SERIAL, road_name VARCHAR(255), road_geom GEOMETRY(Point, 4326));` `INSERT INTO roads (road_id, road_name, road_geom) VALUES (1, 'Jeff Rd', ST_GeomFromText('LINESTRING(191232 243118,191108 243242)', -1));` `INSERT INTO roads (road_id, road_name, road_geom) VALUES (2, 'Geordie Rd', ST_GeomFromText('LINESTRING(189141 244158,189265 244817)', -1));` `INSERT INTO roads (road_id, road_name, road_geom) VALUES (3, 'Paul St', ST_GeomFromText('LINESTRING(192783 228138,192612 229814)', -1));` `INSERT INTO roads (road_id, road_name, road_geom) VALUES (4, 'Graeme Ave', ST_GeomFromText('LINESTRING(189412 252431,189631 259122)', -1));` `INSERT INTO roads (road_id, road_name, road_geom) VALUES (5, 'Phil Tce', ST_GeomFromText('LINESTRING(190131 224148,190871 228134)', -1));` `INSERT INTO roads (road_id, road_name, road_geom) VALUES (6, 'Dave Cres', ST_GeomFromText('LINESTRING(198231 263418,198213 268322)', -1));` `COMMIT;`

#### 4.7.1 SQL: ESRI shapefile to PostgreSQL

ESRI shapefile to PostgreSQL: `ogr2ogr -f PostgreSQL PG:"dbname='roads' host='localhost' user='postgres' password='postgres' geometry_name=roads" roads.shp`

ESRI shapefile to PostgreSQL: `ogr2ogr -f PostgreSQL PG:"dbname='roads' host='localhost' user='postgres' password='postgres' geometry_name=roads" roads.shp`

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

"psql" SQL: `psql -d [database] -f roads.sql`

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

#### 4.7.2 shp2pgsql: ESRI shapefile to PostgreSQL

ESRI shapefile to PostgreSQL: `ogr2ogr -f PostgreSQL PG:"dbname='roads' host='localhost' user='postgres' password='postgres' geometry_name=roads" roads.shp`

ESRI shapefile to PostgreSQL: `ogr2ogr -f PostgreSQL PG:"dbname='roads' host='localhost' user='postgres' password='postgres' geometry_name=roads" roads.shp`

```

claldlp -- &#xc774;&#xb4e4;&#xc740; &#xc0c1;&#xd638;&#xbc30;&#xd0c0;&#xc801;&#xc778; &#xc635;&#xc158;&#xb
-c &#xc0c8; &#xd14c;&#xc774;&#xbe14;&#xc744; &#xc0dd;&#xc131;&#xd55c; &#xb2e4;&#xc74c; shapefile&#xc758;
    &#xb370;&#xc774;&#xd130;&#xb85c; &#xd574;&#xb2f9; &#xd14c;&#xc774;&#xbe14;&#xc744; &#xcc44;&#xc6c1;&
    &#xc774;&#xac83;&#xc774; &#xae30;&#xbcf8; &#xbaa8;&#xb4dc;&#xc785;&#xb2c8;&#xb2e4;.
-a &#xae30;&#xc874; &#xb370;&#xc774;&#xd130;&#xbca0;&#xc774;&#xc2a4; &#xd14c;&#xc774;&#xbe14;&#xc5d0;
    shapefile&#xc758; &#xb370;&#xc774;&#xd130;&#xb97c; &#xcd94;&#xac00;&#xd569;&#xb2c8;&#xb2e4;. &#xc774;
    &#xc635;&#xc158;&#xc744; &#xc774;&#xc6a9;&#xd574;&#xc11c; &#xbcf5;&#xc218;&#xc758; &#xd30c;&#xc77c;&
    &#xb85c;&#xb4dc;&#xd558;&#xb824;&#xba74;. &#xd30c;&#xc77c;&#xb4e4;&#xc774; &#xb3d9;&#xc77c;&#xd55c;
    &#xc18d;&#xc131; &#xbc0f; &#xb3d9;&#xc77c;&#xd55c; &#xb370;&#xc774;&#xd130; &#xc720;&#xd615;&#xc744;
    &#xb2f4;&#xcace0; &#xc788;&#xc5b4;&#xc57c; &#xd55c;&#xb2e4;&#xb294; &#xc810;&#xc744; &#xc8fc;&#xc758;&
-d &#xae30;&#xc874; &#xb370;&#xc774;&#xd130;&#xbca0;&#xc774;&#xc2a4; &#xd14c;&#xc774;&#xbe14;&#xc744;
    &#xc0ad;&#xc81c;(drop)&#xd55c; &#xb2e4;&#xc74c; shapefile&#xc758; &#xb370;&#xc774;&#xd130;&#xb97c;
    &#xac00;&#xc9c4; &#xc0c8; &#xd14c;&#xc774;&#xbe14;&#xc744; &#xc0dd;&#xc131;&#xd569;&#xb2c8;&#xb2e4;.
-p &#xd14c;&#xc774;&#xbe14;&#xc744; &#xc0dd;&#xc131;&#xd558;&#xb294; SQL&#xc754;&#xb4dc;&#xb9cc; &#xc0dd
    &#xc5b4;&#xb5a4; &#xc2e4;&#xc81c; &#xb370;&#xc774;&#xd130;&#xb3c4; &#xcd94;&#xac00;&#xd558;&#xc9c0;
    &#xc54a;&#xc2b5;&#xb2c8;&#xb2e4;. &#xd14c;&#xc774;&#xbe14; &#xc0dd;&#xc131;&#xcacf; &#xb370;&#xc774;&
    &#xb85c;&#xb4dc; &#xb2e8;&#xacc4;&#xb97c; &#xc644;&#xc804;&#xd788; &#xbd84;&#xb9ac;&#xd574;&#xc57c;
    &#xd560; &#xacbd;&#xc6b0; &#xc0ac;&#xc6a9;&#xd560; &#xc218; &#xc788;&#xc2b5;&#xb2c8;&#xb2e4;.
-? &#xb3c4;&#xc6c0;&#xb9d0; &#xd654;&#xba74;&#xc744; &#xd45c;&#xcd9c;&#xd569;&#xb2c8;&#xb2e4;.
-D &#xc0b0;&#xcd9c;&#xbb3c; &#xb370;&#xc774;&#xd130;&#xc758; &#xd615;&#xc2dd;&#xc73c;&#xb85c; PostgreSQL
    "&#xb364;&#xd504;(dump)"&#xd615;&#xc2dd;&#xc744; &#xc0ac;&#xc6a9;&#xd569;&#xb2c8;&#xb2e4;. &#xc774;
    &#xc635;&#xc158;&#xc740; -a, -c &#xbc0f; -d&#xc640; &#xd568;&#xaed8; &#xc0ac;&#xc6a9;&#xd560; &#xc218;
    &#xc788;&#xc2b5;&#xb2c8;&#xb2e4;. &#xc774; &#xb364;&#xd504; &#xd615;&#xc2dd;&#xc740; &#xae30;&#xbcf8;
    "&#xc0bd;&#xc785;" SQL&#xd615;&#xc2dd;&#xbcf4;&#xb2e4; &#xd6e8;&#xc52c; &#xbe68;&#xb9ac; &#xb85c;&#xb4dc;
    &#xc218; &#xc788;&#xc2b5;&#xb2c8;&#xb2e4;. &#xb300;&#xc6a9;&#xb7c9; &#xb370;&#xc774;&#xd130;&#xc14b;&#xc7
    &#xacbd;&#xc6b0; &#xc774; &#xc635;&#xc158;&#xc744; &#xc0ac;&#xc6a9;&#xd558;&#xc2ed;&#xc2dc;&#xc624;.
-s [<FROM_SRID>:<SRID>] &#xb3c4;&#xd615; &#xd14c;&#xc774;&#xbe14;&#xc744; &#xc0dd;&#xc131;&#xd558;&#xcac
    &#xc9c0;&#xc815;&#xb41c; SRID&#xb85c; &#xcc44;&#xc6c1;&#xb2c8;&#xb2e4;. &#xc785;&#xb825; shapefile&#xc774;
    &#xc8fc;&#xc5b4;&#xc9c4; FROM_SRID&#xb97c; &#xc4f0;&#xb3c4;&#xb85d; &#xc124;&#xc815;&#xd558;&#xb294;
    &#xc635;&#xc158;&#xb3c4; &#xc788;&#xc2b5;&#xb2c8;&#xb2e4;. &#xc774;&#xb7f0; &#xacbd;&#xc6b0; &#xb3c4;&#xd6
    &#xbaa9;&#xd45c; SRID&#xb85c; &#xc7ac;&#xd22c;&#xc601;&#xb420; &#xac83;&#xc785;&#xb2c8;&#xb2e4;. FROM_SR
-D &#xc635;&#xc158;&#xcacf; &#xd568;&#xaed8; &#xc0ac;&#xc6a9;&#xb420; &#xc218; &#xc5c6;&#xc2b5;&#xb2c8;&#xb
-k &#xc2dd;&#xbcc4;&#xc790;&#xc758; &#xb300;&#xc18c;&#xbb38;&#xc790;(&#xc5f4;. &#xc2a4;&#xd0a4;&#xb9c8; &#xbc0f;
    &#xc18d;&#xc131;)&#xb97c; &#xc720;&#xc9c0;&#xd569;&#xb2c8;&#xb2e4;. shapefile &#xc548;&#xc758; &#xc18d;&#xc1
    &#xbaa8;&#xb450; &#xb300;&#xbb38;&#xc790;&#xb77c;&#xb294; &#xc810;&#xc744; &#xc8fc;&#xc758;&#xd558;&#xc2
-i DBF&#xd5e4;&#xb354; &#xc11c;&#xba85;&#xc774; 64&#xbe44;&#xd2b8; bigint&#xd615;&#xc2dd;&#xc744; &#xbf4;&#xc7
    &#xbaa8;&#xb4e0; &#xc815;&#xc218;&#xb97c; &#xd45c;&#xc900; 32&#xbe44;&#xd2b8; &#xc815;&#xc218;&#xb85c;
    &#xac15;&#xc81c; &#xbcc0;&#xd658;&#xd558;&#xcace0; 64&#xbe44;&#xd2b8; bigint &#xd615;&#xc2dd;&#xc744;
    &#xc0dd;&#xc131;&#xd558;&#xc9c0; &#xc54a;&#xc2b5;&#xb2c8;&#xb2e4;.
-I &#xb3c4;&#xd615; &#xc5f4;&#xc5d0; GiST&#xc778;&#xb371;&#xc2a4;&#xb97c; &#xc0dd;&#xc131;&#xd569;&#xb2c8;&#xb
-m "-m &#xd30c; &#xc77c; &#xba85;" &#xd615;&#xc2dd;&#xc73c;&#xb85c; (&#xae34;) &#xc5f4; &#xba85;&#xc6ed;&#xcac
    10&#xbb38;&#xc790; DBF&#xc5f4; &#xba85;&#xc6ed;&#xc744; &#xb9e4;&#xd551;&#xd558;&#xb294; &#baa9;&#xb85d
    &#xb2f4;&#xc740; &#xd30c;&#xc77c;&#xc744; &#xc9c0;&#xc815;&#xd569;&#xb2c8;&#xb2e4;. &#xc774; &#xd30c;&#xc7
    &#xb0b4;&#xc6a9;&#xc740; &#xcacf5;&#xb31;&#xc73c;&#xb85c; &#xad6c;&#xbd84;&#xb41c; &#xb450; &#ba85;&#xc6e
    &#xc774;&#xb8e8;&#xc5b4;&#xc9c4; &#xd558;&#xb098; &#xc774;&#xc0c1;&#xc758; &#xd589;&#xc73c;&#xb85c;.
    &#xd589; &#xb9e8; &#xc55e;&#xcacf; &#xb9e8; &#xb4a4;&#xc5d0;&#xb294; &#xcacf5;&#xb31;&#xc774; &#xc5c6;&#xc5
    &#xd569;&#xb2c8;&#xb2e4;. &#xb2e4;&#xc74c;&#xc740; &#xadf8; &#xc608;&#xc2dc;&#xc785;&#xb2c8;&#xb2e4;.

```

```

COLUMNNAME DBFFIELD1
AVERYLONGCOLUMNNAME DBFFIELD2

```





```

6 | LINESTRING (198231 263418,198213 268322) | Dave Cres
7 | LINESTRING (218421 284121,224123 241231) | Chris Way
(6 rows)

```

```

&#xd558;&#xc9c0;&#xb9cc;, &#xbc18;&#xd658;&#xb418;&#xb294; &#xd544;&#xb4dc;&#xc758; &#xac1c;&#xc218;&#xb97c;
&#xc904;&#xc774;&#xae30; &#xc704;&#xd574; &#xc5b4;&#xb5a4; &#xc885;&#xb958;&#xc758; &#xc81c;&#xc57d;&#xc774;
&#xd544;&#xc694;&#xd560; &#xb54c;&#xac00; &#xc788;&#xc744; &#xac83;&#xc785;&#xb2c8;&#xb2e4;. &#xc18d;&#xc131;
&#xae30;&#xbc18; &#xc81c;&#xc57d;&#xc758; &#xacbd;&#xc6b0; &#xc77c;&#xbc18;&#xc801;&#xc778; &#xbe44;&#xacf5;&#
&#xd14c;&#xc774;&#xbe14;&#xc758; &#xacbd;&#xc6b0;&#xc640; &#xb3d9;&#xc77c;&#xd55c; SQL &#bb38;&#xc95;&#xc74
&#xc4f0;&#xba74; &#xb429;&#xb2c8;&#xb2e4;. &#xacf5;&#xac04; &#xc81c;&#xc57d;&#xc758; &#xacbd;&#xc6b0;, &#xb2e4;&#
&#xc720;&#xc6a9;&#xd55c; &#xc5f0;&#xc0b0;&#xc790;&#xb4e4;&#xc744; &#xc4f8; &#xc218; &#xc788;&#xc2b5;&#xb2c8;&#

```

**ST\_Intersects** This function tells whether two geometries share any space.

```

= &#xc774; &#xc5f0;&#xc0b0;&#xc790;&#xb294; &#xb450; &#xb3c4;&#xd615;&#xc774; &#xae30;&#xd558;&#xd559;&#xc801;
&#xb3d9;&#xc77c;&#xd55c;&#xc9c0;&#xb97c; &#xd14c;&#xc2a4;&#xd2b8;&#xd569;&#xb2c8;&#xb2e4;. &#xc608;&#xb97
&#xb4e4;&#xc5b4;, 'POLYGON((0 0,1 1,0 0,0 0))' &#xacfc; 'POLYGON((0 0,1 1,0 0,0 0))' &#xc774; &#xb3d9;&#xc77c;&#xd5
&#xb9d0;&#xc785;&#xb2c8;&#xb2e4;(&#xb3d9;&#xc77c;&#xd569;&#xb2c8;&#xb2e4;).

```

```

&#xb2e4;&#xc74c;&#xc73c;&#xb85c;, &#xc774; &#xc5f0;&#xc0b0;&#xc790;&#xb4e4;&#xc744; &#xcffc;&#xb9ac;&#xc5d0;
&#xc4f8; &#xc218; &#xc788;&#xc2b5;&#xb2c8;&#xb2e4;. SQL &#xba85;&#xb839;&#xd589;&#xc5d0; &#xb3c4;&#xd615;&#xac
&#xacbd;&#xac4; &#xc0c1;&#xc790;&#xb97c; &#xc9c0;&#xc815;&#xd560; &#xb54c;, "ST_GeomFromText()" &#xd568;&#xc218
&#xc774;&#xc6a9;&#xd574;&#xc11c; &#xc2a4;&#xd2b8;&#xb9c1; &#xd45c;&#xd604;&#xc2dd;&#xc744; &#xb3c4;&#xd615;&#
&#xc815;&#xd655;&#xd558;&#xac8c; &#xbcc0;&#xd658;&#xc2dc;&#xcfc1c;&#xc57c; &#xd569;&#xb2c8;&#xb2e4;. &#xd574;&#
&#xb370;&#xc774;&#xd130;&#xc640; &#xc77c;&#xc5e8;&#xd558;&#xb294; &#xac00;&#xacf5;&#xc758; &#xacf5;&#xac04;
&#xcc38;&#xc870; &#xc2dc;&#xc2a4;&#xd15c;&#xc740; 312&#xc785;&#xb2c8;&#xb2e4;. &#xb2e4;&#xc74c;&#xc740;
&#xadf8; &#xc608;&#xc2dc;&#xc785;&#xb2c8;&#xb2e4;

```

```

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

```

```

&#xc774; &#xcffc;&#xb9ac;&#xb294; &#xd574;&#xb2f9; &#xac12;&#xacfc; &#xb3d9;&#xc77c;&#xd55c; &#xb3c4;&#xd615;&#
&#xb2f4;&#xae0; &#xc788;&#xb294; "ROADS_GEOM" &#xd14c;&#xc774;&#xbe14;&#xb85c;&#xbd80;&#xd130; &#xb2e8;&#
&#xb808;&#xcf54;&#xb4dc;&#xb97c; &#bc18;&#xd658;&#xd560; &#xac83;&#xc785;&#xb2c8;&#xb2e4;.

```

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

```

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

```

```

&#xac00;&#xc7a5; &#xd754;&#xd55c; &#xacf5;&#xac04; &#xcffc;&#xb9ac;&#xb294; &#xc544;&#xb9c8;&#xb3c4; &#xb370;&#
&#xbe0c;&#xb77c;&#xc6b0;&#xc800; &#xb610;&#xb294; &#xc6f9; &#xb9e4;&#xd37c; &#xac19;&#xc740; &#xd074;&#xb77c;&#
&#xc18c;&#xd504;&#xd2b8;&#xc6e8;&#xc5b4;&#xac00; &#xd654;&#xba74; &#xd45c;&#xcd9c;&#xc744; &#xc704;&#xd574;
&##xb9f5; &#xd504;&#xb808;&#xc784;(map frame)" &#xc6a9;&#xb7c9;&#xc5d0; &#xd574;&#xb2f9;&#xd558;&#xb294;
&#xb370;&#xc774;&#xd130;&#xb97c; &#xac00;&#xc838;&#xc624;&#xae30; &#xc704;&#xd574; &#xc0ac;&#xc6a9;&#xd558;&#
&#xd504;&#xb808;&#xc784; &#xae30;&#bc18;(frame-based)" &#xcffc;&#xb9ac;&#xc77c; &#xac83;&#xc785;&#xb2c8;&#xb2e4
"&#" &#xc5f0;&#xc0b0;&#xc790; &#xc0ac;&#xc6a9;&#xc2dc;, &#xbe44;&#xad50; &#xd53c;&#xcc98;&#xb85c; BOX3D
&#xb610;&#xb294; &#xb3c4;&#xd615;&#xc744; &#xc9c0;&#xc815;&#xd560; &#xc218; &#xc788;&#xc2b5;&#xb2c8;&#xb2e4;.
&#xd558;&#xc9c0;&#xb9cc; &#xb3c4;&#xd615;&#xc744; &#xc9c0;&#xc815;&#xd588;&#xc744; &#xacbd;&#xc6b0;, &#be44;&#
&#xc791;&#xc5c5;&#xc5d0; &#xd574;&#xb2f9; &#xacbd;&#xac4; &#xc0c1;&#xc790;&#xac00; &#xc0ac;&#xc6a9;&#xb420;
&#xac83;&#xc785;&#xb2c8;&#xb2e4;.

```

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

```

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

```

&#xd654;&#xba74;&#xc5d0;&#xd574;&#xb2f9;&#xd558;&#xb294;&#xb370;&#xc774;&#xd130;&#xc758;&#xd22c;&#xc601;&#xc9c0;&#xc815;&#xd558;&#xb294;&#xb370; SRID 312&#xb97c;&#xc37c;&#xb2e4;&#xb294;&#xc0ac;&#xc2e4;&#xc5d0;&#xc8fc;&#xc758;&#xd558;&#xc2ed;&#xc2dc;&#xc624;.

#### 4.8.2 &#xb364;&#xd37c;&#xc774;&#xc6a9;&#xd558;&#xae30;

pgsql2shp &#xd14c;&#xc774;&#xbe14;&#xb364;&#xd37c;&#xb294;&#xb370;&#xc774;&#xd130;&#xbca0;&#xc774;&#xc2a4;&#xc9c1;&#xc811;&#xc5f0;&#xacb0;&#xb418;&#xc5b4;&#xc544;&#xb9c8;&#xb3c4;&#xcffc;&#xb9ac;&#xac00;&#xc815;&#xd14c;&#xc774;&#xbe14;&#xc744; shapefile&#xb85c;&#xbcc0;&#xd658;&#xd569;&#xb2c8;&#xb2e4;. &#xae30;&#xbcf5;&#xbb38;&#xbc95;&#xc740;&#xb2e4;&#xc74c;&#xacfc;&#xac19;&#xc2b5;&#xb2c8;&#xb2e4;:

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

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

&#xb2e4;&#xc74c;&#xacfc;&#xac19;&#xc740;&#xba85;&#xb839;&#xd589;&#xc635;&#xc158;&#xc774;&#xc788;&#xc2b5;&#xc791;

**-f <filename>** &#xd2b9;&#xc815;&#xd30c;&#xc77c;&#xba85;&#xc73c;&#xb85c;&#xcd9c;&#xb825;&#xbb3c;&#xc744;&#xc791;

**-h <host>** &#xc5f0;&#xacb0;&#xd560;&#xb370;&#xc774;&#xd130;&#xbca0;&#xc774;&#xc2a4;&#xd638;&#xc2a4;&#xd2b8;&#xc124;&#xc815;&#xd569;&#xb2c8;&#xb2e4;.

**-p <port>** &#xb370;&#xc774;&#xd130;&#xbca0;&#xc774;&#xc2a4;&#xd638;&#xc2a4;&#xd2b8;&#xc5f0;&#xacb0;&#xc2dc;&#xc0ac;&#xc6a9;&#xd560;&#xd3ec;&#xd2b8;&#xb97c;&#xc124;&#xc815;&#xd569;&#xb2c8;&#xb2e4;.

**-P <password>** &#xb370;&#xc774;&#xd130;&#xbca0;&#xc774;&#xc2a4;&#xc5f0;&#xacb0;&#xc5d0;&#xc0ac;&#xc6a9;&#xd560;&#xbe44;&#xbc00;&#xbc88;&#xd638;&#xb97c;&#xc124;&#xc815;&#xd569;&#xb2c8;&#xb2e4;.

**-u <user>** &#xb370;&#xc774;&#xd130;&#xbca0;&#xc774;&#xc2a4;&#xc5f0;&#xacb0;&#xc5d0;&#xc0ac;&#xc6a9;&#xd560;&#xc0ac;&#xc6a9;&#xc790;&#xba85;&#xc744;&#xc124;&#xc815;&#xd569;&#xb2c8;&#xb2e4;.

**-g <geometry column>** &#xbcf5;&#xc218;&#xc758;&#xb3c4;&#xd615;&#xc5f4;&#xc744;&#xac00;&#xc9c4;&#xd14c;&#xc774;&#xacbd;&#xc6b0;., shapefile &#xc791;&#xc131;&#xc5d0;&#xc774;&#xc6a9;&#xb420;&#xb3c4;&#xd615;&#xc5f4;&#xc774;&#xc124;&#xc815;&#xd569;&#xb2c8;&#xb2e4;.

**-b** &#xbc14;&#xc774;&#xb108;&#xb9ac;&#xcce4;&#xc11c;&#xb97c;&#xc0ac;&#xc6a9;&#xd558;&#xb3c4;&#xb85d;&#xc124;&#xc774;&#xc635;&#xc158;&#xc744;&#xc4f0;&#xba74;&#xc2e4;&#xd589;&#xc18d;&#xb3c4;&#xac00;&#xbe68;&#xb7&#xd14c;&#xc774;&#xbe14;&#xc548;&#xc5d0;&#xc788;&#xb294;&#xbe44;(&#x975e;)&#xb3c4;&#xd615;&#xc18d;&#xc&#xac00;&#xc6b4;&#xb370;&#xd558;&#xb098;&#xb77c;&#xb3c4;&#xd14d;&#xc2a4;&#xd2b8;&#xb85c;&#xc791;&#xc1&#xc90;&#xc2a4;&#xd2b8;(cast)&#xac00;&#xbd80;&#xc871;&#xd560;&#xacbd;&#xc6b0;&#xc2e4;&#xd589;&#xb418;&#xc54a;&#xc744;&#xac83;&#xc785;&#xb2c8;&#xb2e4;.

**-r** &#xb85c;(raw) &#xaa8;&#xb4dc;&#xc785;&#xb2c8;&#xb2e4;. gid&#xd544;&#xb4dc;&#xb97c;&#xc0ad;&#xc81c;&#xd558&#xc5f4;&#xba85;&#xc6d;&#xc744;&#xc81c;&#xc678;&#xd558;&#xc9c0;&#xc54a;&#xc2b5;&#xb2c8;&#xb2e4;.

**-m filename** &#xc2dd;&#xbcc4;&#xc790;&#xb97c; 10&#xbb38;&#xc790;&#xba85;&#xc6d;&#xc73c;&#xb85c;&#xb2e4;&#xb9e4;&#xd551;(remap)&#xd569;&#xb2c8;&#xb2e4;. &#xd574;&#xb2f9;&#xd30c;&#xc77c;&#xc758;&#xb0b4;&#xc6a9&#xacf5;&#xbc31;&#xc73c;&#xb85c;&#xad6c;&#xbd84;&#xb41c;&#xb450;&#xc2ec;&#xbcf5;&#xb85c;&#xc774;&#xb8e&#xbcf5;&#xc218;&#xc758;&#xd589;&#xc73c;&#xb85c;.,&#xd589;&#xb9e8;&#xc55e;&#xacfc;&#xb9e8;&#xb4a4;&#xc5&#xacf5;&#xbc31;&#xc774;&#xc5c6;&#xc5b4;&#xc57c;&#xd569;&#xb2c8;&#xb2e4;. VERYLONGSYMBOL SHORT-ONE ANOTHERVERYLONGSYMBOL SHORTER &#xb4f1;&#xacfc;&#xac19;&#xc740;&#xc608;&#xac00;&#xc788;&#xc791;

#### 4.9 &#xc778;&#xb371;&#xc2a4;&#xbe4c;&#xb4dc;&#xc791;&#xc5c5;

&#xc778;&#xb371;&#xc2a4;&#xb355;&#xbd84;&#xc5d0;&#xacf5;&#xac04;&#xb370;&#xc774;&#xd130;&#xbca0;&#xc774;&#xb300;&#xc6a9;&#xb7c9;&#xb370;&#xc774;&#xd130;&#xc14b;&#xc744;&#xc0ac;&#xc6a9;&#xd560;&#xc218;&#xc788;&#xc778;&#xb371;&#xc2a4;&#xc791;&#xc5c5;&#xc744;&#xd558;&#xc9c0;&#xc54a;&#xc73c;&#xba74;. &#xc5b4;&#xb5a4;

&#xd53c;&#xcc98;&#xb97c;&#xac80;&#xc0c9;&#xd558;&#xb4e0;&#xb370;&#xc774;&#xd130;&#xbca0;&#xc774;&#xc2a4;&#xc548;&#xc758;&#xbaa8;&#xb4e0;&#xb808;&#xcf54;&#xb4dc;&#xb97c;&#xc21c;&#xcc28;&#xc2a4;&#xce94;"&#xd574;&#xd560;&#xac83;&#xc785;&#xb2c8;&#xb2e4;. &#xc778;&#xb371;&#xc2a4;&#xc791;&#xc5c5;&#xc740;&#xb370;&#xc774;&#xd2b9;&#xc815;&#xb808;&#xcf54;&#xb4dc;&#xb97c;&#xcc3e;&#xae30;&#xc704;&#xd574;&#xbe60;&#xb974;&#xac8c;&#xd6d1;&#xc5b4;&#xac08;&#xc218;&#xc788;&#xb294;&#xac80;&#xc0c9;&#xd2b8;&#xb9ac;&#xb85c;&#xc870;&#xc9c1;&#xac80;&#xc0c9;&#xc18d;&#xb3c4;&#xb97c;&#xd5a5;&#xc0c1;&#xc2dc;&#xd0b5;&#xb2c8;&#xb2e4;. PostgreSQL&#xb294;&#xae30;&#xbcf8;&#xc801;&#xc73c;&#xb85c; B-Tree, R-Tree, GiST &#xc138;&#xc885;&#xb958;&#xc758;&#xc778;&#xb371;&#xc9c0;&#xc6d0;&#xd569;&#xb2c8;&#xb2e4;.

The B-tree index method commonly used for attribute data is not very useful for spatial data, since it only supports storing and querying data in a single dimension. Data such as geometry (which has 2 or more dimensions) requires an index method that supports range query across all the data dimensions. One of the key advantages of PostgreSQL for spatial data handling is that it offers several kinds of index methods which work well for multi-dimensional data: GiST, BRIN and SP-GiST indexes.

- GiST (Generalized Search Tree)** &#xc778;&#xb371;&#xc2a4;&#xb294;&#xb370;&#xc774;&#xd130;&#xb97c; "&#xd55c;&#xcabd;&#xc5d0;&#xc788;&#xb294;&#xcac83;" "&#xacb9;&#xce58;&#xb294;&#xcac83;" "&#xb0b4;&#xbd80;&#xc5d0;&#xc788;&#xb294;&#xcac83;"&#xc73c;&#xb85c;&#xbd84;&#xd574;&#xd558;&#xba70; GIS &#xb370;&#xc774;&#xd130;&#xb97c;&#xd3ec;&#xd568;&#xd55c;&#xad11;&#xbc94;&#xc704;&#xd55c;&#xb370;&#xc774;&#xd130;&#xc720;&#xd615;&#xc5d0;&#xc4f0;&#xc77c;&#xc218;&#xc788;&#xc2b5;&#xb2c8;&#xb2e4;. PostGIS&#xb294; GiST&#xb97c;&#xc368;&#xc11c; GIS &#xb370;&#xc774;&#xd130;&#xc5d0;&#xc778;&#xb371;&#xc2a4;&#xc791;&#xc5c5;&#xc744;&#xd55c;&#xb2e4;&#xc774;&#xd574;&#xb2f9;&#xb370;&#xc774;&#xd130;&#xc5d0;&#xb2e4;&#xc2dc;&#xc791;&#xc5c5;&#xb41c; R-Tree &#xc778;&#xc774;&#xc6a9;&#xd569;&#xb2c8;&#xb2e4;.
- BRIN (Block Range Index)** indexes operate by summarizing the spatial extent of ranges of table records. Search is done via a scan of the ranges. BRIN is only appropriate for use for some kinds of data (spatially sorted, with infrequent or no update). But it provides much faster index create time, and much smaller index size.
- SP-GiST (Space-Partitioned Generalized Search Tree)** is a generic index method that supports partitioned search trees such as quad-trees, k-d trees, and radix trees (tries).

Spatial indexes store only the bounding box of geometries. Spatial queries use the index as a **primary filter** to quickly determine a set of geometries potentially matching the query condition. Most spatial queries require a **secondary filter** that uses a spatial predicate function to test a more specific spatial condition. For more information on queying with spatial predicates see Section 5.2.

See also the [PostGIS Workshop section on spatial indexes](#), and the [PostgreSQL manual](#).

#### 4.9.1 GiST &#xc778;&#xb371;&#xc2a4;

GiST&#xb294;"&#xc77c;&#xbc18;&#xd654;&#xb41c;&#xac80;&#xc0c9;&#xd2b8;&#xb9ac;"&#xc758;&#xc904;&#xc784;&#xb97c;&#xc778;&#xb371;&#xc2a4;&#xc791;&#xc5c5;&#xc758;&#xd3ec;&#xad04;&#xc801;&#xc778;&#xd615;&#xd0dc;&#xc785;&#xc774;&#xb371;&#xc2a4;&#xc791;&#xc5c5;&#xc678;&#xc5d0;&#xb3c4;,&#xc77c;&#xbc18; B-Tree &#xc778;&#xb371;&#xc791;&#xc5c5;&#xc73c;&#xb85c;&#xb294;&#xc4f8;&#xc218;&#xc5c6;&#xb294;&#xc628;&#xac16;&#xc885;&#xb958;&#xb85c;&#xc815;&#xaddd;&#xb370;&#xc774;&#xd130;&#xad6c;&#xc870;(&#xc815;&#xc218;&#xbc30;&#xc5f4;,&#xbd84;&#xb370;&#xc774;&#xd130;&#xb4f1;&#xb4f1;)&#xc5d0;&#xb300;&#xd55c;&#xac80;&#xc0c9;&#xc18d;&#xb3c4;&#xb97c;&#xd5a5;&#xc0c1;&#xc2dc;&#xd0a4;&#xb294;&#xb370; GiST&#xb97c;&#xc774;&#xc6a9;&#xd569;&#xb2c8;&#xb2e4;.

GIS &#xb370;&#xc774;&#xd130;&#xd14c;&#xc774;&#xbe14;&#xc774;&#xc218;&#xcc9c;&#xd589;&#xc744;&#xb118;&#xac80;&#xb418;&#xba74;,&#xb370;&#xc774;&#xd130;&#xacf5;&#xac04;&#xac80;&#xc0c9;&#xc758;&#xc18d;&#xb3c4;&#xb97c;&#xd5a5;&#xc0c1;&#xc2dc;&#xd0a4;&#xae30;&#xc704;&#xd574;&#xc778;&#xb371;&#xc2a4;&#xb97c;&#xbe4c;&#xb4dc;&#xc2f6;&#xac8c;&#xb420;&#xcac83;&#xc785;&#xb2c8;&#xb2e4;(&#xc0ac;&#xc6a9;&#xc790;&#xc758;&#xbaa8;&#xb4e0;&#xac80;&#xc0c9;&#xc774;&#xc18d;&#xc131;&#xc5d0;&#xae30;&#xbc18;&#xd558;&#xb294;&#xacbd;&#xc6b0;&#xac00;&#xc544;&#xb2c8;&#xb77c;&#xba74;&#xb9d0;&#xc785;&#xb2c8;&#xb2e4;.&#xadf8;&#xb7f0;&#xacbd;&#xc6b0;,&#xc18d;&#xd544;&#xb4dc;&#xc5d0;&#xb300;&#xd574;&#xc77c;&#xbc18;&#xc778;&#xb371;&#xc2a4;&#xb97c;&#xbe4c;&#xb4dc;&#xb429;&#xb2c8;&#xb2e4;).

"&#xb3c4;&#xd615;"&#xc5f4;&#xc5d0;&#xb300;&#xd574; GiST &#xc778;&#xb371;&#xc2a4;&#xb97c;&#xbe4c;&#xb4dc;&#xb370;&#xd544;&#xc694;&#xd55c;&#xbb38;&#xbc95;&#xc740;&#xb2e4;&#xc74c;&#xacfc;&#xc19;&#xc2b5;&#xb2c8;&#xc774;

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

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

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

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

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

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

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

#### 4.9.2 GiST

BRIN stands for "Block Range Index". It is a general-purpose index method introduced in PostgreSQL 9.5. BRIN is a *lossy* index method, meaning that a secondary check is required to confirm that a record matches a given search condition (which is the case for all provided spatial indexes). It provides much faster index creation and much smaller index size, with reasonable read performance. Its primary purpose is to support indexing very large tables on columns which have a correlation with their physical location within the table. In addition to spatial indexing, BRIN can speed up searches on various kinds of attribute data structures (integer, arrays etc). For more information see the [PostgreSQL manual](#).

GIS

A BRIN index stores the bounding box enclosing all the geometries contained in the rows in a contiguous set of table blocks, called a *block range*. When executing a query using the index the block ranges are scanned to find the ones that intersect the query extent. This is efficient only if the data is physically ordered so that the bounding boxes for block ranges have minimal overlap (and ideally are mutually exclusive). The resulting index is very small in size, but is typically less performant for read than a GiST index over the same data.

Building a BRIN index is much less CPU-intensive than building a GiST index. It's common to find that a BRIN index is ten times faster to build than a GiST index over the same data. And because a BRIN index stores only one bounding box for each range of table blocks, it's common to use up to a thousand times less disk space than a GiST index.

You can choose the number of blocks to summarize in a range. If you decrease this number, the index will be bigger but will probably provide better performance.

For BRIN to be effective, the table data should be stored in a physical order which minimizes the amount of block extent overlap. It may be that the data is already sorted appropriately (for instance, if it is loaded from another dataset that is already sorted in spatial order). Otherwise, this can be accomplished by sorting the data by a one-dimensional spatial key. One way to do this is to create a new table sorted by the geometry values (which in recent PostGIS versions uses an efficient Hilbert curve ordering):

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

Alternatively, data can be sorted in-place by using a GeoHash as a (temporary) index, and clustering on that index:

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

```
"&#xb3c4;&#xd615;" &#xc5f4;&#xc5d0; &#xb300;&#xd574; GiST &#xc778;&#xb371;&#xc2a4;&#xb97c; &#xbe4c;&#xb4dc;&#xc778;&#xb370; &#xd544;&#xc694;&#xd55c; &#xbb38;&#xbc95;&#xc740; &#xb2e4;&#xc74c;&#xacfc; &#xac19;&#xc2b5;&#xb2c8;&#xc778;&#xb371;&#xc2a4;&#xb97c; &#xbe4c;&#xb4dc;&#xd560;&#xac83;&#xc785;&#xb2c8;&#xb2e4;. &#xd574;&#xb2f9; &#xb3c4;&#xd615; &#xc720;&#xd615;&#xc5d0; PostGIS 2.0 &#xc774;&#xc0c1; &#xbc84;&#xc804;&#xc774; &#xc9c0;&#xc6d0;&#xd558;&#xb294; n&#xcc28;&#xc6d0; &#xc778;&#xb371;&#xc5bb;&#xc73c;&#xb824;&#xba74;, &#xb2e4;&#xc74c; &#xbb38;&#xbc95;&#xc73c;&#xb85c; &#xc0dd;&#xc131;&#xd560; &#xc218; &#xc788;&#xc2b5;&#xb2c8;&#xb2e4;:
```

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

```
&#xc774; &#xbb38;&#xbc95;&#xc740; &#xd56d;&#xc0c1; 2D &#xc778;&#xb371;&#xc2a4;&#xb97c; &#xbe4c;&#xb4dc;&#xd560;&#xac83;&#xc785;&#xb2c8;&#xb2e4;. &#xd574;&#xb2f9; &#xb3c4;&#xd615; &#xc720;&#xd615;&#xc5d0; PostGIS 2.0 &#xc774;&#xc0c1; &#xbc84;&#xc804;&#xc774; &#xc9c0;&#xc6d0;&#xd558;&#xb294; n&#xcc28;&#xc6d0; &#xc778;&#xb371;&#xc5bb;&#xc73c;&#xb824;&#xba74;, &#xb2e4;&#xc74c; &#xbb38;&#xbc95;&#xc73c;&#xb85c; &#xc0dd;&#xc131;&#xd560; &#xc218; &#xc788;&#xc2b5;&#xb2c8;&#xb2e4;:
```

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

You can also get a 4D-dimensional index using the 4D operator class:

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

The above commands use the default number of blocks in a range, which is 128. To specify the number of blocks to summarise in a range, use this syntax

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

Keep in mind that a BRIN index only stores one index entry for a large number of rows. If your table stores geometries with a mixed number of dimensions, it's likely that the resulting index will have poor performance. You can avoid this performance penalty by choosing the operator class with the least number of dimensions of the stored geometries

```
"&#xb3c4;&#xd615;" &#xc5f4;&#xc5d0; &#xb300;&#xd574; GiST &#xc778;&#xb371;&#xc2a4;&#xb97c; &#xbe4c;&#xb4dc;&#xc778;&#xb370; &#xd544;&#xc694;&#xd55c; &#xbb38;&#xbc95;&#xc740; &#xb2e4;&#xc74c;&#xacfc; &#xac19;&#xc2b5;&#xb2c8;&#xc778;&#xb371;&#xc2a4;&#xb97c; &#xbe4c;&#xb4dc;&#xd560;&#xac83;&#xc785;&#xb2c8;&#xb2e4;. &#xd574;&#xb2f9; &#xb3c4;&#xd615; &#xc720;&#xd615;&#xc5d0; PostGIS 2.0 &#xc774;&#xc0c1; &#xbc84;&#xc804;&#xc774; &#xc9c0;&#xc6d0;&#xd558;&#xb294; n&#xcc28;&#xc6d0; &#xc778;&#xb371;&#xc5bb;&#xc73c;&#xb824;&#xba74;, &#xb2e4;&#xc74c; &#xbb38;&#xbc95;&#xc73c;&#xb85c; &#xc0dd;&#xc131;&#xd560; &#xc218; &#xc788;&#xc2b5;&#xb2c8;&#xb2e4;:
```

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

```
&#xc774; &#xbb38;&#xbc95;&#xc740; &#xd56d;&#xc0c1; 2D &#xc778;&#xb371;&#xc2a4;&#xb97c; &#xbe4c;&#xb4dc;&#xd560;&#xac83;&#xc785;&#xb2c8;&#xb2e4;. &#xd574;&#xb2f9; &#xb3c4;&#xd615; &#xc720;&#xd615;&#xc5d0; PostGIS 2.0 &#xc774;&#xc0c1; &#xbc84;&#xc804;&#xc774; &#xc9c0;&#xc6d0;&#xd558;&#xb294; n&#xcc28;&#xc6d0; &#xc778;&#xb371;&#xc5bb;&#xc73c;&#xb824;&#xba74;, &#xb2e4;&#xc74c; &#xbb38;&#xbc95;&#xc73c;&#xb85c; &#xc0dd;&#xc131;&#xd560; &#xc218; &#xc788;&#xc2b5;&#xb2c8;&#xb2e4;:
```

Currently, only "inclusion support" is provided, meaning that just the &&, ~ and @ operators can be used for the 2D cases (for both geometry and geography), and just the &&& operator for 3D geometries. There is currently no support for kNN searches.

An important difference between BRIN and other index types is that the database does not maintain the index dynamically. Changes to spatial data in the table are simply appended to the end of the index. This will cause index search performance to degrade over time. The index can be updated by performing a VACUUM, or by using a special function `brin_summarize_new_values`. For this reason BRIN may be most appropriate for use with data that is read-only, or only rarely changing. For more information refer to the [manual](#).

To summarize using BRIN for spatial data:

- Index build time is very fast, and index size is very small.
- Index query time is slower than GiST, but can still be very acceptable.
- Requires table data to be sorted in a spatial ordering.
- Requires manual index maintenance.
- Most appropriate for very large tables, with low or no overlap (e.g. points), which are static or change infrequently.
- More effective for queries which return relatively large numbers of data records.

### 4.9.3 GiST

SP-GiST stands for "Space-Partitioned Generalized Search Tree" and is a generic form of indexing for multi-dimensional data types that supports partitioned search trees, such as quad-trees, k-d trees, and radix trees (tries). The common feature of these data structures is that they repeatedly divide the search space into partitions that need not be of equal size. In addition to spatial indexing, SP-GiST is used to speed up searches on many kinds of data, such as phone routing, ip routing, substring search, etc. For more information see the [PostgreSQL manual](#).

As it is the case for GiST indexes, SP-GiST indexes are lossy, in the sense that they store the bounding box enclosing spatial objects. SP-GiST indexes can be considered as an alternative to GiST indexes.

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

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

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

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

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

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

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

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

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

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

There is no support for kNN searches at the moment.

### 4.9.4

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

```

&#xbabb; &#xd558;&#xae30; &#xb54c;&#xbb38;&#xc5d0;, &#xc885;&#xc885; &#xacf5;&#xac04; &#xc778;&#xb371;&#xc2a4;&#
&#xd65c;&#xc6a9;&#xd574;&#xc57c; &#xd560; &#xac80;&#xc0c9;&#xc774; &#xb300;&#xc2e0; &#xae30;&#xbcf8;&#xac12;&#
&#xc804;&#xccb4; &#xb370;&#xc774;&#xd130;&#xc758; &#xc21c;&#xcc28; &#xc2a4;&#xce94;&#xc744; &#xc774;&#xc6a9;&#
&#xb54c;&#xac00; &#xc788;&#xc2b5;&#xb2c8;&#xb2e4;.

```

```

&#xc0ac;&#xc6a9;&#xc790;&#xc758; &#xacf5;&#xac04; &#xc778;&#xb371;&#xc2a4;&#xac00; (&#xb610;&#xb294; &#xc0ac;&#
&#xc18d;&#xc131; &#xc778;&#xb371;&#xc2a4;&#xac00;) &#xd65c;&#xc6a9;&#xb418;&#xc9c0; &#xc54a;&#xace0; &#xc788;&#
&#xc0ac;&#xc2e4;&#xc744; &#xc54c;&#xac8c; &#xb418;&#xc5c8;&#xb2e4;&#xba74;, &#xba87; &#xac00;&#xc9c0; &#xd574;&#
&#xbc29;&#xbc95;&#xc774; &#xc788;&#xc2b5;&#xb2c8;&#xb2e4;.

```

- Examine the query plan and check your query actually computes the thing you need. An erroneous JOIN, either forgotten or to the wrong table, can unexpectedly retrieve table records multiple times. To get the query plan, execute with `EXPLAIN` in front of the query.
- Make sure statistics are gathered about the number and distributions of values in a table, to provide the query planner with better information to make decisions around index usage. `VACUUM ANALYZE` will compute both.

You should regularly vacuum your databases anyways. Many PostgreSQL DBAs run `VACUUM` as an off-peak cron job on a regular basis.

- &#xb450; &#xbc88;&#xc9f8;, &#xbe48;&#xacf5;&#xac04; &#xbd84;&#xc11d;&#xc73c;&#xb85c; &#xd574;&#xacb0;&#xc774; &#xc548; &#xb420; &#xacbd;&#xc6b0; **SET ENABLE\_SEQSCAN=OFF** &#xba85;&#xb839;&#xc5b4;&#xb97c; &#xd1b5;&#xc &#xac15;&#xc81c;&#xb85c; &#xcffc;&#xb9ac; &#xc124;&#xacc4;&#xc790;&#xac00; &#xc778;&#xb371;&#xc2a4; &#xc815;&# &#xc774;&#xc6a9;&#xd558;&#xb3c4;&#xb85d; &#xd560; &#xc218; &#xc788;&#xc2b5;&#xb2c8;&#xb2e4;. &#xc774; &#xba85;&#xb839;&#xc5b4;&#xb294; &#xacf5;&#xac04; &#xc778;&#xb371;&#xc2a4; &#xcffc;&#xb9ac;&#xc77c; &#xacbd;&# &#xd55c;&#xd574; &#xb4dc;&#xbb3c;&#xac8c; &#xc774;&#xc6a9;&#xd574;&#xc57c; &#xd569;&#xb2c8;&#xb2e4;. &#xc77c; &#xcffc;&#xb9ac; &#xc124;&#xacc4;&#xc790;&#xb294; &#xc5b8;&#xc81c; &#xc77c;&#xbc18; B-Tree &#xc778;&#xb371;&#xc &#xd65c;&#xc6a9;&#xd574;&#xc57c; &#xd558;&#xb294;&#xc9c0; &#xc0ac;&#xc6a9;&#xc790;&#xbcf4;&#xb2e4; &#xb354; &#xc798; &#xc54c;&#xace0; &#xc788;&#xc2b5;&#xb2c8;&#xb2e4;. &#xc0ac;&#xc6a9;&#xc790; &#xcffc;&#xb9ac; &#xc2e4;& &#xd6c4;, &#xb2e4;&#xb978; &#xcffc;&#xb9ac;&#xac00; &#xd3c9;&#xc18c;&#xcc98;&#xb7fc; &#xcffc;&#xb9ac; &#xc124;&# &#xd65c;&#xc6a9;&#xd558;&#xb3c4;&#xb85d; ENABLE\_SEQSCAN &#xc744; &#xb2e4;&#xc2dc; &#xcfc24;&#xc9c0; &#xace0; &#xd569;&#xb2c8;&#xb2e4;.
- &#xcffc;&#xb9ac; &#xc124;&#xacc4;&#xc790;&#xac00; &#xc21c;&#xcc28; &#xbc0f; &#xc778;&#xb371;&#xc2a4; &#xc2a4;&# &#xacbd;&#xc911;(cost)&#xc744; &#xc798;&#xbabb; &#xd310;&#xb2e8;&#xd558;&#xace0; &#xc788;&#xb2e4;&#xba74;, postgresql.conf &#xd30c;&#xc77c;&#xc758; random\_page\_cost&#xc758; &#xac12;&#xc744; &#xc904;&#xc5ec;&#xbcf4;&#xac7 "SET random\_page\_cost=#" &#xb85c; &#xc368;&#xbcf4;&#xc2ed;&#xc2dc;&#xc624;. &#xd574;&#xb2f9; &#xd30c;&#xb77c;&# &#xae30;&#xbcf8;&#xac12;&#xc740; 4&#xc774;&#xc9c0;&#xb9cc;, 1&#xb610;&#xb294; 2&#xb85c; &#xc124;&#xc815;&#xd5 &#xac12;&#xc744; &#xac10;&#xc18c;&#xc2dc;&#xd0ac;&#xc218;&#xb85d; &#xc810;&#xc810; &#xb354; &#xcffc;&#xb9ac; &#xc124;&#xacc4;&#xc790;&#xac00; &#xc778;&#xb371;&#xc2a4; &#xc2a4;&#xce94;&#xc744; &#xd65c;&#xc6a9;&#xd558;& &#xb420; &#xac83;&#xc785;&#xb2c8;&#xb2e4;.
- If **SET ENABLE\_SEQSCAN TO OFF**; does not help your query, the query may be using a SQL construct that the Postgres planner is not yet able to optimize. It may be possible to rewrite the query in a way that the planner is able to handle. For example, a subquery with an inline SELECT may not produce an efficient plan, but could possibly be rewritten using a LATERAL JOIN.

For more information see the Postgres manual section on [Query Planning](#).

## Chapter 5

# Spatial Queries

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

### 5.1 Determining Spatial Relationships

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

#### 5.1.1 Dimensionally Extended 9-Intersection Model

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

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

##### Boundary

The boundary of a geometry is the set of geometries of the next lower dimension. For POINTs, which have a dimension of 0, the boundary is the empty set. The boundary of a LINESTRING is the two endpoints. For POLYGONS, the boundary is the linework of the exterior and interior rings.

##### Interior

The interior of a geometry are those points of a geometry that are not in the boundary. For POINTs, the interior is the point itself. The interior of a LINESTRING is the set of points between the endpoints. For POLYGONS, the interior is the areal surface inside the polygon.

##### Exterior

The exterior of a geometry is the rest of the space in which the geometry is embedded; in other words, all points not in the interior or on the boundary of the geometry. It is a 2-dimensional non-closed surface.

The [Dimensionally Extended 9-Intersection Model](#) (DE-9IM) describes the spatial relationship between two geometries by specifying the dimensions of the 9 intersections between the above sets for each geometry. The intersection dimensions can be formally represented in a 3x3 **intersection matrix**.

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

- 0 => point

- 1 => line
- 2 => area
- F => empty set

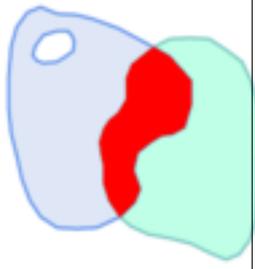
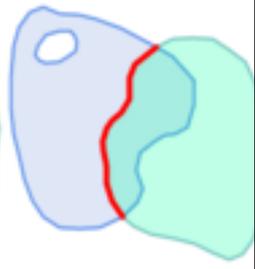
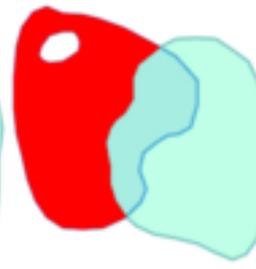
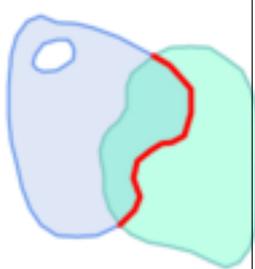
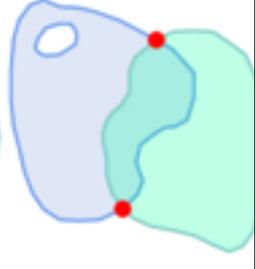
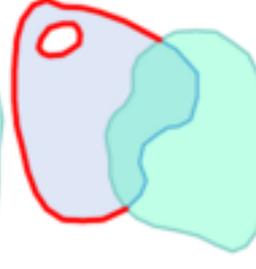
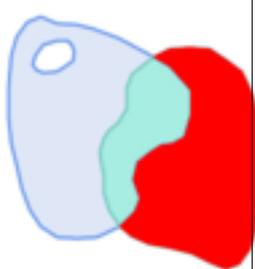
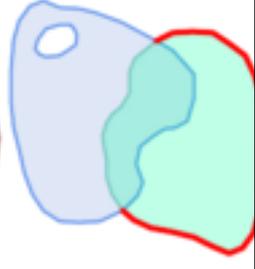
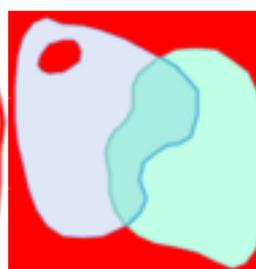
Using this notation, the intersection matrix for two geometries  $a$  and  $b$  is:

|                 | <b>Interior</b>         | <b>Boundary</b>         | <b>Exterior</b>         |
|-----------------|-------------------------|-------------------------|-------------------------|
| <b>Interior</b> | $dim( I(a) \cap I(b) )$ | $dim( I(a) \cap B(b) )$ | $dim( I(a) \cap E(b) )$ |
| <b>Boundary</b> | $dim( B(a) \cap I(b) )$ | $dim( B(a) \cap B(b) )$ | $dim( B(a) \cap E(b) )$ |
| <b>Exterior</b> | $dim( E(a) \cap I(b) )$ | $dim( E(a) \cap B(b) )$ | $dim( E(a) \cap E(b) )$ |

Visually, for two overlapping polygonal geometries, this looks like:





|          | Interior  | Boundary   | Exterior  |
|----------|---|--|---|
| Interior |  $dim(I(a) \cap I(b)) = 2$   |  $dim(I(a) \cap B(b)) = 1$   |  $dim(I(a) \cap E(b)) = 2$   |
| Boundary |  $dim(B(a) \cap I(b)) = 1$  |  $dim(B(a) \cap B(b)) = 0$  |  $dim(B(a) \cap E(b)) = 1$  |
| Exterior |  $dim(E(a) \cap I(b)) = 2$ |  $dim(E(a) \cap B(b)) = 1$ |  $dim(E(a) \cap E(b)) = 2$ |

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

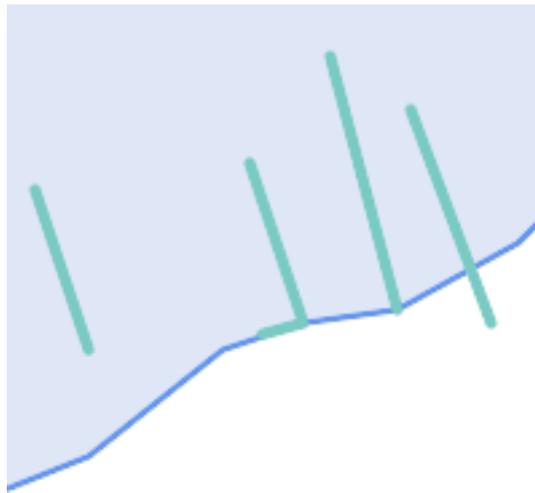
For more information, refer to:

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

### 5.1.2 Named Spatial Relationships

To make it easy to determine common spatial relationships, the OGC SFS defines a set of *named spatial relationship predicates*. PostGIS provides these as the functions [ST\\_Contains](#), [ST\\_Crosses](#), [ST\\_Disjoint](#), [ST\\_Equals](#), [ST\\_Intersects](#), [ST\\_Overlaps](#), [ST\\_Touches](#), [ST\\_Within](#). It also defines the non-standard relationship predicates [ST\\_Covers](#), [ST\\_CoveredBy](#), and [ST\\_ContainsProperly](#).





A second example is locating wharves that intersect a lake's boundary on a line and where one end of the wharf is up on shore. In other words, where a wharf is within but not completely contained by a lake, intersects the boundary of a lake on a line, and where exactly one of the wharf's endpoints is within or on the boundary of the lake. It is possible to use a combination of spatial predicates to find the required features:

- `ST_Contains(lake, wharf) = TRUE`
- `ST_ContainsProperly(lake, wharf) = FALSE`
- `ST_GeometryType(ST_Intersection(wharf, lake)) = 'LINESTRING'`
- `ST_NumGeometries(ST_Multi(ST_Intersection(ST_Boundary(wharf), ST_Boundary(lake)))) = 1`  
... but needless to say, this is quite complicated.

These requirements can be met by computing the full DE-9IM intersection matrix. PostGIS provides the `ST_Relate` function to do this:

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

To test a particular spatial relationship, an **intersection matrix pattern** is used. This is the matrix representation augmented with the additional symbols {T, \*}:

- T => intersection dimension is non-empty; i.e. is in {0, 1, 2}
- \* => don't care

Using intersection matrix patterns, specific spatial relationships can be evaluated in a more succinct way. The `ST_Relate` and the `ST_RelateMatch` functions can be used to test intersection matrix patterns. For the first example above, the intersection matrix pattern specifying two lines intersecting in a line is `'1*1***1**'`:

```
-- Find road segments that intersect in a line
SELECT a.id
FROM roads a, roads b
WHERE a.id != b.id
      AND a.geom && b.geom
      AND ST_Relate(a.geom, b.geom, '1*1***1**');
```

For the second example, the intersection matrix pattern specifying a line partly inside and partly outside a polygon is '102101FF2':

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

## 5.2 Using Spatial Indexes

When constructing queries using spatial conditions, for best performance it is important to ensure that a spatial index is used, if one exists (see Section 4.9). To do this, a spatial operator or index-aware function must be used in a WHERE or ON clause of the query.

Spatial operators include the bounding box operators (of which the most commonly used is &&; see Section 8.10.1 for the full list) and the distance operators used in nearest-neighbor queries (the most common being <->; see Section 8.10.2 for the full list.)

Index-aware functions automatically add a bounding box operator to the spatial condition. Index-aware functions include the named spatial relationship predicates [ST\\_Contains](#), [ST\\_ContainsProperly](#), [ST\\_CoveredBy](#), [ST\\_Covers](#), [ST\\_Crosses](#), [ST\\_Intersects](#), [ST\\_Overlaps](#), [ST\\_Touches](#), [ST\\_Within](#), [ST\\_Within](#), and [ST\\_3DIntersects](#), and the distance predicates [ST\\_DWithin](#), [ST\\_DFullyWithin](#), [ST\\_3DDFullyWithin](#), and [ST\\_3DDWithin](#).)

Functions such as [ST\\_Distance](#) do *not* use indexes to optimize their operation. For example, the following query would be quite slow on a large table:

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

This query selects all the geometries in `geom_table` which are within 100 units of the point (100000, 200000). It will be slow because it is calculating the distance between each point in the table and the specified point, ie. one `ST_Distance()` calculation is computed for **every** row in the table.

The number of rows processed can be reduced substantially by using the index-aware function [ST\\_DWithin](#):

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

This query selects the same geometries, but it does it in a more efficient way. This is enabled by `ST_DWithin()` using the && operator internally on an expanded bounding box of the query geometry. If there is a spatial index on `geom`, the query planner will recognize that it can use the index to reduce the number of rows scanned before calculating the distance. The spatial index allows retrieving only records with geometries whose bounding boxes overlap the expanded extent and hence which *might* be within the required distance. The actual distance is then computed to confirm whether to include the record in the result set.

For more information and examples see the [PostGIS Workshop](#).

## 5.3 Examples of Spatial SQL

The examples in this section make use of a table of linear roads, and a table of polygonal municipality boundaries. The definition of the `bc_roads` table is:

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

The definition of the `bc_municipality` table is:

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

1. *What is the total length of all roads, expressed in kilometers?*

You can answer this question with a very simple piece of SQL:

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

km_roads
-----
70842.1243039643
```

2. *How large is the city of Prince George, in hectares?*

This query combines an attribute condition (on the municipality name) with a spatial calculation (of the polygon area):

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

hectares
-----
32657.9103824927
```

3. *What is the largest municipality in the province, by area?*

This query uses a spatial measurement as an ordering value. There are several ways of approaching this problem, but the most efficient is below:

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

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

Note that in order to answer this query we have to calculate the area of every polygon. If we were doing this a lot it would make sense to add an area column to the table that could be indexed for performance. By ordering the results in a descending direction, and then using the PostgreSQL "LIMIT" command we can easily select just the largest value without using an aggregate function like `MAX()`.

4. *What is the length of roads fully contained within each municipality?*

This is an example of a "spatial join", which brings together data from two tables (with a join) using a spatial interaction ("contained") as the join condition (rather than the usual relational approach of joining on a common key):

```
SELECT
  m.name,
  sum(ST_Length(r.geom))/1000 as roads_km
FROM bc_roads AS r
JOIN bc_municipality AS m
```

```

    ON ST_Contains(m.geom, r.geom)
GROUP BY m.name
ORDER BY roads_km;

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

```

This query takes a while, because every road in the table is summarized into the final result (about 250K roads for the example table). For smaller datasets (several thousand records on several hundred) the response can be very fast.

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

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

```

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

```

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

```

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

kilometers
-----
4.89151904172838

```

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

```

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

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

```

## Chapter 6

# 6. TOAST

## 6.1 TOAST

### 6.1.1 TOAST

PostgreSQL (8.0 and later) uses TOAST (TOAST) to store large objects (LOBs) that exceed the maximum size of a regular PostgreSQL record (1 MB). TOAST is implemented as a separate table (pg\_toast) and is used to store large objects (LOBs) that exceed the maximum size of a regular PostgreSQL record (1 MB). TOAST is implemented as a separate table (pg\_toast) and is used to store large objects (LOBs) that exceed the maximum size of a regular PostgreSQL record (1 MB). TOAST is implemented as a separate table (pg\_toast) and is used to store large objects (LOBs) that exceed the maximum size of a regular PostgreSQL record (1 MB).

TOAST is implemented as a separate table (pg\_toast) and is used to store large objects (LOBs) that exceed the maximum size of a regular PostgreSQL record (1 MB). TOAST is implemented as a separate table (pg\_toast) and is used to store large objects (LOBs) that exceed the maximum size of a regular PostgreSQL record (1 MB). TOAST is implemented as a separate table (pg\_toast) and is used to store large objects (LOBs) that exceed the maximum size of a regular PostgreSQL record (1 MB). TOAST is implemented as a separate table (pg\_toast) and is used to store large objects (LOBs) that exceed the maximum size of a regular PostgreSQL record (1 MB).

TOAST is implemented as a separate table (pg\_toast) and is used to store large objects (LOBs) that exceed the maximum size of a regular PostgreSQL record (1 MB). TOAST is implemented as a separate table (pg\_toast) and is used to store large objects (LOBs) that exceed the maximum size of a regular PostgreSQL record (1 MB). TOAST is implemented as a separate table (pg\_toast) and is used to store large objects (LOBs) that exceed the maximum size of a regular PostgreSQL record (1 MB). TOAST is implemented as a separate table (pg\_toast) and is used to store large objects (LOBs) that exceed the maximum size of a regular PostgreSQL record (1 MB).

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

## 6.1.2

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

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

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

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

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

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

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

## 6.2

```

CREATE INDEX my_geom_index ON my_table USING GIST (geom);
ALTER TABLE my_table ALTER COLUMN geom SET NOT NULL;

```

PostgreSQL: CLUSTER my\_geom\_index ON my\_table;

PostGIS GiST

```

CREATE INDEX my_geom_index ON my_table USING GIST (geom);
ALTER TABLE my_table ALTER COLUMN geom SET NOT NULL;

```

```

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

```

```

HINT: "not null"

```

```

lgeom=# ALTER TABLE my_table ALTER COLUMN the_geom SET not null;
ALTER TABLE

```

```

ALTER TABLE my_table ADD CHECK (geometry IS NOT NULL);

```

## 6.3

```

CREATE INDEX my_geom_index ON my_table USING GIST (geom);
ALTER TABLE my_table ALTER COLUMN geom SET NOT NULL;

```

```
ST_Force2D()
  &#xd568;&#xc218;&#xb97c; &#xd638;&#xcd9c;&#xd558;&#xb294;&#xb370;, &#xb300;&#xc6a9;&#xb7c9;
  &#xb3c4;&#xd615;&#xc758; &#xacbd;&#xc6b0; &#xc774; &#xd568;&#xc218;&#xb294; &#xc2dc;&#xc2a4;&#xd15c;&#xc758;
  &#xc790;&#xc6d0;&#xc744; &#xc0c1;&#xb2f9;&#xd788; &#xc7a1;&#xc544;&#xba39;&#xc2b5;&#xb2c8;&#xb2e4;. &#xc774;&#xc790;&#xc6d0;
  &#xb0ad;&#xbe44;&#xb97c; &#xd53c;&#xd558;&#xb824;&#xba74;, &#xbbf8;&#xb9ac; &#xadf8; &#xcd94;&#xc790;&#xc6d0;&#xc744;
  &#xc644;&#xc804;&#xd788; &#xc0ad;&#xc81c;&#xd558;&#xb294; &#xd3b8;&#xc774; &#xd6a8;&#xc218; &#xc788;&#xc2b5;&#xb2c8;&#xb2e4;.
```

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

```
AddGeometryColumn()
  &#xd568;&#xc218;&#xb97c; &#xd1b5;&#xd574; &#xc0ac;&#xc6a9;&#xc790; &#xb3c4;&#xd615;
  &#xc5f4;&#xc744; &#xcd94;&#xac00;&#xd588;&#xc744; &#xacbd;&#xc6b0; &#xb3c4;&#xd615; &#xcc28;&#xc6d0;&#xc5d0;
  &#xc81c;&#xc7d;&#xc870;&#xac74;&#xc774; &#xac78;&#xb824; &#xc788;&#xb2e4;&#xb294; &#xc0ac;&#xc2e4;&#xc744;
  &#xc8fc;&#xc758;&#xd558;&#xc2ed;&#xc2dc;&#xc624;. &#xc774;&#xb97c; &#xc6b0;&#xd68c;&#xd558;&#xb824;&#xba74;
  &#xc81c;&#xc7d;&#xc870;&#xac74;&#xc744; &#xc0ad;&#xc81c;&#xd574;&#xc57c; &#xd569;&#xb2c8;&#xb2e4;. ge-
  ometry_columns &#xd14c;&#xc774;&#xbe14; &#xb0b4;&#xbd80; &#xd56d;&#xbaa9;&#xc744; &#xc5c5;&#xb370;&#xc774;&#xd14c;
  &#xb2e4;&#xc74c; &#xc81c;&#xc57d;&#xc870;&#xac74;&#xc744; &#xb2e4;&#xc2dc; &#xc0dd;&#xc131;&#xd558;&#xb294;
  &#xac78; &#xc78a;&#xc9c0; &#xb9c8;&#xc2ed;&#xc2dc;&#xc624;.
```

```
&#xb300;&#xc6a9;&#xb7c9; &#xd14c;&#xc774;&#xbe14;&#xc758; &#xacbd;&#xc6b0;, &#xc0ac;&#xc6a9;&#xc790;&#xc758;
  &#xae30;&#xbcf8; &#xd0a4; &#xb610;&#xb294; &#xb610;&#xb2e4;&#xb978; &#xc0ac;&#xc6a9;&#xac00;&#xb2a5;&#xd55c;
  &#xae30;&#xc900;&#xacfc; &#xd568;&#xaed8; WHERE &#xc808;&#xc744; &#xc774;&#xc6a9;&#xd574;&#xc11c; &#xd14c;&#xc774;
  &#xc77c;&#xbd80;&#xbd84;&#xb9cc; &#xc5c5;&#xb370;&#xc774;&#xd2b8;&#xd558;&#xb3c4;&#xb85d; &#xc81c;&#xd55c;&#xb2e4;
  &#xc74c; &#xc5c5;&#xb370;&#xc774;&#xd2b8;&#xb9c8;&#xb2e4; &#xac04;&#xb2e8;&#xd788; "VACUUM;" &#xc744;
  &#xc2e4;&#xd589;&#xd558;&#xb294; &#xbc29;&#xc2dd;&#xc73c;&#xb85c; &#xc774; &#xc5c5;&#xb370;&#xc774;&#xd2b8;&#xc791;
  &#xc740; &#xbd80;&#xbd84;&#xc73c;&#xb85c; &#xb098;&#xb204;&#xb294; &#xd3b8;&#xc774; &#xc88b;&#xc744;
  &#xc218; &#xc788;&#xc2b5;&#xb2c8;&#xb2e4;. &#xc774;&#xb807;&#xac8c; &#xd558;&#xba74; &#xd544;&#xc694;&#xd55c;
  &#xc784;&#xc2dc; &#xb514;&#xc2a4;&#xd06c; &#xacf5;&#xac04;&#xc774; &#xae09;&#xaca9;&#xd788; &#xc904;&#xc5b4;&#xb610;
  &#xd55c; &#xba87; &#xc885;&#xb958;&#xc758; &#xcc28;&#xc6d0;&#xc774; &#xc11e;&#xc778; &#xb3c4;&#xd615;&#xac00;
  &#xc9c0;&#xace0; &#xc788;&#xc744; &#xacbd;&#xc6b0;, "WHERE dimension(the_geom)>2" &#xb85c; &#xc5c5;&#xb370;
  &#xc81c;&#xd55c;&#xd558;&#xba74; &#xc774;&#xbbf8; 2D&#xc778; &#xb3c4;&#xd615;&#xc744; &#xb2e4;&#xc2dc;
  &#xc791;&#xc131;&#xd558;&#xb294; &#xc77c;&#xc744; &#xac74;&#xb108;&#xb6f8; &#xc218; &#xc788;&#xc2b5;&#xb2c8;&#xb2e4;.
```

## Chapter 7

# PostGIS &#xb3c4;&#xd615; &#xd65c;&#xc6a9;: &#xc751;&#xc6a9; &#xd504;&#xb85c;&#xadf8;&#x &#xbe4c;&#xb4dc;

### 7.1 MapServer &#xd65c;&#xc6a9;

&#xbbf8;&#xb124;&#xc18c;&#xd0c0; &#xb9f5;&#xc11c;&#xbc84;(Minnesota MapServer)&#xb294; OpenGIS &#xc6f9; &#xb9e4;&#xc11c;&#xbc84; &#xc0ac;&#xc591;&#xc11c;&#xb97c; &#xc900;&#xc218;&#xd558;&#xb294; &#xc778;&#xd130;&#xb137; &#xc6f9; &#xb9e4;&#xd551; &#xc11c;&#xbc84;&#xc785;&#xb2c8;&#xb2e4;.

- MapServer &#xd648;&#xd398;&#xc774;&#xc9c0;&#xb294; <http://mapserver.org> &#xc785;&#xb2c8;&#xb2e4;.
- <http://www.opengeospatial.org/standards/wms> &#xc5d0;&#xc11c; OpenGIS &#xc6f9; &#xb9f5; &#xc0ac;&#xc591;&#xc11c;&#xb137; &#xc3e;&#xc544;&#xbcf; &#xc218; &#xc788;&#xc2b5;&#xb2c8;&#xb2e4;.

#### 7.1.1 &#xae30;&#xbcf8; &#xd65c;&#xc6a9;

MapServer&#xc640; &#xd568;&#xaed8; PostGIS&#xb97c; &#xc0ac;&#xc6a9;&#xd558;&#xb824;&#xba74; MapServer &#xc124;&#xc54c;&#xc544;&#xc57c; &#xd558;&#xb294;&#xb370;., MapServer &#xc124;&#xc815;&#xc740; &#xc774; &#xb38;&#xc11c;&#xb9c94;&#xc704;&#xb97c; &#xb9c97;&#xc5b4;&#xb0a9;&#xb2c8;&#xb2e4;. &#xc774; &#xb2e8;&#xc6d0;&#xc5d0;&#xc11c;&#xb9c97;&#xc815; PostGIS &#xb38;&#xc81c;&#xc810; &#xbcf; &#xc124;&#xc815; &#xc0c1;&#xc138; &#xc815;&#xbcf4;&#xb2e4;&#xb8f0; &#xac83;&#xc785;&#xb2c8;&#xb2e4;.

MapServer&#xc640; &#xd568;&#xaed8; PostGIS&#xb97c; &#xc0ac;&#xc6a9;&#xd558;&#xb824;&#xba74; &#xb2e4;&#xc74c; &#xd504;&#xb85c;&#xadf8;&#xb7a8;&#xc774; &#xd544;&#xc694;&#xd569;&#xb2c8;&#xb2e4;:

- PostGIS 0.6 &#xc774;&#xc0c1; &#xbc84;&#xc804;
- MapServer 3.5 &#xc774;&#xc0c1; &#xbc84;&#xc804;

MapServer&#xb294; &#xb2e4;&#xb978; &#xc5b4;&#xb5a4; PostgreSQL &#xd074;&#xb77c;&#xc774;&#xc5b8;&#xd2b8;&#xc640; &#xb3d9;&#xc77c;&#xd55c; &#xb29;&#xc2dd;&#xc73c;&#xb85c; -- libpq &#xc778;&#xd130;&#xd398;&#xc774;&#xc2a4;&#xc774;&#xc6a9;&#xd574;&#xc11c; PostGIS/PostgreSQL &#xb370;&#xc774;&#xd130;&#xc5d0; &#xc811;&#xadfc;&#xd569;&#xc989; PostGIS &#xc11c;&#xbc84;&#xc5d0; &#xb124;&#xd2b8;&#xc6cc;&#xd06c; &#xc5f0;&#xacb0;&#xb41c; &#xc5b4;&#xc9ef4;&#xd4e8;&#xd130;&#xc5d0;&#xb77c;&#xb3c4; MapServer&#xb97c; &#xc124;&#xce58;&#xd560; &#xc218; &#xc788;&#xc774; PostGIS&#xb97c; &#xb370;&#xc774;&#xd130; &#xc18c;&#xc2a4;&#xb85c; &#xc774;&#xc6a9;&#xd560; &#xc218; &#xc788;&#xb73b;&#xc785;&#xb2c8;&#xb2e4;. &#xb450; &#xc2dc;&#xc2a4;&#xd15c; &#xac04;&#xc758; &#xc5f0;&#xacb0;&#xc774; &#xbe60;&#xb97c;&#xc218;&#xb85d; &#xc88b;&#xc2b5;&#xb2c8;&#xb2e4;.







```
(DATA FROM "FROM" &#xc624;&#xb978;&#xcabd;&#xc5d0; &#xc624;&#xb294;) &#xc18c;&#xc5d0;
&#xd14c;&#xc774;&#xb14;&#xb85c; &#xc0ac;&#xc6a9;&#xd560; &#xb54c;,, MapServer&#xac00; &#xac01; &#xd589;&#xc5d0;
&#xb300;&#xd55c; &#xc720;&#xc77c;&#xd55c; &#xc2dd;&#xbcc4;&#xc790;&#xb294; &#xbb3c;&#xb860; &#xd14c;&#xc774;&#xc624;
SRID&#xb97c; &#xc790;&#xb3d9;&#xc801;&#xc73c;&#xb85c; &#xd30c;&#xc545;&#xd558;&#xae30; &#xc5b4;&#xb824;&#xc624;
USING &#xc808;&#xc774; MapServer&#xc5d0; &#xc774; &#xb450; &#xac00;&#xc9c0; &#xc815;&#xbcf4;&#xb97c; &#xb2e4;&#xc624;
&#xac19;&#xc774; &#xc81c;&#xacf5;&#xd560; &#xc218; &#xc788;&#xc2b5;&#xb2c8;&#xb2e4;;
```

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

**USING UNIQUE <uniqueid>** &#xb9f5; &#xcffc;&#xb9ac; &#xc2e4;&#xd589;&#xc2dc; &#xc5b4;&#xb5a4; &#xd589;&#xc778;&#xc2dd;&#xbcc4;&#xd558;&#xae30; &#xc704;&#xd574; MapServer&#xac00; &#xac01; &#xd589;&#xc5d0; &#xb300;&#xc720;&#xc77c; ID&#xb97c; &#xc694;&#xad6c;&#xd569;&#xb2c8;&#xb2e4;. &#xc77c;&#xbc18;&#xc801;&#xc73c;&#xc2dc;&#xc2a4;&#xd15c; &#xd14c;&#xc774;&#xb14;&#xc5d0;&#xc11c; &#xae30;&#xbcf8; &#xd0a4;&#xb97c; &#xc778;&#xc2dd;&#xd569;&#xb2c8;&#xb2e4;. &#xd558;&#xc9c0;&#xb9cc;. &#xbdf0; &#xbc0f; &#xd558;&#xc704;&#xc54c;&#xb824;&#xc9c4; &#xc720;&#xc77c;&#xac12; &#xc5f4;&#xc744; &#xc790;&#xb3d9;&#xc801;&#xc73c;&#xb85c; &#xac16;&#xcd94;&#xc9c0; &#xc54a;&#xc2b5;&#xb2c8;&#xb2e4;. MapServer&#xc758; &#xcffc;&#xb9ac; &#xae30;&#xb2d65c;&#xc6a9;&#xd558;&#xae0;&#xc790; &#xd55c;&#xb2e4;&#xba74;. &#xc0ac;&#xc6a9;&#xc790;&#xc758; &#xbdf0; &#xb610;&#xb294; &#xd558;&#xc704;&#xc9d1;&#xd569;&#xc774; &#xc720;&#xc77c;&#xac12; &#xc5f4;&#xc744; &#xd3ec;&#xd568;&#xd558;&#xb294;&#xc9c0; &#xd655;&#xc778;&#xd558;&#xae0;. USING UNIQUE &#xc808;&#xc744; &#xd1b5;&#xd574; &#xadf8; &#xc0ac;&#xc2e4;&#xc744; &#xc120;&#xc5b8;&#xd558;&#xc2ed;&#xc2dc;&#xc624;. &#xc608;&#xb97c; &#xb4e4;&#xc5b4;. &#xc774;&#xb7f0; &#xbaa9;&#xc801;&#xc744; &#xc704;&#xd574; &#xba85;&#xc744; &#xd14c;&#xc774;&#xb14;&#xc758; &#xae30;&#xbcf8; &#xd0a4;&#xb97c; &#xc120;&#xd0dd;&#xd560; &#xc218; &#xc788;&#xae0;. &#xb610;&#xb294; &#xacb0;&#xacfc; &#xcd9c;&#xb825;&#xbb3c;&#xc5d0;&#xc11c; &#xc720;&#xc774; &#xac83;&#xc774; &#xbcf4;&#xc7a5;&#xb418;&#xb294; &#xb2e4;&#xb978; &#xc5b4;&#xb5a4; &#xc5f4;&#xc774;&#xb7f0; &#xc120;&#xd0dd;&#xd560; &#xc218; &#xc788;&#xc2b5;&#xb2c8;&#xb2e4;.

#### Note



"&#xb9f5; &#xcffc;&#xb9ac;&#xd558;&#xae30;"&#xb780; &#xd574;&#xb2f9; &#xc704;&#xc5e8;&#xc758; &#xc815;&#xbcf4;&#xb97c; &#xad6c;&#xd558;&#xae30; &#xc704;&#xd574; &#xb9f5;&#xc744; &#xd074;&#xb9ad;&#xd558;&#xb294; &#xd589;&#xc704;&#xc785;&#xb2c8;&#xb2e4;. DATA &#xc815;&#xc758;&#xc5d0; &#xc788;&#xb294; SQL &#xcffc;&#xb9ac;&#xb97c; &#xd1b5;&#xd55c; "&#xb9f5; &#xcffc;&#xb9ac;"&#xc640; &#xcc29;&#xac01;&#xd558;&#xc9c0; &#xb9c8;&#xc2ed;&#xc2dc;&#xc624;.

**USING SRID=<sruid>** PostGIS&#xb294;. MapServer&#xc5d0; &#xc815;&#xd655;&#xd55c; &#xb370;&#xc774;&#xd130;&#xb97c; &#xbc18;&#xd658;&#xd558;&#xae30; &#xc704;&#xd574;. &#xb3c4;&#xd615;&#xc774; &#xc5b4;&#xb5a4; &#xacf5;&#xac&#xcc38;&#xc870; &#xc2dc;&#xc2a4;&#xd15c;&#xc744; &#xc0ac;&#xc6a9;&#xd558;&#xb294;&#xc9c0; &#xc54c;&#xc54&#xd569;&#xb2c8;&#xb2e4;. &#xc77c;&#xbc18;&#xc801;&#xc73c;&#xb85c; PostGIS &#xb370;&#xc774;&#xd130;&#xbca&#xc788;&#xb294; "geometry\_columns" &#xd14c;&#xc774;&#xb14;&#xc5d0;&#xc11c; &#xc774; &#xc815;&#xbcf4;&#xb97c; &#xcc3e;&#xc744; &#xc218; &#xc788;&#xc9c0;&#xb9cc;. &#xbdf0; &#xbc0f; &#xd558;&#xc704;&#xc9d1;&#xd569;&#xac&#xac19;&#xc774; &#xc2e4;&#xc2dc;&#xac04;&#xc73c;&#xb85c;(on the fly) &#xc0dd;&#xc131;&#xb41c; &#xd14c;&#xc774; &#xbd88;&#xac00;&#xb2a5;&#xd569;&#xb2c8;&#xb2e4;. &#xb530;&#xb77c;&#xc11c; USING SRID= &#xc635;&#xc158; &#xc774;&#xc6a9;&#xd558;&#xba74; DATA &#xc815;&#xc758; &#xc548;&#xc5d0; &#xc815;&#xd655;&#xd55c; SRID&#xc&#xc124;&#xc815;&#xd560; &#xc218; &#xc788;&#xc2b5;&#xb2c8;&#xb2e4;.

### 7.1.4 &#xc608;&#xc81c;

&#xac04;&#xb2e8;&#xd55c; &#xc608;&#xc81c;&#xb85c; &#xc2dc;&#xc791;&#xd574;&#xc11c; &#xb2e8;&#xacc4;&#xb97c; &#xb192;&#xc5ec; &#xac00;&#xbcf4;&#xb3c4;&#xb85d; &#xd569;&#xc2dc;&#xb2e4;. &#xb2e4;&#xc74c; MapServer &#xb808;&#xc815;&#xc758;&#xb97c; &#xc0b4;&#xd3b4;&#xbcf4;&#xc2ed;&#xc2dc;&#xc624;.

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

```
&#xc774; &#xb808;&#xc774;&#xc5b4;&#xb294; &#xb3c4;&#xb85c; &#xd14c;&#xc774;&#xbe14;&#xc758; &#xbaa8;&#xb4e0;
&#xb3c4;&#xb85c; &#xb3c4;&#xd615;&#xc744; &#xac80;&#xc815;&#xc0c9; &#xb77c;&#xc778;&#xc73c;&#xb85c; &#xd45c;&#
&#xac83;&#xc785;&#xb2c8;&#xb2e4;.
```

```
&#xc774;&#xc81c; &#xcd5c;&#xc18c;&#xd55c; 1:100,000 &#xcd95;&#xcc99;&#xc73c;&#xb85c; &#xd655;&#xb300;&#xd558;&#
&#xc804;&#xae4c;&#xc9c0;&#xb294; &#xace0;&#xc18d;&#xb3c4;&#xb85c;&#xb9cc; &#xbcf4;&#xc774;&#xb3c4;&#xb85d;
&#xd558;&#xb824; &#xd55c;&#xb2e4;&#xace0; &#xd574;&#xbd05;&#xc2dc;&#xb2e4;. &#xb2e4;&#xc74c; &#xb450; &#xb808;&
&#xc774;&#xb7f0; &#xd6a8;&#xacfc;&#xb97c; &#xb0bc; &#xac83;&#xc785;&#xb2c8;&#xb2e4;:
```

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

```
LAYER
  CONNECTIONTYPE postgis
  CONNECTION "user=theuser password=thepass dbname=thedb host=theserver"
  PROCESSING "CLOSE_CONNECTION=DEFER"
  DATA "geom from roads"
  MAXSCALE 100000
  STATUS ON
  TYPE LINE
  CLASSITEM road_type
  CLASS
    EXPRESSION "highway"
    STYLE
      WIDTH 2
      COLOR 255 0 0
    END
  END
  CLASS
    STYLE
      COLOR 0 0 0
    END
  END
END
```

```
&#xcd95;&#xcc99;&#xc774; 1:100,000&#xc744; &#xcd08;&#xacfc;&#xd560; &#xacbd;&#xc6b0; &#xccab; &#xbc88;&#xc9f8;
&#xb808;&#xc774;&#xc5b4;&#xb97c; &#xc0ac;&#xc6a9;&#xd574;&#xc11c; "highway" &#xc720;&#xd615;&#xc758; &#xb3c4;&#
```



```

import java.lang.*;
import org.postgis.*;

public class JavaGIS {

public static void main(String[] args) {

    java.sql.Connection conn;

    try {
        /*
        * JDBC <=
        * <=
        */
        Class.forName("org.postgresql.Driver");
        String url = "jdbc:postgresql://localhost:5432/database";
        conn = DriverManager.getConnection(url, "postgres", "");
        /*
        * <=
        * <=
        * <=
        * <=
        */
        ((org.postgresql.PGConnection) conn).addDataType("geometry", Class.forName("org.postgis. <=
        PGgeometry"));
        ((org.postgresql.PGConnection) conn).addDataType("box3d", Class.forName("org.postgis. <=
        PGbox3d"));
        /*
        * <=
        */
        Statement s = conn.createStatement();
        ResultSet r = s.executeQuery("select geom,id from geomtable");
        while( r.next() ) {
            /*
            * <=
            * <=
            * <=
            */
            PGgeometry geom = (PGgeometry)r.getObject(1);
            int id = r.getInt(2);
            System.out.println("Row " + id + ":");
            System.out.println(geom.toString());
        }
        s.close();
        conn.close();
    }
    catch( Exception e ) {
        e.printStackTrace();
    }
}
}

```

"PGgeometry" &#xacc4;&#xb294; &#xd3ec;&#xc778;&#xd2b8;, &#xb77c;&#xc778;&#xc2a4;&#xd2b8;&#xb9c1;, &#xd3f4;&#xb9ac;&#xace4;, &#xba40;&#xd2f0;&#xd3f4;&#xb9ac;&#xace4;, &#xba40;&#xd2f0;&#xb77c;&#xc778;&#xc2a4;&#xb9c1;, &#xba40;&#xd2f0;&#xd3f4;&#xb9ac;&#xace4; &#xb4f1;&#xc758; &#xc720;&#xd615;&#xc5d0; &#xc758;&#xc874;&#xd558;&#xb9c1;, &#xd2b9;&#xc815;&#xc9c0;&#xd615; &#xb3c4;&#xd615; &#xac1d;&#xccb4;(&#xcd94;&#xcd9c; &#xd074;&#xb798;&#xc2a4;

"&#xb3c4;&#xd615;"&#xc758; &#xd558;&#xc704; &#xd074;&#xb798;&#xc2a4;)&#xb97c; &#xb2f4;&#xace0; &#xc788;&#xb294; &#xb798;&#xd37c;(wrapper) &#xac1d;&#xccb4;&#xc785;&#xb2c8;&#xb2e4;.

```
PGgeometry geom = (PGgeometry)r.getObject(1);
if( geom.getType() == Geometry.POLYGON ) {
    Polygon pl = (Polygon)geom.getGeometry();
    for( int r = 0; r < pl.numRings(); r++) {
        LinearRing rng = pl.getRing(r);
        System.out.println("Ring: " + r);
        for( int p = 0; p < rng.numPoints(); p++ ) {
            Point pt = rng.getPoint(p);
            System.out.println("Point: " + p);
            System.out.println(pt.toString());
        }
    }
}
```

&#xd655;&#xc7a5; &#xd504;&#xb85c;&#xadf8;&#xb7a8; &#xac1d;&#xccb4;&#xb97c; &#xc704;&#xd55c; JavaDoc&#xc740; &#xae30;&#xd558;&#xd559;&#xc801; &#xac1d;&#xccb4;&#xc758; &#xb2e4;&#xc591;&#xd55c; &#xb370;&#xc774;&#xd130; &#xc811;&#xadfc;&#xc790;(accessor) &#xd568;&#xc218;&#xc5d0; &#xb300;&#xd55c; &#xcc38;&#xc870;&#xb97c; &#xc81c;&#xc

## 7.3 C &#xd074;&#xb77c;&#xc774;&#xc5b8;&#xd2b8;(libpq)

...

### 7.3.1 &#xd14d;&#xc2a4;&#xd2b8; &#xcee4;&#xc11c;

...

### 7.3.2 &#xbc14;&#xc774;&#xb108;&#xb9ac; &#xcee4;&#xc11c;

...



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

`box2d` objects have a text representation which looks like `BOX (1 2, 5 6)`.

**Box2D Text Representation**

Box2D text representation: `BOX(1 2, 5 6)`

|                       |                            |
|-----------------------|----------------------------|
| <code>box2d</code>    | <code>BOX(1 2, 5 6)</code> |
| <code>geometry</code> | <code>BOX(1 2, 5 6)</code> |

**Box3D Text Representation**

Section 15.7

### 8.1.2 box3d

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

**Box3D Text Representation**

Box3D text representation: `BOX3D(1 2 3, 5 6 7)`

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

`box3d` objects have a text representation which looks like `BOX3D (1 2 3, 5 6 7)`.

**Box3D Text Representation**

Box3D text representation: `BOX3D(1 2 3, 5 6 7)`

|                       |                                  |
|-----------------------|----------------------------------|
| <code>box3d</code>    | <code>BOX3D(1 2 3, 5 6 7)</code> |
| <code>geometry</code> | <code>BOX3D(1 2 3, 5 6 7)</code> |

**Box3D Text Representation**

Section 15.7

### 8.1.3 geometry

`geometry` — geography

#### ;

geography

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

#### ;

;

|           |   |
|-----------|---|
| ;         | ; |
| box       | ; |
| box2d     | ; |
| box3d     | ; |
| bytea     | ; |
| geography | ; |
| text      | ; |

#### ;

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

### 8.1.4 geometry\_dump

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

#### ;

`geometry_dump` is a **composite data type** containing the fields:

- `geom` - a geometry representing a component of the dumped geometry. The geometry type depends on the originating function.
- `path[]` - an integer array that defines the navigation path within the dumped geometry to the `geom` component. The path array is 1-based (i.e. `path[1]` is the first element.)

It is used by the `ST_Dump*` family of functions as an output type to explode a complex geometry into its constituent parts.

#### ;

Section [15.6](#)

### 8.1.5 geography

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

**Geography**

geography — A geography type is a spatial type that stores geographic coordinates in a standard format. It is designed to be used for storing and querying geographic data. It is a superset of the geometry type, meaning that all geometry data can be stored in a geography type. The geography type is designed to be used for storing and querying geographic data. It is a superset of the geometry type, meaning that all geometry data can be stored in a geography type.

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

**Geography Functions**

ST\_GeomFromText(text geography, integer srid) — Returns a geography from a Well-Known Text (WKT) representation. The text parameter is a WKT string, and the srid parameter is the SRID. The function returns a geography type.

|   |           |
|---|-----------|
| ST_GeomFromText(text geography, integer srid) | geography |
| ST_GeomFromTextWKB(bytea wkb, integer srid)   | geography |

**Geography Operators**

Section 4.3, Section 4.3

**8.2 AddGeometryColumn****8.2.1 AddGeometryColumn**

AddGeometryColumn — Adds a geometry column to a table. The function takes the following arguments: table\_name, column\_name, srid, type, dimension, and use\_typmod. The function returns the name of the column that was added.

**Synopsis**

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

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

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

**Parameters**

table\_name — The name of the table to which the column is being added. The name is case-insensitive and can be qualified with a schema name and a catalog name. The name must be a valid table name.

column\_name — The name of the column to be added. The name is case-insensitive and can be qualified with a schema name and a catalog name. The name must be a valid column name.

srid — The SRID of the geometry. The SRID is an integer value. If the SRID is not specified, the function will use the SRID of the table.

type — The type of the geometry. The type is a string value. The valid values are: POINT, LINESTRING, POLYGON, MULTIPOINT, MULTILINESTRING, MULTIPOLYGON, and GEOMETRY. The type must be a valid geometry type.

dimension — The dimension of the geometry. The dimension is an integer value. The valid values are: 2 and 3. The dimension must be a valid dimension value.

use\_typmod — A boolean value that indicates whether to use the type-modifier (typmod) feature. The default value is true. If the value is false, the function will not use the typmod feature.

**Note**

geometry\_columns; ALTER TABLE some\_table ADD COLUMN geom geometry(Point, 4326);

geometry\_columns; ALTER TABLE some\_table ADD COLUMN geom geometry(Point, 4326);

Note!

**Note**

geometry\_columns; ALTER TABLE some\_table ADD COLUMN geom geometry(Point, 4326);

geometry\_columns; ALTER TABLE some\_table ADD COLUMN geom geometry(Point, 4326);

Note!



This method implements the [OGC Simple Features Implementation Specification for SQL 1.1](#).



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



This method supports Circular Strings and Curves

```
ALTER TABLE some_table ADD COLUMN geom geometry(Point, 4326);
```

**ALTER TABLE some\_table ADD COLUMN geom geometry(Point, 4326);**

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

```

-- &#xb2e8;&#xc77c; "id" &#xc5f4;&#xc744; &#xac00;&#xc9c4; &#xb2e8;&#xc21c; ←
  &#xd14c;&#xc774;&#xbel4;&#xc744; &#xbcfc4;&#xc5ec;&#xc8fc;&#xb294; ←
  &#xd14c;&#xc774;&#xbel4;&#xc744; &#xc124;&#xba85;&#xd558;&#xae30;
postgis=# \d my_schema.my_spatial_table
                                     Table "my_schema.my_spatial_table"
Column | Type | Modifiers
-----+-----+-----
id     | integer | not null default nextval('my_schema.my_spatial_table_id_seq'::regclass)
-- &#xd14c;&#xc774;&#xbel4;&#xc5d0; &#xacf5;&#xac04; &#xc5f4;&#xc744; &#xcd94;&#xac00;
SELECT AddGeometryColumn ('my_schema','my_spatial_table','geom',4326,'POINT',2);
-- &#xc608;&#xc804; &#xc81c;&#xc57d;&#xc870;&#xac74; &#xae30;&#xbc18; ←
  &#xc2b5;&#xc131;&#xc744; &#xc774;&#xc6a9;&#xd574;&#xc11c; ←
  &#xd3ec;&#xc778;&#xd2b8;&#xb97c; &#xcd94;&#xac00;
SELECT AddGeometryColumn ('my_schema','my_spatial_table','geom_c',4326,'POINT',2, false);
-- &#xc608;&#xc804; &#xc81c;&#xc57d;&#xc870;&#xac74; &#xc2b5;&#xc131;&#xc744; ←
  &#xc774;&#xc6a9;&#xd574;&#xc11c; &#xb9cc;&#xace1; &#xd3f4;&#xb9ac;&#xace4;&#xc744; ←
  &#xcd94;&#xac00;
SELECT AddGeometryColumn ('my_schema','my_spatial_table','geomcp_c',4326,'CURVEPOLYGON',2, ←
  false);
-- &#xc0c8; &#xb3c4;&#xd615;&#xc744; &#xcd94;&#xac00;&#xd588;&#xc74c;&#xc744; ←
  &#bcfc4;&#xc5ec;&#xc8fc;&#xb3c4;&#xb85d; &#xd14c;&#xc774;&#xbel4;&#xc744; ←
  &#xb2e4;&#xc2dc; &#xc124;&#xba85;&#xd558;&#xae30;
\d my_schema.my_spatial_table
                                     addgeometrycolumn
-----+-----+-----
my_schema.my_spatial_table.geomcp_c SRID:4326 TYPE:CURVEPOLYGON DIMS:2
(1 row)
                                     Table "my_schema.my_spatial_table"
Column | Type | Modifiers
-----+-----+-----
id     | integer | not null default nextval('my_schema. ←
  my_spatial_table_id_seq'::regclass)
geom   | geometry(Point,4326) |
geom_c | geometry |
geomcp_c | geometry |
Check constraints:
  "enforce_dims_geom_c" CHECK (st_ndims(geom_c) = 2)
  "enforce_dims_geomcp_c" CHECK (st_ndims(geomcp_c) = 2)
  "enforce_geotype_geom_c" CHECK (geometrytype(geom_c) = 'POINT'::text OR geom_c IS NULL)
  "enforce_geotype_geomcp_c" CHECK (geometrytype(geomcp_c) = 'CURVEPOLYGON'::text OR ←
  geomcp_c IS NULL)
  "enforce_srid_geom_c" CHECK (st_srid(geom_c) = 4326)
  "enforce_srid_geomcp_c" CHECK (st_srid(geomcp_c) = 4326)
-- geometry_columns &#xbd0;&#xb3c4; &#xc0c8; &#xc5f4;&#xb4e4;&#xc744; ←
  &#xb4f1;&#xb85d;&#xd569;&#xb2c8;&#xb2e4;. --
SELECT f_geometry_column As col_name, type, srid, coord_dimension As ndims
FROM geometry_columns
WHERE f_table_name = 'my_spatial_table' AND f_table_schema = 'my_schema';
col_name | type | srid | ndims
-----+-----+-----+-----
geom     | Point | 4326 | 2
geom_c   | Point | 4326 | 2

```

geomcp\_c | CurvePolygon | 4326 | 2

**&#xad00;&#xb828; &#xc815;&#xbcf4;****DropGeometryColumn, DropGeometryTable**, Section 4.6.2, Section 4.6.3

## 8.2.2 DropGeometryColumn

DropGeometryColumn — &#xacf5;&#xac04; &#xd14c;&#xc774;&#xbe14;&#xc5d0;&#xc11c; &#xc9c0;&#xc624;&#xba54;&#xd2b0;&#xccec;&#xb7fc;&#xc744; &#xc81c;&#xac70;&#xd569;&#xb2c8;&#xb2e4;.

### Synopsis

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

**&#xc124;&#xba85;**

&#xacf5;&#xac04; &#xd14c;&#xc774;&#xbe14;&#xc5d0;&#xc11c; &#xb3c4;&#xd615; &#xc5f4;&#xc744; &#xc81c;&#xac70;&#xc9c0;&#xc624;&#xba54;&#xd2b0;&#xccec;&#xb7fc;&#xc744; &#xc81c;&#xac70;&#xd569;&#xb2c8;&#xb2e4;.  
 schema\_name&#xc774; geometry\_columns &#xd14c;&#xc774;&#xbe14;&#xc5d0; &#xc788;&#xb294; &#xd14c;&#xc774;&#xbe14;&#xc5d0;&#xc788;&#xb294; &#xd14c;&#xc774;&#xbe14;&#xc5d0;&#xc788;&#xb294; f\_table\_schema &#xd56d;&#xbaa9;&#xacfc; &#xc77c;&#xc5e8;&#xd574;&#xc57c; &#xd55c;&#xb2e4;&#xb294; &#xc810;&#xc5d0; &#xc8fc;&#xc758;&#xd558;&#xc2ed;&#xc2dc;&#xc624;.

This method implements the [OGC Simple Features Implementation Specification for SQL 1.1](#).

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



This method supports Circular Strings and Curves

### Note

&#xbcc0;&#xacbd; &#xc0ac;&#xd56d;: 2.0.0 &#xc84;&#xc804;: &#xd558;&#xc704; &#xd638;&#xd658;&#xc131;&#xc744; &#xc704;&#xd574; &#xc774; &#xd568;&#xc218;&#xb97c; &#xc81c;&#xacf5;&#xd569;&#xb2c8;&#xb2e4;: &#xc774;&#xc81c;&#xb294; geometry\_columns&#xc774; &#xc2dc;&#xc2a4;&#xd15c; &#xc74;&#xd0c8;&#xb85c;&#xadf8;&#xb97c; &#xae30;&#xc18;&#xc73c;&#xb85c; &#xd558;&#xb294; &#xbdff0;&#xc774;&#xae30; &#xb54c;&#xbb38;&#xc5d0; &#xb2e4;&#xb978; &#xc5b4;&#xb5a4; &#xd14c;&#xc774;&#xbe14; &#xc5f4;&#xacfc;&#xb3c4; &#xb9c8;&#xc2c;&#xc00;&#xc9c0;&#xb85c; &#xb3c4;&#xd615; &#xc5f4;&#xb3c4; ALTER TABLE &#xc774; &#xc774;&#xc6a9;&#xd574;&#xc11c; &#xc0ad;&#xc81c;&#xd560; &#xc218; &#xc788;&#xc2b5;&#xb2c8;&#xb2e4;.

**&#xc608;&#xc81c;**

```
SELECT DropGeometryColumn ('my_schema', 'my_spatial_table', 'geom');
-- &#xacb0;&#xacfc; &#xcd9c; &#xb825; &#xbb3c; --
-- dropgeometrycolumn
-----
my_schema.my_spatial_table.geom effectively removed.

-- PostGIS 2.0 &#xc774;&#xc0c1; &#xc84;&#xc804;&#xc5d0;&#xc11c; &#xc704; &#xcffc;&#xb9ac;&#xb294; &#xd45c;&#xc900; &#xb300;&#xc2cb4; &#xc774; &#xc774;&#xbe14;&#xacfc;&#xb3c4; &#xb3d9;&#xb4f1;&#xd569;&#xb2c8;&#xb2e4; .
-- &#xc591;&#xcabd; &#xbaa8;&#xb450; geometry_columns&#xb85c;&#xbd80;&#xd130; &#xc774; &#xb4f1;&#xb85d; &#xd574;&#xc81c;&#xb420; &#xac83;&#xc785;&#xb2c8;&#xb2e4; .
ALTER TABLE my_schema.my_spatial_table DROP column geom;
```









SRID, schema-aware postgresql installations, current\_schema(),



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



This method supports Circular Strings and Curves

## UpdateGeometrySRID

Insert geometries into roads table with a SRID set already using **EWKT format**:

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

```
'unknown' SRID=4326;LINESTRING(0 0, 10 10)
SRID=4326;LINESTRING(10 10, 15 0)
\.
```

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

```
DDL MULTILINESTRING(4326) SRID=4326;LINESTRING(0 0, 10 10)
SRID=4326;LINESTRING(10 10, 15 0)
\.
```

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

```
'unknown' SRID=4326;LINESTRING(0 0, 10 10)
SRID=4326;LINESTRING(10 10, 15 0)
\.
```

PostGIS UpdateGeometrySRID, UpdateRasterSRID, ST\_SetSRID, ST\_Transform

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

## UpdateRasterSRID, ST\_SetSRID, ST\_Transform

UpdateRasterSRID, ST\_SetSRID, ST\_Transform

## 8.3 ST\_GeomCollFromText (constructor)

### 8.3.1 ST\_GeomCollFromText

ST\_GeomCollFromText — Creates a GeometryCollection or Multi\* geometry from a set of geometries.

#### Synopsis

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



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

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

**Using an array constructor for a subquery.**

Using an array constructor for a subquery.

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

Using an array constructor for values.

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

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

**Creating multiple collections by grouping geometries in a table.**

Creating multiple collections by grouping geometries in a table.

```
SELECT stusps, ST_Collect(f.geom) as geom
    FROM (SELECT stusps, (ST_Dump(geom)).geom As geom
          FROM
          somestatetable ) As f
    GROUP BY stusps
```

**ST\_Dump, ST\_AsBinary**

**ST\_Dump, ST\_AsBinary**

### 8.3.2 ST\_LineFromMultiPoint

**ST\_LineFromMultiPoint** — Returns a LineString geometry from a MultiPoint geometry.

#### Synopsis

geometry **ST\_LineFromMultiPoint**(geometry aMultiPoint);

**ST\_LineFromMultiPoint**

Returns a LineString geometry from a MultiPoint geometry.

Use **ST\_MakeLine** to create lines from Point or LineString inputs.



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





```
&#xd3ec;&#xc778;&#xd2b8;,&#xba40;&#xd2f0;&#xd3ec;&#xc778;&#xd2b8; &#xb610;&#xb294; &#xb77c;&#xc778; &#xb3c4;&#xb77c;&#xc778;&#xc2a4;&#xd2b8;&#xb9c1;&#xc744; &#xc0dd;&#xc131;&#xd569;&#xb2c8;&#xb2e4;.
```

```
select ST_AsText( ST_MakeLine( 'LINESTRING(0 0, 1 1)', 'LINESTRING(2 2, 3 3)' ) );

      st_astext
-----
LINESTRING(0 0,1 1,2 2,3 3)
```

**&#xc608;&#xc2dc;: &#xbc30;&#xc5f4; &#xbc84;&#xc804; &#xc0ac;&#xc6a9;&#xd558;&#xae30;**

Create a line from an array formed by a subquery with ordering.

```
SELECT ST_MakeLine( ARRAY( SELECT ST_Centroid(geom) FROM visit_locations ORDER BY visit_time) );
```

Create a 3D line from an array of 3D points

```
SELECT ST_MakeLine(ARRAY(SELECT ST_Centroid(the_geom) FROM visit_locations ORDER BY visit_time));

-- 3D &#xd3ec;&#xc778;&#xd2b8; 3&#xac1c;&#xb85c; 3D &#xb77c;&#xc778; &#xb9cc;&#xb4e4;&#xae30;
SELECT ST_AsEWKT(ST_MakeLine(ARRAY[ST_MakePoint(1,2,3),
                                ST_MakePoint(3,4,5), ST_MakePoint(6,6,6)]));

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

**&#xc608;&#xc2dc;: &#xacf5;&#xac04; &#xd569;&#xc0b0; &#xbc84;&#xc804;**

```
&#xc774; &#xc608;&#xc2dc;&#xb294; GPS &#xd3ec;&#xc778;&#xd2b8; &#xbc30;&#xc5f4;&#xc744; &#xc785;&#xb825;&#xbc1
&#xb3c4;&#xd615; &#xd56d;&#xbaa9;&#xc774; &#xc774;&#xb3d9; &#xc21c;&#xc11c;&#xb300;&#xb85c;&#xc758; GPS
&#xd3ec;&#xc778;&#xd2b8;&#xb4e4;&#xb85c; &#xc774;&#xb8e8;&#xc5b4;&#xc9c4; &#xb77c;&#xc778;&#xc2a4;&#xd2b8;&#xc
GPS &#xc774;&#xb3d9; &#xd558;&#xb098;&#xb2f9; &#xd55c; &#xac1c;&#xc758; &#xb808;&#xcf54;&#xb4dc;&#xb97c;
&#xc0dd;&#xc131;&#xd569;&#xb2c8;&#xb2e4;.
```

Using aggregate ORDER BY provides a correctly-ordered LineString.

```
SELECT gps.track_id, ST_MakeLine(gps.geom ORDER BY gps_time) As geom
FROM gps_points As gps
GROUP BY track_id;
```

Prior to PostgreSQL 9, ordering in a subquery can be used. However, sometimes the query plan may not respect the order of the subquery.

```
SELECT gps.track_id, ST_MakeLine(gps.geom) As geom
FROM ( SELECT track_id, gps_time, geom
FROM gps_points ORDER BY track_id, gps_time ) As gps
GROUP BY track_id;
```

**&#xcc38;&#xace0;**

[ST\\_RemoveRepeatedPoints](#), [ST\\_AsText](#), [ST\\_GeomFromText](#), [ST\\_MakePoint](#)

### 8.3.5 ST\_MakePoint

ST\_MakePoint — Creates a 2D, 3DZ or 4D Point.



## Synopsis

geometry **ST\_MakePointM**(float x, float y, float m);

**Parameters:**

x, y and m: float values representing longitude, latitude, and elevation (in meters).

Use **ST\_MakePoint** to make points with XY, XYZ, or XYZM coordinates.



### Note

For geodetic coordinates, X is longitude and Y is latitude

**Examples:**



### Note

**ST\_AsEWKT** is used for text output because **ST\_AsText** does not support M values.

Create point with unknown SRID.

```
SELECT ST_AsEWKT( ST_MakePointM(-71.1043443253471, 42.3150676015829, 10) );

          st_asewkt
-----
POINTM(-71.1043443253471 42.3150676015829 10)
```

x, y and m: float values representing longitude, latitude, and elevation (in meters).

```
SELECT ST_AsEWKT( ST_SetSRID( ST_MakePointM(-71.104, 42.315, 10), 4326) );

          st_asewkt
-----
SRID=4326;POINTM(-71.104 42.315 10)
```

Get measure of created point.

```
SELECT ST_M( ST_MakePointM(-71.104, 42.315, 10) );

result
-----
10
```

**See also:**

**ST\_AsEWKT**, **ST\_MakePoint**, **ST\_SetSRID**

## 8.3.7 ST\_MakePolygon

**ST\_MakePolygon** — Creates a Polygon from a shell and optional list of holes.

## Synopsis

geometry **ST\_MakePolygon**(geometry linestring);

geometry **ST\_MakePolygon**(geometry outerlinestring, geometry[] interiorlinestrings);

### Parameters

**linestring**: A **LINESTRING** geometry. If the linestring is closed, the function will return a **POLYGON**. If the linestring is not closed, the function will return a **MULTILINESTRING**. If the linestring is a **MULTILINESTRING**, the function will return a **MULTILINESTRING**. If the linestring is a **MULTILINESTRING**, the function will return a **MULTILINESTRING**.

**Variant 1:** Accepts one shell **LineString**.

**Variant 2:** Accepts a shell **LineString** and an array of inner (hole) **LineStrings**. A geometry array can be constructed using the PostgreSQL `array_agg()`, `ARRAY[]` or `ARRAY()` constructs.

### Note



**ST\_MakePolygon** will return a **POLYGON** if the linestring is closed. If the linestring is not closed, the function will return a **MULTILINESTRING**. If the linestring is a **MULTILINESTRING**, the function will return a **MULTILINESTRING**. If the linestring is a **MULTILINESTRING**, the function will return a **MULTILINESTRING**.



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

### Examples

Convert a **MULTILINESTRING** to a **POLYGON** using **ST\_MLineFromText**:

```
SELECT ST_MLineFromText('MULTILINESTRING((1 2, 3 4), (4 5, 6 7))');
```

Create a **Polygon** from an open **LineString**, using **ST\_StartPoint** and **ST\_AddPoint** to close it.

```
SELECT ST_MakePolygon( ST_AddPoint(foo.open_line, ST_StartPoint(foo.open_line)) )
FROM (
  SELECT ST_GeomFromText('LINESTRING(75 29,77 29,77 29, 75 29)') As open_line) As foo;
```

Convert a **LINESTRING** to a **POLYGON** using **ST\_MakePolygon**:

```
SELECT ST_AsEWKT( ST_MakePolygon( 'LINESTRING(75.15 29.53 1,77 29 1,77.6 29.5 1, 75.15 29.53 1)') );
```

```
st_asewkt
```

```
-----
POLYGON((75.15 29.53 1,77 29 1,77.6 29.5 1,75.15 29.53 1))
```

Create a **Polygon** from a **LineString** with measures

```
SELECT ST_AsEWKT( ST_MakePolygon( 'LINESTRINGM(75.15 29.53 1,77 29 1,77.6 29.5 2, 75.15 29.53 2)') );
```

```
st_asewkt
```

```
-----
POLYGONM((75.15 29.53 1,77 29 1,77.6 29.5 2,75.15 29.53 2))
```



```

        ST_MakePolygon(ST_LineMerge(ST_Boundary(p.the_geom)),
                      ARRAY(SELECT w.the_geom
                            FROM waterlines w
                            WHERE ST_Within(w.the_geom, p.the_geom)
                            AND ST_IsClosed(w.the_geom)))
    ELSE p.the_geom END As the_geom
FROM
    provinces p;

```

**ST\_BuildArea**

**ST\_Polygon**

### 8.3.8 ST\_Point

**ST\_Point** — Creates a Point with X, Y and SRID values.

#### Synopsis

geometry **ST\_Point**(float x\_lon, float y\_lat);

geometry **ST\_MakePointM**(float x, float y, float m);

**ST\_Point**

Returns a Point with the given X and Y coordinate values. This is the SQL-MM equivalent for **ST\_MakePoint** that takes just X and Y.



#### Note

For geodetic coordinates, X is longitude and Y is latitude

Enhanced: 3.2.0 srid as an extra optional argument was added. Older installs require combining with **ST\_SetSRID** to mark the srid on the geometry.



This method implements the SQL/MM specification. SQL-MM 3: 6.1.2

**ST\_Point**

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

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

New in 3.2.0: With SRID specified

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

**Pre-PostGIS 3.2 syntax**

Pre-PostGIS 3.2 syntax

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

3.2 and on you can include the srid

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

PostgreSQL also provides the `::` short-hand for casting

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

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

```
SELECT CAST(ST_SetSRID(ST_Point(-71.1043443253471, 42.3150676015829), 4326) As geography);
```

**Section 4.3, [ST\\_MakePoint](#), [ST\\_SetSRID](#), [ST\\_Transform](#), [ST\\_Point](#), [ST\\_Point](#), [ST\\_Point](#)**

Section 4.3, [ST\\_MakePoint](#), [ST\\_SetSRID](#), [ST\\_Transform](#), [ST\\_Point](#), [ST\\_Point](#), [ST\\_Point](#)

**8.3.9 ST\_Point**

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

**Synopsis**

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

**Enhanced:**

3.2.0 srid as an extra optional argument was added. Older installs require combining with `ST_SetSRID` to mark the srid on the geometry.

Enhanced: 3.2.0 srid as an extra optional argument was added. Older installs require combining with `ST_SetSRID` to mark the srid on the geometry.

**SELECT ST\_SetSRID(ST\_Point(-71.1043443253471, 42.3150676015829), 4326)**

```
SELECT ST_SetSRID(ST_Point(-71.1043443253471, 42.3150676015829), 4326)
```

```
SELECT ST_SetSRID(ST_Point(-71.1043443253471, 42.3150676015829), 4326)
```

```
SELECT ST_SetSRID(ST_Point(-71.1043443253471, 42.3150676015829), 4326)
```

**ST\_MakePoint, ST\_PointFromText, ST\_SetSRID, ST\_MakePointM**

**[ST\\_MakePoint](#), [ST\\_PointFromText](#), [ST\\_SetSRID](#), [ST\\_MakePointM](#)**

### 8.3.10 ST\_Point

ST\_Point — Creates a Point with X, Y, M and SRID values.

#### Synopsis

geometry **ST\_PointM**(float x, float y, float m, integer srid=unknown);

#### Enhanced:

ST\_PointM(float x, float y, float m, integer srid=unknown);  
 ST\_MakePoint(float x, float y, float m, integer srid=unknown);  
 OGC ST\_MakePoint(float x, float y, float m, integer srid=unknown);

Enhanced: 3.2.0 srid as an extra optional argument was added. Older installs require combining with ST\_SetSRID to mark the srid on the geometry.

#### Examples:

```
SELECT ST_SetSRID(ST_Point(-71.1043443253471, 42.3150676015829), 4326)
```

```
SELECT ST_SetSRID(ST_Point(-71.1043443253471, 42.3150676015829), 4326)
```

```
SELECT ST_SetSRID(ST_Point(-71.1043443253471, 42.3150676015829), 4326)
```

#### Related Functions:

[ST\\_MakePoint](#), [ST\\_PointFromText](#), [ST\\_SetSRID](#), [ST\\_MakePointM](#)

### 8.3.11 ST\_Point

ST\_Point — Creates a Point with X, Y, Z, M and SRID values.

#### Synopsis

geometry **ST\_MakeEnvelope**(double precision xmin, double precision ymin, double precision xmax, double precision ymax, integer srid=unknown);

#### Enhanced:

ST\_MakeEnvelope(double precision xmin, double precision ymin, double precision xmax, double precision ymax, integer srid=unknown);  
 ST\_MakePoint(float x, float y, float m, integer srid=unknown);  
 OGC ST\_MakePoint(float x, float y, float m, integer srid=unknown);

Enhanced: 3.2.0 srid as an extra optional argument was added. Older installs require combining with ST\_SetSRID to mark the srid on the geometry.

#### Examples:

```
SELECT ST_SetSRID(ST_Point(-71.1043443253471, 42.3150676015829), 4326)
```

```
SELECT ST_SetSRID(ST_Point(-71.1043443253471, 42.3150676015829), 4326)
```

```
SELECT ST_SetSRID(ST_Point(-71.1043443253471, 42.3150676015829), 4326)
```

**ST\_MakePoint**

[ST\\_MakePoint](#), [ST\\_Point](#), [ST\\_Point](#), [ST\\_Point](#), [ST\\_SetSRID](#)

### 8.3.12 ST\_Polygon

**ST\_Polygon** — Creates a Polygon from a LineString with a specified SRID.

#### Synopsis

geometry **ST\_Polygon**(geometry aLineString, integer srid);

**ST\_Polygon**

Returns a polygon built from the given LineString and sets the spatial reference system from the `srid`.

**ST\_Polygon** is similar to [ST\\_MakePolygon](#) Variant 1 with the addition of setting the SRID.

, [ST\\_MakePoint](#), [ST\\_SetSRID](#)



#### Note

**ST\_Polygon** is similar to [ST\\_MakePolygon](#) Variant 1 with the addition of setting the SRID. **ST\_Polygon** is similar to [ST\\_MakePolygon](#) Variant 1 with the addition of setting the SRID. **ST\_LineMerge** [ST\\_LineMerge](#) [ST\\_Dump](#)

- This method implements the [OGC Simple Features Implementation Specification for SQL 1.1](#).
- This method implements the SQL/MM specification. SQL-MM 3: 8.3.2
- This function supports 3d and will not drop the z-index.

**ST\_Polygon**

Create a 2D polygon.

```
SELECT ST_AsText( ST_Polygon('LINESTRING(75 29, 77 29, 77 29, 75 29)::geometry, 4326) );
-- result --
POLYGON((75 29, 77 29, 77 29, 75 29))
```

Create a 3D polygon.

```
SELECT ST_AsEWKT( ST_Polygon( ST_GeomFromEWKT('LINESTRING(75 29 1, 77 29 2, 77 29 3, 75 29 1)'), 4326) );
-- result --
SRID=4326;POLYGON((75 29 1, 77 29 2, 77 29 3, 75 29 1))
```

**ST\_AsEWKT**

[ST\\_AsEWKT](#), [ST\\_AsText](#), [ST\\_GeomFromEWKT](#), [ST\\_GeomFromText](#), [ST\\_LineMerge](#), [ST\\_MakePolygon](#)

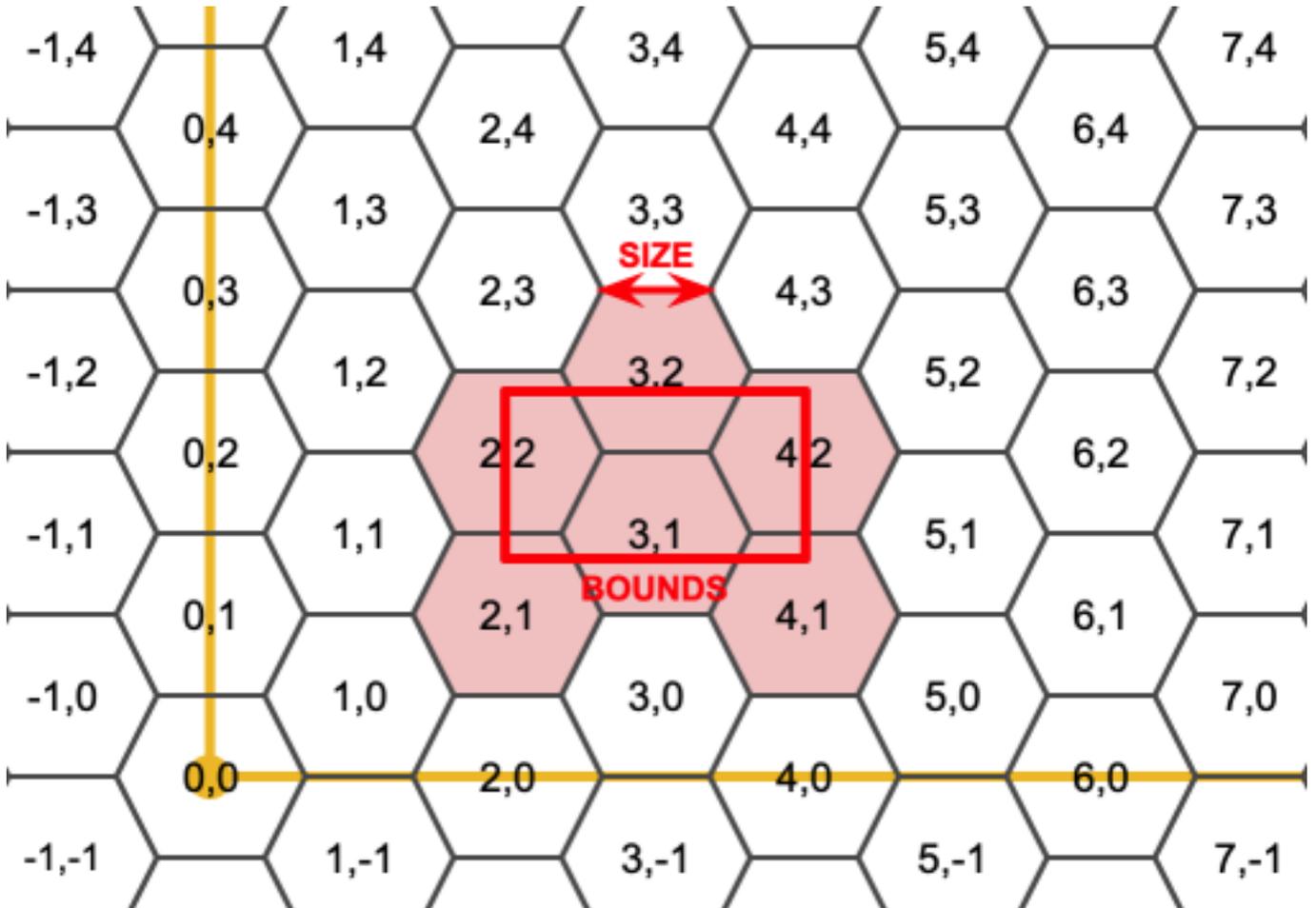


**Synopsis**

geometry **ST\_Point**(float x\_lon, float y\_lat);

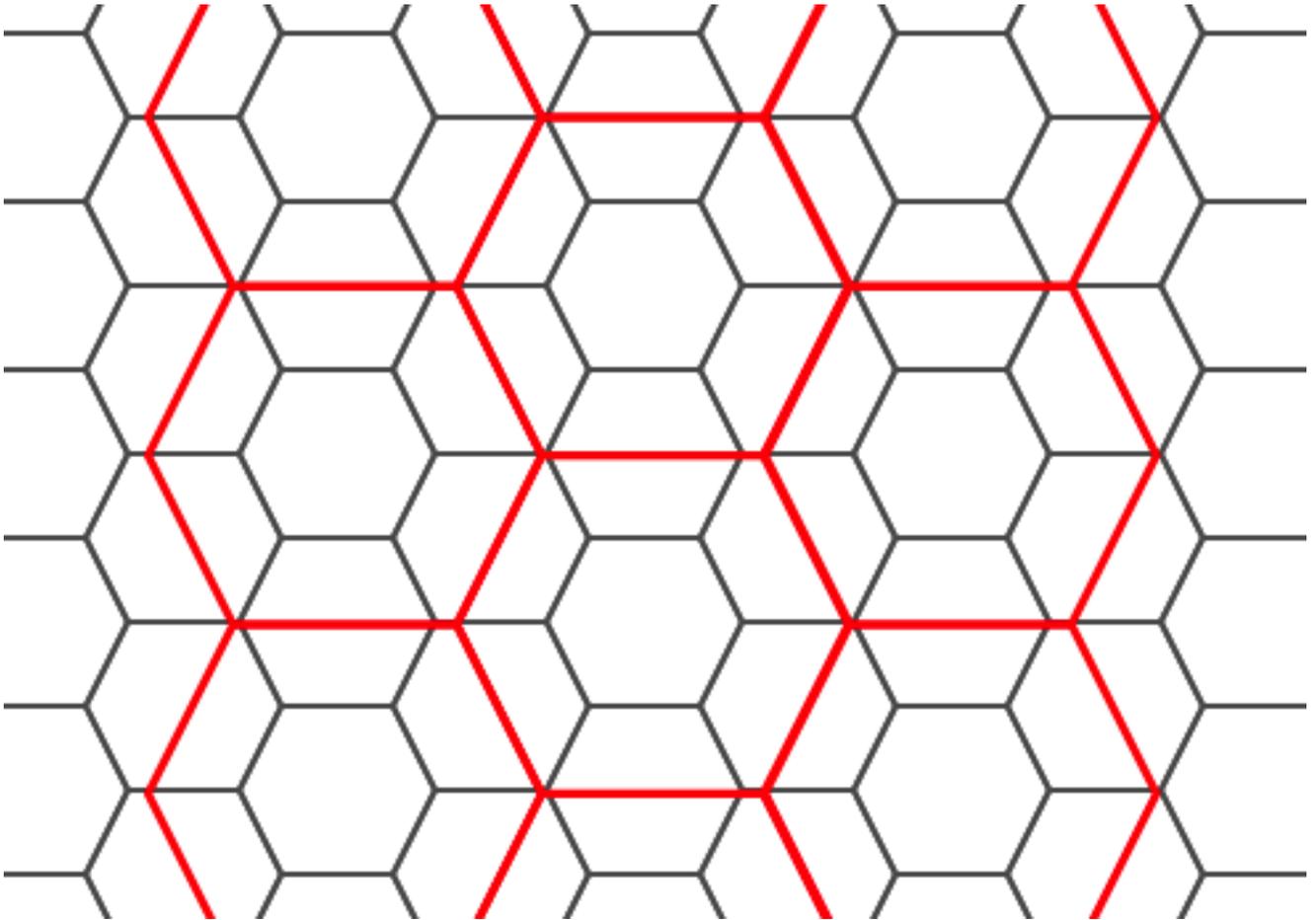
**&#xc124;&#xba85;**

Starts with the concept of a hexagon tiling of the plane. (Not a hexagon tiling of the globe, this is not the **H3** tiling scheme.) For a given planar SRS, and a given edge size, starting at the origin of the SRS, there is one unique hexagonal tiling of the plane, **Tiling(SRS, Size)**. This function answers the question: what hexagons in a given **Tiling(SRS, Size)** overlap with a given bounds.



The SRS for the output hexagons is the SRS provided by the bounds geometry.

Doubling or tripling the edge size of the hexagon generates a new parent tiling that fits with the origin tiling. Unfortunately, it is not possible to generate parent hexagon tilings that the child tiles perfectly fit inside.



2.1.0

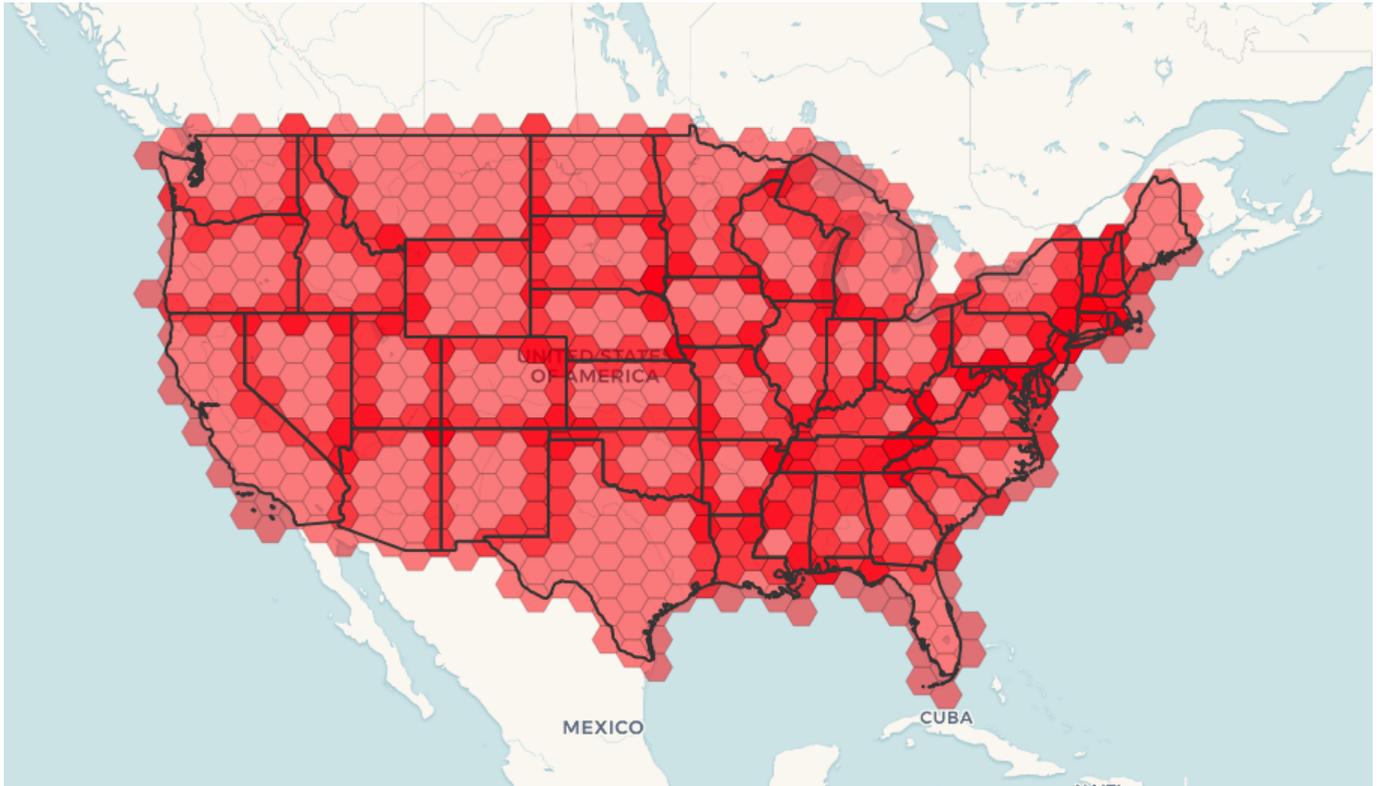
**Hexagonal Grid**

To do a point summary against a hexagonal tiling, generate a hexagon grid using the extent of the points as the bounds, then spatially join to that grid.

```
SELECT COUNT(*), hexes.geom
FROM
  ST_HexagonGrid(
    10000,
    ST_SetSRID(ST_EstimatedExtent('pointtable', 'geom'), 3857)
  ) AS hexes
INNER JOIN
  pointtable AS pts
  ON ST_Intersects(pts.geom, hexes.geom)
GROUP BY hexes.geom;
```

**Hexagonal Grid**

If we generate a set of hexagons for each polygon boundary and filter out those that do not intersect their hexagons, we end up with a tiling for each polygon.



Tiling states results in a hexagon coverage of each state, and multiple hexagons overlapping at the borders between states.

**Note**

The LATERAL keyword is implied for set-returning functions when referring to a prior table in the FROM list. So CROSS JOIN LATERAL, CROSS JOIN, or just plain , are equivalent constructs for this example.

```
SELECT admin1.gid, hex.geom
FROM
  admin1
  CROSS JOIN
  ST_HexagonGrid(100000, admin1.geom) AS hex
WHERE
  adm0_a3 = 'USA'
  AND
  ST_Intersects(admin1.geom, hex.geom)
```

**ST\_Hexagon**

[ST\\_EstimatedExtent](#), [ST\\_MakePoint](#), [ST\\_Point](#), [ST\\_SRID](#)

### 8.3.15 ST\_Hexagon

**ST\_Hexagon** — Returns a single hexagon, using the provided edge size and cell coordinate within the hexagon grid space.

**Synopsis**

geometry **ST\_MakePoint**(double precision x, double precision y, double precision z, double precision m);

**ST\_Hexagon**

Uses the same hexagon tiling concept as [ST\\_HexagonGrid](#), but generates just one hexagon at the desired cell coordinate. Optionally, can adjust origin coordinate of the tiling, the default origin is at 0,0.

Hexagons are generated with no SRID set, so use [ST\\_SetSRID](#) to set the SRID to the one you expect.

2.1.0 [ST\\_Hexagon](#), [ST\\_HexagonGrid](#), [ST\\_SetSRID](#), [ST\\_MakeEnvelope](#), [ST\\_MakePoint](#), [ST\\_SetSRID](#)

**Example: Creating a hexagon at the origin**

```
SELECT ST_AsText(ST_SetSRID(ST_Hexagon(1.0, 0, 0), 3857));

POLYGON((-1 0,-0.5
         -0.866025403784439,0.5
         -0.866025403784439,1
         0,0.5
         0.866025403784439,-0.5
         0.866025403784439,-1 0))
```

**ST\_MakeEnvelope**

[ST\\_MakeEnvelope](#), [ST\\_MakePoint](#), [ST\\_SetSRID](#)

**8.3.16 ST\_SquareGrid**

**ST\_SquareGrid** — Returns a set of grid squares and cell indices that completely cover the bounds of the geometry argument.

**Synopsis**

geometry **ST\_Point**(float x\_lon, float y\_lat);

**ST\_SquareGrid**

Starts with the concept of a square tiling of the plane. For a given planar SRS, and a given edge size, starting at the origin of the SRS, there is one unique square tiling of the plane, `Tiling(SRS, Size)`. This function answers the question: what grids in a given `Tiling(SRS, Size)` overlap with a given bounds.

The SRS for the output squares is the SRS provided by the bounds geometry.

Doubling or edge size of the square generates a new parent tiling that perfectly fits with the original tiling. Standard web map tilings in mercator are just powers-of-two square grids in the mercator plane.

2.1.0 [ST\\_SquareGrid](#), [ST\\_MakeEnvelope](#), [ST\\_MakePoint](#), [ST\\_SetSRID](#), [ST\\_Hexagon](#), [ST\\_HexagonGrid](#), [ST\\_SetSRID](#)

**ST\_SquareGrid**

The grid will fill the whole bounds of the country, so if you want just squares that touch the country you will have to filter afterwards with [ST\\_Intersects](#).

```
WITH grid AS (
SELECT (ST_SquareGrid(1, ST_Transform(geom,4326))).*
FROM admin0 WHERE name = 'Canada'
)
SELECT ST_AsText(geom)
FROM grid
```





**Example: Scaling and moving words**

```
SELECT ST_Translate(ST_Scale(ST_Letters('Yo'), 10, 10), 100, 100);
```

**ST\_AsTWKB, ST\_Scale, ST\_Translate**

**8.4 ST\_GeometryAccessor(accessor)****8.4.1 ST\_GeometryType(geometry geomA);**

**ST\_GeometryType**(geometry geomA); — Returns the geometry type of the input geometry.

**Synopsis**

text **GeometryType**(geometry geomA);

**Parameters**

**geomA**: A geometry object. The geometry type is returned as a text string. The geometry type is returned as a text string. The geometry type is returned as a text string.

OGC Simple Features Implementation Specification for SQL 1.1 - 2.1.1.1 - GeometryType(geometry geomA);

**Note**

The geometry type is returned as a text string. The geometry type is returned as a text string. The geometry type is returned as a text string.

2.0.0 - GeometryType(geometry geomA);



This method implements the [OGC Simple Features Implementation Specification for SQL 1.1](#).



This method supports Circular Strings and Curves



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



This function supports Polyhedral surfaces.



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

**ST\_GeometryType**

```
SELECT GeometryType(ST_GeomFromText('LINESTRING(77.29 29.07,77.42 29.26,77.27 29.31,77.29 29.07)'));
geometrytype
-----
LINESTRING
```

```
SELECT ST_GeometryType(ST_GeomFromEWKT('POLYHEDRALSURFACE( ((0 0 0, 0 0 1, 0 1 1, 0 1 0, 0 0 0)),
((0 0 0, 0 1 0, 1 1 0, 1 0 0, 0 0 0)), ((0 0 0, 1 0 0, 1 0 1, 0 0 1, 0 0 0)),
((1 1 0, 1 1 1, 1 0 1, 1 0 0, 1 1 0)),
((0 1 0, 0 1 1, 1 1 1, 1 1 0, 0 1 0)), ((0 0 1, 1 0 1, 1 1 1, 0 1 1, 0 0 1))
) )'));
-- POLYHEDRALSURFACE
```

```
SELECT GeometryType(geom) as result
FROM
  (SELECT
    ST_GeomFromEWKT('TIN (((
      0 0 0,
      0 0 1,
      0 1 0,
      0 0 0
    )), ((
      0 0 0,
      0 1 0,
      1 1 0,
      0 0 0
    ))) AS geom
  ) AS g;
result
-----
TIN
```

**ST\_GeometryType****ST\_GeometryType****8.4.2 ST\_Boundary**

**ST\_Boundary** — Returns the boundary of a geometry. The boundary of a geometry is the set of points that form the outer edge of the geometry. The boundary of a point is the point itself. The boundary of a line is the line itself. The boundary of a polygon is the set of lines that form the outer edge of the polygon.

**Synopsis**

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

**ST\_Boundary**

Returns the boundary of a geometry. The boundary of a geometry is the set of points that form the outer edge of the geometry. The boundary of a point is the point itself. The boundary of a line is the line itself. The boundary of a polygon is the set of lines that form the outer edge of the polygon.

boundary) OGC 3.12.3.2 &#x2e8;&#xc6d0;&#xc774; &#xc124;&#xba85;&#xd558 &#xc815;&#xc758;&#xb429;&#xb2c8;&#xb2e4;. &#xc774; &#xd568;&#xc218;&#xc758; &#xac0;&#xacfc;&#xac00; &#xb2eb;&#xb54c;&#xbb38;&#xc5d0;, &#xc989; &#xc704;&#xc0c1;&#xc801;(&#x4f4d;&#x76f8;&#x7684;)&#xc73c;&#xb85c; &#xd3d0;&#xb54c;&#xbb38;&#xc5d0;, OGC &#xc0ac;&#xc591;&#xc11c; 3.12.2 &#xb2e8;&#xc6d0;&#xc5d0;&#xc11c; &#xc124;&#xba85;&#xd45c;&#xd604;&#xc801;&#xc778; &#xb3c4;&#xd615; &#xc6d0;&#xd615;(primitive)&#xc744; &#xc774;&#xc6a9;&#xd574;&#xac0;&#xacfc; &#xb94;&#xc704;&#xb97c; &#xd45c;&#xd604;&#xd560; &#xc218; &#xc788;&#xc2b5;&#xb2c8;&#xb2e4;. GEOS &#xaa8;&#xb4c8;&#xb85c; &#xc2e4;&#xd589;

**Note**



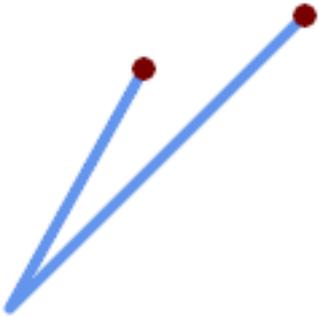
2.0.0 &#xbbf8;&#xb9cc; &#xbc84;&#xc804;&#xc5d0;&#xc11c; &#xc774; &#xd568;&#xc218;&#xb97c; GEOMETRYCOLLECTION&#xacfc; &#xd568;&#xaed8; &#xc0ac;&#xc6a9;&#xd558;&#xba74; &#xc608;&#xc678;&#xac00; &#xbc1c;&#xc0dd;&#xd588;&#xc2b5;&#xb2c8;&#xb2e4;. 2.0.0 &#xc774;&#xd6c4; &#xbc84;&#xc804;&#xc740; &#xb300;&#xc2e0; (&#xc785;&#xb825;&#xc744; &#xc9c0;&#xc6d0;&#xd558;&#xc9c0; &#xc54a;&#xb294;&#xb2e4;&#xb294; &#xc758;&#xbbf8;&#xc758;) NULL&#xc744; &#xbc18;&#xd658;&#xd569;&#xb2c8;&#xb2e4;.

-  This method implements the **OGC Simple Features Implementation Specification for SQL 1.1**. OGC SPEC s2.1.1.1
-  This method implements the SQL/MM specification. SQL-MM IEC 13249-3: 5.1.17
-  This function supports 3d and will not drop the z-index.

&#xac1c;&#xc120; &#xc0ac;&#xd56d;: 2.1.0 &#xbc84;&#xc804;&#xbd80;&#xd130; &#xc0bc;&#xac01;&#xd615;&#xc744; &#xc9c0;&#xc6d0;&#xd558;&#xae30; &#xc2dc;&#xc791;&#xd588;&#xc2b5;&#xb2c8;&#xb2e4;.

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

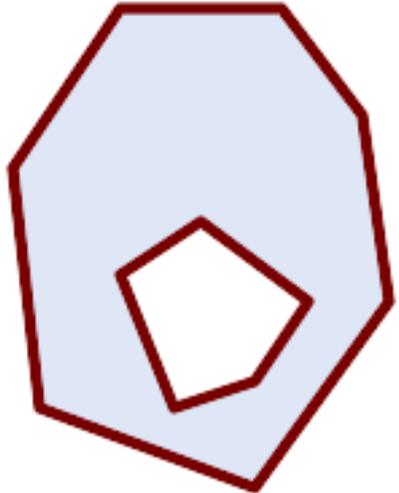
**&#xc608;&#xc2dc;**



```

SELECT ST_Boundary(geom)
FROM (SELECT 'LINESTRING(100 150,50 60,
70 80, 160 170)::geometry As geom) As f;

-- ST_AsText output
MULTIPOINT((100 150),(160 170))
    
```



```

SELECT ST_Boundary(geom)
FROM (SELECT
'POLYGON (( 10 130, 50 190, 110 190, 140
150, 150 80, 100 10, 20 40, 10 130 ),
( 70 40, 100 50, 120 80, 80 110,
50 90, 70 40 ))::geometry As geom) As f;

-- ST_AsText \&\#xcd9c;\&\#xb825;
MULTILINESTRING((10 130,50 190,110
190,140 150,150 80,100 10,20 40,10 130),
(70 40,100 50,120 80,80 110,50
90,70 40))
    
```

```

SELECT ST_AsText (ST_Boundary(ST_GeomFromText ('LINESTRING(1 1,0 0, -1 1)')));
st_astext
-----
MULTIPOINT((1 1),(-1 1))

SELECT ST_AsText (ST_Boundary(ST_GeomFromText ('POLYGON((1 1,0 0, -1 1, 1 1)'))));
st_astext
-----
LINESTRING(1 1,0 0,-1 1,1 1)

--Using a 3d polygon
SELECT ST_AsEWKT(ST_Boundary(ST_GeomFromEWKT('POLYGON((1 1 1,0 0 1, -1 1 1, 1 1 1)'))));
st_asewkt
-----
LINESTRING(1 1 1,0 0 1,-1 1 1,1 1 1)

--Using a 3d multilinestring
SELECT ST_AsEWKT(ST_Boundary(ST_GeomFromEWKT('MULTILINESTRING((1 1 1,0 0 0.5, -1 1 1),(1 1
0.5,0 0 0.5, -1 1 0.5, 1 1 0.5) ))));
st_asewkt
    
```

```
-----
MULTIPOINT((-1 1 1), (1 1 0.75))
```

**ST\_AsText**, **ST\_ExteriorRing**, **ST\_MakePolygon**

### 8.4.3 ST\_BoundingDiagonal

**ST\_BoundingDiagonal** — Returns the bounding diagonal of a geometry. The bounding diagonal is the shortest line segment that connects the two most distant vertices of the geometry.

#### Synopsis

geometry **ST\_BoundingDiagonal**(geometry geom, boolean fits=false);

**ST\_BoundingDiagonal**(geometry geom, boolean fits=false);

ST\_BoundingDiagonal(geom, fits) — Returns the bounding diagonal of a geometry. The bounding diagonal is the shortest line segment that connects the two most distant vertices of the geometry. If fits is true, the bounding diagonal is the shortest line segment that connects the two most distant vertices of the geometry, even if it is not a straight line segment.

**fits** — If true, the bounding diagonal is the shortest line segment that connects the two most distant vertices of the geometry, even if it is not a straight line segment. If false, the bounding diagonal is the shortest straight line segment that connects the two most distant vertices of the geometry.

**SRID** — The SRID of the geometry. If not specified, the SRID of the geometry is used.

#### Note



The bounding diagonal of a geometry is the shortest line segment that connects the two most distant vertices of the geometry. If the geometry is a polygon, the bounding diagonal is the shortest line segment that connects the two most distant vertices of the polygon, even if it is not a straight line segment.

2.2.0 The bounding diagonal of a geometry is the shortest line segment that connects the two most distant vertices of the geometry.

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

**Example 1**

```
-- 3D point with 10m buffer
-- 3D point with 10m buffer
SELECT ST_X(ST_StartPoint(ST_BoundingDiagonal(
  ST_Buffer(ST_MakePoint(0,0),10)
)));
-----
-10
```

**Example 2**

[ST\\_StartPoint](#), [ST\\_EndPoint](#), [ST\\_X](#), [ST\\_Y](#), [ST\\_Z](#), [ST\\_M](#), &&&

## 8.4.4 ST\_CoordDim

**ST\_CoordDim** — ST\_Geometry

### Synopsis

integer **ST\_CoordDim**(geometry geomA);

**Example 1**

ST\_Geometry  
 ST\_CoordDim('CIRCULARSTRING(1 2 3, 1 3 4, 5 6 7, 8 9 10, 11 12 13)');  
 3  
 ST\_CoordDim(ST\_Point(1,2));  
 2

- ✓ This method implements the [OGC Simple Features Implementation Specification for SQL 1.1](#).
- ✓ This method implements the SQL/MM specification. SQL-MM 3: 5.1.3
- ✓ This method supports Circular Strings and Curves
- ✓ This function supports 3d and will not drop the z-index.
- ✓ This function supports Polyhedral surfaces.
- ✓ This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).

**Example 2**

```
SELECT ST_CoordDim('CIRCULARSTRING(1 2 3, 1 3 4, 5 6 7, 8 9 10, 11 12 13)');
-- 3
3

SELECT ST_CoordDim(ST_Point(1,2));
-- 2
2
```

**ST\_NDims**

**ST\_NDims**

## 8.4.5 ST\_Dimension

**ST\_Dimension** — Returns the dimension of the geometry.

### Synopsis

integer **ST\_Dimension**(geometry g);

**ST\_Dimension**

OGC 2.1.1.1 POINT (0 0), LINESTRING (1 1, 2 2), POLYGON ((0 0, 1 0, 1 1, 0 1, 0 0)), GEOMETRYCOLLECTION (POINT (0 0), LINESTRING (1 1, 2 2), POLYGON ((0 0, 1 0, 1 1, 0 1, 0 0)), GEOMETRYCOLLECTION (POINT (0 0), LINESTRING (1 1, 2 2), POLYGON ((0 0, 1 0, 1 1, 0 1, 0 0)))



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

2.0.0 TIN (0 0, 1 0, 1 1, 0 1, 0 0)



### Note

2.0.0 TIN (0 0, 1 0, 1 1, 0 1, 0 0)



This function supports Polyhedral surfaces.



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

**ST\_Dump**

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

**ST\_Dump**

**ST\_Dump**

## 8.4.6 ST\_Dump

**ST\_Dump** — Returns a set of geometry\_dump rows for the components of a geometry.

## Synopsis

geometry **ST\_Envelope**(geometry g1);

### Notes

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

For an atomic geometry type (POINT,LINestring,POLYGON) a single record is returned with an empty *path* array and the input geometry as *geom*. For a collection or multi-geometry a record is returned for each of the collection components, and the *path* denotes the position of the component inside the collection.

ST\_Dump is useful for expanding geometries. It is the inverse of a **ST\_GeomCollFromText** / GROUP BY, in that it creates new rows. For example it can be used to expand MULTIPOLYGONS into POLYGONS.

2.0.0; 2.0.0; TIN; Availability: PostGIS 1.0.0RC1. Requires PostgreSQL 7.3 or higher.

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

### Note



1.3.4; 2.0.0; TIN; Availability: PostGIS 1.0.0RC1. Requires PostgreSQL 7.3 or higher.



This method supports Circular Strings and Curves



This function supports Polyhedral surfaces.



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



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

### Example

```
SELECT sometable.field1, sometable.field1,
       (ST_Dump(sometable.geom)).geom AS geom
FROM sometable;

-- Break a compound curve into its constituent linestrings and circularstrings
SELECT ST_AsEWKT(a.geom), ST_HasArc(a.geom)
FROM ( SELECT (ST_Dump(p_geom)).geom AS geom
       FROM (SELECT ST_GeomFromEWKT('COMPOUNDCURVE (CIRCULARSTRING(0 0, 1 1, 1 0), (1 0, 0
       1))') AS p_geom) AS b
       ) AS a;
       st_asewkt          | st_hasarc
-----+-----
CIRCULARSTRING(0 0,1 1,1 0) | t
LINESTRING(1 0,0 1)       | f
(2 rows)
```

**geometry\_dump, ST\_GeomFromEWKT, ST\_Dump, ST\_GeometryN, ST\_NumGeometries**

```
-- geometry_dump, ST_GeomFromEWKT, ST_Dump, ST_GeometryN, ST_NumGeometries
-- geometry_dump, ST_GeomFromEWKT, ST_Dump, ST_GeometryN, ST_NumGeometries; ←
-- geometry_dump, ST_GeomFromEWKT, ST_Dump, ST_GeometryN, ST_NumGeometries;
SELECT ST_AsEWKT(ST_GeometryN(p_geom,3)) AS geom_ewkt
FROM (SELECT ST_GeomFromEWKT('POLYHEDRALSURFACE(
((0 0 0, 0 0 1, 0 1 1, 0 1 0, 0 0 0)),
((0 0 0, 0 1 0, 1 1 0, 1 0 0, 0 0 0)),
((0 0 0, 1 0 0, 1 0 1, 0 0 1, 0 0 0)),
((1 1 0, 1 1 1, 1 0 1, 1 0 0, 1 1 0)),
((0 1 0, 0 1 1, 1 1 1, 1 1 0, 0 1 0)),
((0 0 1, 1 0 1, 1 1 1, 0 1 1, 0 0 1))
)') AS p_geom ) AS a;

          geom_ewkt
-----
POLYGON((0 0 0,1 0 0,1 0 1,0 0 1,0 0 0))
```

```
-- TIN --
SELECT ST_AsEWKT(ST_GeometryN(geom,2)) as wkt
FROM
  (SELECT
    ST_GeomFromEWKT('TIN (((
      0 0 0,
      0 0 1,
      0 1 0,
      0 0 0
    )), ((
      0 0 0,
      0 1 0,
      1 1 0,
      0 0 0
    ))) AS geom
  ) AS g;
-- geometry_dump, ST_GeomFromEWKT, ST_Dump, ST_GeometryN, ST_NumGeometries;
          wkt
-----
TRIANGLE((0 0 0,0 1 0,1 1 0,0 0 0))
```

**geometry\_dump, ST\_GeomFromEWKT, ST\_Dump, ST\_GeometryN, ST\_NumGeometries**

**geometry\_dump, ST\_GeomFromEWKT, ST\_Dump, ST\_GeometryN, ST\_NumGeometries**

## 8.4.7 ST\_NumPoints

**ST\_NumPoints** — Returns the number of points in the geometry. If the geometry is a **POINT**, it returns 1. If the geometry is a **LINESTRING**, it returns the number of vertices. If the geometry is a **POLYGON**, it returns the number of vertices in the outer boundary. If the geometry is a **MULTIPOINT**, it returns the number of points. If the geometry is a **MULTILINESTRING**, it returns the sum of the number of vertices in all the lines. If the geometry is a **MULTIPOLYGON**, it returns the sum of the number of vertices in all the polygons.

### Synopsis

```
geometry ST_NumPoints( geometry geom );
```

**ST\_Dump**

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

- the `geom` field POINTs represent the coordinates of the supplied geometry.
- the `path` field (an `integer[]`) is an index enumerating the coordinate positions in the elements of the supplied geometry. The indices are 1-based. For example, for a `LINestring` the paths are `{i}` where `i` is the `n`th coordinate in the `LINestring`. For a `POLYGON` the paths are `{i, j}` where `i` is the ring number (1 is outer; inner rings follow) and `j` is the coordinate position in the ring.

To obtain a single geometry containing the coordinates use `ST_Points`.

Enhanced: 2.1.0 Faster speed. Reimplemented as native-C.

2.0.0 `ST_Dump` and `ST_DumpPoints`: 2.0.0 `ST_Dump` and `ST_DumpPoints` support `POLYGON` with holes.

1.2.2 `ST_Dump` and `ST_DumpPoints`: 1.2.2 `ST_Dump` and `ST_DumpPoints` support `POLYGON` with holes.



This method supports Circular Strings and Curves



This function supports Polyhedral surfaces.



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



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

**Classic Explode a Table of LineStrings into nodes**

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

**&#xd45c;&#xc900; &#xc608;&#xc2dc;**



```
SELECT path, ST_AsText(geom)
FROM (
  SELECT (ST_DumpPoints(g.geom)).*
  FROM
    (SELECT
      'GEOMETRYCOLLECTION(
        POINT ( 0 1 ),
        LINESTRING ( 0 3, 3 4 ),
        POLYGON (( 2 0, 2 3, 0 2, 2 0 )),
        POLYGON (( 3 0, 3 3, 6 3, 6 0, 3 0 ),
          ( 5 1, 4 2, 5 2, 5 1 )),
        MULTIPOLYGON (
          (( 0 5, 0 8, 4 8, 4 5, 0 5 ),
            ( 1 6, 3 6, 2 7, 1 6 )),
            (( 5 4, 5 8, 6 7, 5 4 ))
          )
      )'::geometry AS geom
    ) AS g
  ) j;
```

| path      | st_astext  |
|-----------|------------|
| {1,1}     | POINT(0 1) |
| {2,1}     | POINT(0 3) |
| {2,2}     | POINT(3 4) |
| {3,1,1}   | POINT(2 0) |
| {3,1,2}   | POINT(2 3) |
| {3,1,3}   | POINT(0 2) |
| {3,1,4}   | POINT(2 0) |
| {4,1,1}   | POINT(3 0) |
| {4,1,2}   | POINT(3 3) |
| {4,1,3}   | POINT(6 3) |
| {4,1,4}   | POINT(6 0) |
| {4,1,5}   | POINT(3 0) |
| {4,2,1}   | POINT(5 1) |
| {4,2,2}   | POINT(4 2) |
| {4,2,3}   | POINT(5 2) |
| {4,2,4}   | POINT(5 1) |
| {5,1,1,1} | POINT(0 5) |
| {5,1,1,2} | POINT(0 8) |
| {5,1,1,3} | POINT(4 8) |

```

{5,1,1,4} | POINT(4 5)
{5,1,1,5} | POINT(0 5)
{5,1,2,1} | POINT(1 6)
{5,1,2,2} | POINT(3 6)
{5,1,2,3} | POINT(2 7)
{5,1,2,4} | POINT(1 6)
{5,2,1,1} | POINT(5 4)
{5,2,1,2} | POINT(5 8)
{5,2,1,3} | POINT(6 7)
{5,2,1,4} | POINT(5 4)
(29 rows)

```

### Polyhedral surface cube; TIN

```

-- Polyhedral surface cube --
SELECT (g.gdump).path, ST_AsEWKT((g.gdump).geom) as wkt
FROM
  (SELECT
    ST_DumpPoints(ST_GeomFromEWKT('POLYHEDRALSURFACE( ((0 0 0, 0 0 1, 0 1 1, 0 1 0, 0 0 ←
    0)),
    ((0 0 0, 0 1 0, 1 1 0, 1 0 0, 0 0 0)), ((0 0 0, 1 0 0, 1 0 1, 0 0 1, 0 0 0)),
    ((1 1 0, 1 1 1, 1 0 1, 1 0 0, 1 1 0)),
    ((0 1 0, 0 1 1, 1 1 1, 1 1 0, 0 1 0)), ((0 0 1, 1 0 1, 1 1 1, 0 1 1, 0 0 1)) )' ) AS gdump
  ) AS g;
-- result --
path | wkt
-----+-----
{1,1,1} | POINT(0 0 0)
{1,1,2} | POINT(0 0 1)
{1,1,3} | POINT(0 1 1)
{1,1,4} | POINT(0 1 0)
{1,1,5} | POINT(0 0 0)
{2,1,1} | POINT(0 0 0)
{2,1,2} | POINT(0 1 0)
{2,1,3} | POINT(1 1 0)
{2,1,4} | POINT(1 0 0)
{2,1,5} | POINT(0 0 0)
{3,1,1} | POINT(0 0 0)
{3,1,2} | POINT(1 0 0)
{3,1,3} | POINT(1 0 1)
{3,1,4} | POINT(0 0 1)
{3,1,5} | POINT(0 0 0)
{4,1,1} | POINT(1 1 0)
{4,1,2} | POINT(1 1 1)
{4,1,3} | POINT(1 0 1)
{4,1,4} | POINT(1 0 0)
{4,1,5} | POINT(1 1 0)
{5,1,1} | POINT(0 1 0)
{5,1,2} | POINT(0 1 1)
{5,1,3} | POINT(1 1 1)
{5,1,4} | POINT(1 1 0)
{5,1,5} | POINT(0 1 0)
{6,1,1} | POINT(0 0 1)
{6,1,2} | POINT(1 0 1)
{6,1,3} | POINT(1 1 1)
{6,1,4} | POINT(0 1 1)
{6,1,5} | POINT(0 0 1)
(30 rows)

```

```
-- TIN --
```

```

SELECT ST_AsEWKT(ST_GeometryN(geom,2)) as wkt
FROM
  (SELECT
    ST_GeomFromEWKT('TIN (((
      0 0 0,
      0 0 1,
      0 1 0,
      0 0 0
    )), ((
      0 0 0,
      0 1 0,
      1 1 0,
      0 0 0
    ))
  )') AS geom
) AS g;
-- \s; --
      wkt
-----
TRIANGLE((0 0 0,0 1 0,1 1 0,0 0 0))

```

```

-- TIN --
SELECT ST_AsEWKT(ST_GeometryN(geom,2)) as wkt
FROM
  (SELECT
    ST_GeomFromEWKT('TIN (((
      0 0 0,
      0 0 1,
      0 1 0,
      0 0 0
    )), ((
      0 0 0,
      0 1 0,
      1 1 0,
      0 0 0
    ))
  )') AS geom
) AS g;
-- \s; --
      wkt
-----
TRIANGLE((0 0 0,0 1 0,1 1 0,0 0 0))

```

**&#x38;&#x3e;**

[geometry\\_dump](#), [ST\\_GeomFromEWKT](#), [ST\\_Dump](#), [ST\\_GeometryN](#), [ST\\_NumGeometries](#)

## 8.4.8 ST\_NumPoints

`ST_NumPoints` — [&#xb3c4;&#xd615;&#xc758;](#) [&#xb0b4;&#xc6a9;&#xc744;](#) [&#xc694;&#xc57d;&#xd55c;](#) [&#xd14d;&#xc2a4;&#x](#)  
[&#xbc18;&#xd658;&#xd569;](#) [&#xb2c8;&#xb2e4;](#).

### Synopsis

geometry **ST\_Points**( geometry geom );

**ST\_DumpSegments**

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

- `LINestring` returns `TRUE` if the segment is a line segment, otherwise `FALSE`.
- the `path` field (an `integer[]`) is an index enumerating the segment start point positions in the elements of the supplied geometry. The indices are 1-based. For example, for a `LINestring` the paths are `{i}` where `i` is the `n`th segment start point in the `LINestring`. For a `POLYGON` the paths are `{i, j}` where `i` is the ring number (1 is outer; inner rings follow) and `j` is the segment start point position in the ring.

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



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



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

**ST\_AsText**

```
SELECT path, ST_AsText(geom)
FROM (
  SELECT (ST_DumpSegments(g.geom)).*
  FROM (SELECT 'GEOMETRYCOLLECTION(
    LINestring(1 1, 3 3, 4 4),
    POLYGON((5 5, 6 6, 7 7, 5 5))
  )'::geometry AS geom
  ) AS g
) j;
```

| path    | st_astext           |
|---------|---------------------|
| {1,1}   | LINestring(1 1,3 3) |
| {1,2}   | LINestring(3 3,4 4) |
| {2,1,1} | LINestring(5 5,6 6) |
| {2,1,2} | LINestring(6 6,7 7) |
| {2,1,3} | LINestring(7 7,5 5) |

(5 rows)

**ST\_GeomFromEWKT**

```
-- TIN --
SELECT ST_AsEWKT(ST_GeometryN(geom,2)) as wkt
FROM
  (SELECT
    ST_GeomFromEWKT('TIN (((
      0 0 0,
      0 0 1,
      0 1 0,
      0 0 0
    ))), ((
      0 0 0,
      0 1 0,
      1 1 0,
      0 0 0
    )))
```







**ST\_Envelope**

**ST\_PointN, ST\_StartPoint**

### 8.4.11 ST\_Envelope

**ST\_Envelope** — Returns the bounding box of the geometry. The bounding box is a rectangle that encloses the geometry. The bounding box is defined by the minimum and maximum X and Y coordinates of the geometry. The bounding box is returned as a geometry of type `LINESTRING`. The bounding box is defined by the minimum and maximum X and Y coordinates of the geometry. The bounding box is returned as a geometry of type `LINESTRING`.

#### Synopsis

geometry **ST\_Envelope**(geometry g1);

**ST\_Envelope**

geometry **ST\_Envelope**(geometry g1);  
 Returns the bounding box of the geometry. The bounding box is a rectangle that encloses the geometry. The bounding box is defined by the minimum and maximum X and Y coordinates of the geometry. The bounding box is returned as a geometry of type `LINESTRING`. The bounding box is defined by the minimum and maximum X and Y coordinates of the geometry. The bounding box is returned as a geometry of type `LINESTRING`.

geometry **ST\_Envelope**(geometry g1);  
 Returns the bounding box of the geometry. The bounding box is a rectangle that encloses the geometry. The bounding box is defined by the minimum and maximum X and Y coordinates of the geometry. The bounding box is returned as a geometry of type `LINESTRING`. The bounding box is defined by the minimum and maximum X and Y coordinates of the geometry. The bounding box is returned as a geometry of type `LINESTRING`.

1.5.0 and later: geometry **ST\_Envelope**(geometry g1);  
 Returns the bounding box of the geometry. The bounding box is a rectangle that encloses the geometry. The bounding box is defined by the minimum and maximum X and Y coordinates of the geometry. The bounding box is returned as a geometry of type `LINESTRING`. The bounding box is defined by the minimum and maximum X and Y coordinates of the geometry. The bounding box is returned as a geometry of type `LINESTRING`.



This method implements the [OGC Simple Features Implementation Specification for SQL 1.1](#). s2.1.1.1



This method implements the SQL/MM specification. SQL-MM 3: 5.1.19

**ST\_Envelope**

```
SELECT ST_AsText(ST_Envelope('POINT(1 3)::geometry'));
 st_astext
```

```
-----
POINT(1 3)
(1 row)
```

```
SELECT ST_AsText(ST_Envelope('LINESTRING(0 0, 1 3)::geometry'));
 st_astext
```

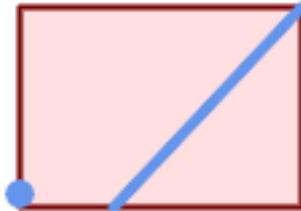
```
-----
POLYGON((0 0,0 3,1 3,1 0,0 0))
(1 row)
```

```
SELECT ST_AsText(ST_Envelope('POLYGON((0 0, 0 1, 1.0000001 1, 1.0000001 0, 0 0))::geometry ←
));
 st_astext
```

```
-----
POLYGON((0 0,0 1,1.00000011920929 1,1.00000011920929 0,0 0))
```

```
(1 row)
SELECT ST_AsText(ST_Envelope('POLYGON((0 0, 0 1, 1.0000000001 1, 1.0000000001 0, 0 0))'::
  geometry));
          st_astext
-----
POLYGON((0 0,0 1,1.00000011920929 1,1.00000011920929 0,0 0))
(1 row)

SELECT Box3D(geom), Box2D(geom), ST_AsText(ST_Envelope(geom)) As envelopewkt
  FROM (SELECT 'POLYGON((0 0, 0 1000012333334.34545678, 1.0000001 1, 1.0000001 0, 0
  0))'::geometry As geom) As foo;
```



*Envelope of a point and linestring.*

```
SELECT ST_AsText(ST_Envelope(
  ST_Collect(
    ST_GeomFromText('LINESTRING(55 75,125 150)'),
    ST_Point(20, 80)
  )) As wktenv);
wktenv
-----
POLYGON((20 75,20 150,125 150,125 75,20 75))
```

**Envelope**

**Box2D, Box3D, ST\_OrientedEnvelope**

#### 8.4.12 ST\_ExteriorRing

**ST\_ExteriorRing** — Returns the exterior ring of a polygon. If the polygon is not closed, the function will return the exterior ring of the convex hull of the polygon.

##### Synopsis

geometry **ST\_ExteriorRing**(geometry a\_polygon);

**ST\_ExteriorRing**

POLYGON ((0 0 1, 1 1 1, 1 2 1, 1 1 1, 0 0 1))

**Note**

ST\_ExteriorRing(geometry) returns the exterior ring of a polygon. If the geometry is not a polygon, it returns NULL.

- ✔ This method implements the [OGC Simple Features Implementation Specification for SQL 1.1](#), 2.1.5.1
- ✔ This method implements the SQL/MM specification. SQL-MM 3: 8.2.3, 8.3.3
- ✔ This function supports 3d and will not drop the z-index.

**ST\_ExteriorRings**

```
-- ST_ExteriorRings
SELECT gid, ST_ExteriorRing(the_geom) AS ering
FROM sometable;

-- ST_ExteriorRings (3D)
SELECT gid, ST_ExteriorRing(the_geom) AS ering
FROM (SELECT gid, (ST_Dump(the_geom)).geom As the_geom
      FROM sometable) As foo
GROUP BY gid;

--3d Example
SELECT ST_AsEWKT(
  ST_ExteriorRing(
    ST_GeomFromEWKT('POLYGON((0 0 1, 1 1 1, 1 2 1, 1 1 1, 0 0 1))')
  )
);

st_asewkt
-----
LINESTRING(0 0 1,1 1 1,1 2 1,1 1 1,0 0 1)
```

**ST\_InteriorRingN**

[ST\\_InteriorRingN](#), [ST\\_Boundary](#), [ST\\_NumInteriorRings](#)

**8.4.13 ST\_GeometryN**

**ST\_GeometryN** — ST\_Geometry



```

WHERE n <= ST_NumGeometries(geom);

n |          geomewkt
---+-----
1 | POINT(1 2 7)
2 | POINT(3 4 7)
3 | POINT(5 6 7)
4 | POINT(8 9 10)
1 | CIRCULARSTRING(2.5 2.5,4.5 2.5,3.5 3.5)
2 | LINESTRING(10 11,12 11)

--Extracting all geometries (useful when you want to assign an id)
SELECT gid, n, ST_GeometryN(geom, n)
FROM sometable CROSS JOIN generate_series(1,100) n
WHERE n <= ST_NumGeometries(geom);

```

### Geometries: TIN

```

-- Polygon
-- Geometries: TIN
SELECT ST_AsEWKT(ST_GeometryN(p_geom,3)) As geom_ewkt
FROM (SELECT ST_GeomFromEWKT('POLYHEDRALSURFACE(
((0 0 0, 0 0 1, 0 1 1, 0 1 0, 0 0 0)),
((0 0 0, 0 1 0, 1 1 0, 1 0 0, 0 0 0)),
((0 0 0, 1 0 0, 1 0 1, 0 0 1, 0 0 0)),
((1 1 0, 1 1 1, 1 0 1, 1 0 0, 1 1 0)),
((0 1 0, 0 1 1, 1 1 1, 1 1 0, 0 1 0)),
((0 0 1, 1 0 1, 1 1 1, 0 1 1, 0 0 1))
)') AS p_geom ) AS a;

          geom_ewkt
-----
POLYGON((0 0 0,1 0 0,1 0 1,0 0 1,0 0 0))

```

```

-- TIN --
SELECT ST_AsEWKT(ST_GeometryN(geom,2)) as wkt
FROM
  (SELECT
    ST_GeomFromEWKT('TIN (((
      0 0 0,
      0 0 1,
      0 1 0,
      0 0 0
    )), ((
      0 0 0,
      0 1 0,
      1 1 0,
      0 0 0
    )))') AS geom
  ) AS g;
-- Geometries: TIN
          wkt
-----
TRIANGLE((0 0 0,0 1 0,1 1 0,0 0 0))

```



```

SELECT ST_GeometryType(geom) as result
FROM
  (SELECT
    ST_GeomFromEWKT('TIN (((
      0 0 0,
      0 0 1,
      0 1 0,
      0 0 0
    )), ((
      0 0 0,
      0 1 0,
      1 1 0,
      0 0 0
    )))
  ) AS g;
result
-----
ST_Tin

```

**8.4.14 ST\_IsEmpty**

[ST\\_IsEmpty](#)

### 8.4.15 ST\_HasArc

ST\_HasArc — Tests if a geometry contains a circular arc

#### Synopsis

boolean **ST\_IsEmpty**(geometry geomA);

**8.4.15.1**

[ST\\_HasArc](#)

1.2.2 [ST\\_HasArc](#)



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



This method supports Circular Strings and Curves

**8.4.15.2**

```

SELECT ST_HasArc(ST_Collect('LINESTRING(1 2, 3 4, 5 6)', 'CIRCULARSTRING(1 1, 2 3, 4 5, 6 7, 5 6)'));
          st_hasarc
          -----
          t

```

**8.4.15.3**

[ST\\_CurveToLine](#), [ST\\_PointN](#)

## 8.4.16 ST\_InteriorRingN

`ST_InteriorRingN` — Returns the *n*th interior ring of a polygon. If *n* is 0, the outer boundary is returned. If *n* is 1, the first interior ring is returned, and so on. If *n* is greater than the number of interior rings, NULL is returned.

### Synopsis

geometry `ST_InteriorRingN`(geometry a\_polygon, integer n);

**Note:**

For a given polygon, the rings are ordered by their area. The outer boundary is the ring with the largest area. The first interior ring is the ring with the next largest area, and so on. The area of a ring is the area of the polygon formed by the ring, minus the area of the polygon formed by the ring immediately inside it. If the ring is the outer boundary, the area of the polygon immediately inside it is zero.



#### Note

For a given polygon, the rings are ordered by their area. The outer boundary is the ring with the largest area. The first interior ring is the ring with the next largest area, and so on. The area of a ring is the area of the polygon formed by the ring, minus the area of the polygon formed by the ring immediately inside it. If the ring is the outer boundary, the area of the polygon immediately inside it is zero.



This method implements the [OGC Simple Features Implementation Specification for SQL 1.1](#).



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



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

**Example:**

```
SELECT ST_AsText(ST_InteriorRingN(the_geom, 1)) As the_geom
FROM (SELECT ST_BuildArea(
    ST_Collect(ST_Buffer(ST_Point(1,2), 20,3),
              ST_Buffer(ST_Point(1, 2), 10,3))) As the_geom
) as foo
```

**See also:**

[ST\\_ExteriorRing](#), [ST\\_M](#), [ST\\_X](#), [ST\\_Y](#), [ST\\_ZMax](#), [ST\\_ZMin](#)

## 8.4.17 ST\_IsClosed

`ST_IsClosed` — Returns true if the geometry is a closed line string. If the geometry is not a closed line string, false is returned. A closed line string is a line string where the start and end points are the same. If the geometry is a closed line string, true is returned. If the geometry is not a closed line string, false is returned.

### Synopsis

boolean `ST_IsClosed`(geometry g);

**ST\_IsClosed**

LINESTRING (0 0, 1 1)::geometry;  
 TRUE  
 LINESTRING (0 0, 0 1, 1 1, 0 0)::geometry;  
 TRUE  
 MULTILINESTRING ((0 0, 0 1, 1 1, 0 0), (0 0, 1 1))::geometry;  
 TRUE  
 POINT (0 0)::geometry;  
 TRUE  
 MULTIPOINT ((0 0), (1 1))::geometry;  
 TRUE



This method implements the [OGC Simple Features Implementation Specification for SQL 1.1](#).



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

**Note**

SQL-MM 3: 7.1.5, 9.3.3  
 ST\_IsClosed(NULL) returns NULL  
 PostGIS 3.3.6dev: ST\_IsClosed(NULL) returns TRUE



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



This method supports Circular Strings and Curves

LINESTRING (0 0, 1 1)::geometry;  
 TRUE  
 CURVEPOLY (0 0, 1 1)::geometry;  
 TRUE



This function supports Polyhedral surfaces.

**ST\_IsClosed**

```

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

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

postgis=# SELECT ST_IsClosed('MULTILINESTRING((0 0, 0 1, 1 1, 0 0),(0 0, 1 1))::geometry');
 st_isclosed
-----
 f
(1 row)

postgis=# SELECT ST_IsClosed('POINT(0 0)::geometry');
 st_isclosed
-----
 t
(1 row)

postgis=# SELECT ST_IsClosed('MULTIPOINT((0 0), (1 1))::geometry');
 st_isclosed
-----
 t
(1 row)

```

**ST\_IsClosed**

```
-- ST_IsClosed --
SELECT ST_IsClosed(ST_GeomFromEWKT('POLYHEDRALSURFACE( ((0 0 0, 0 0 1, 0 1
1, 0 1 0, 0 0 0)),
((0 0 0, 0 1 0, 1 1 0, 1 0 0, 0 0 0)), ((0 0 0, 1 0 0, 1 0 1, 0 0 1, 0 0 0)
),
((1 1 0, 1 1 1, 1 0 1, 1 0 0, 1 1 0)),
((0 1 0, 0 1 1, 1 1 1, 1 1 0, 0 1 0)), ((0 0 1, 1 0 1, 1 1 1, 0 1 1, 0 0 1)
) )'));

st_isclosed
-----
t

-- ST_IsClosed --
SELECT ST_IsClosed(ST_GeomFromEWKT('POLYHEDRALSURFACE( ((0 0 0, 0 0 1, 0 1 1, 0 1 0, 0 0
0)),
((0 0 0, 0 1 0, 1 1 0, 1 0 0, 0 0 0)), ((0 0 0, 1 0 0, 1 0 1, 0 0 1, 0 0 0)
),
((1 1 0, 1 1 1, 1 0 1, 1 0 0, 1 1 0)),
((0 1 0, 0 1 1, 1 1 1, 1 1 0, 0 1 0)) )'));

st_isclosed
-----
f
```

**ST\_IsRing****ST\_IsRing****8.4.18 ST\_IsCollection**

**ST\_IsCollection** — Returns true if the geometry is a collection of points, lines, or surfaces.

**Synopsis**

boolean **ST\_IsCollection**(geometry g);

**ST\_IsCollection**

Returns true if the geometry is a collection of points, lines, or surfaces. Returns false if the geometry is a single point, line, or surface.

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

**Note**

This function supports 3D and will not drop the z-index. This method supports Circular Strings and Curves.



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



This method supports Circular Strings and Curves

**ST\_IsCollection**

```

postgis=# SELECT ST_IsCollection('LINESTRING(0 0, 1 1)::geometry');
 st_iscollection
-----
 f
(1 row)

postgis=# SELECT ST_IsCollection('MULTIPOINT EMPTY)::geometry');
 st_iscollection
-----
 t
(1 row)

postgis=# SELECT ST_IsCollection('MULTIPOINT((0 0))::geometry');
 st_iscollection
-----
 t
(1 row)

postgis=# SELECT ST_IsCollection('MULTIPOINT((0 0), (42 42))::geometry');
 st_iscollection
-----
 t
(1 row)

postgis=# SELECT ST_IsCollection('GEOMETRYCOLLECTION(POINT(0 0))::geometry');
 st_iscollection
-----
 t
(1 row)

```

**ST\_NumGeometries****ST\_NumGeometries****8.4.19 ST\_IsEmpty**

**ST\_IsEmpty** — Tests if a geometry is empty.

**Synopsis**

boolean **ST\_IsEmpty**(geometry geomA);

**ST\_IsEmpty**

ST\_IsEmpty(geometry) RETURNS BOOLEAN  
 TRUE if the geometry is empty, FALSE otherwise.  
 Note: ST\_IsEmpty(NULL) returns NULL.

**Note**

SQL-MM3: ST\_IsEmpty(NULL) returns NULL.  
 PostGIS: ST\_IsEmpty(NULL) returns NULL.



This method implements the [OGC Simple Features Implementation Specification for SQL 1.1](#), s2.1.1.1



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



This method supports Circular Strings and Curves

**Warning**

ST\_IsEmpty(NULL) returns NULL.  
 PostGIS 2.0.0: ST\_IsEmpty(NULL) returns NULL.  
 SQL/MM: ST\_IsEmpty(NULL) returns NULL.

**Examples**

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

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

SELECT ST_IsEmpty(ST_GeomFromText('POLYGON((1 2, 3 4, 5 6, 1 2))'));
 st_isempty
-----
f
(1 row)

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

SELECT ST_IsEmpty(ST_GeomFromText('CIRCULARSTRING EMPTY'));
 st_isempty
-----
```

```
t
(1 row)
```

## 8.4.20 ST\_IsPolygonCCW

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

### Synopsis

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

### Parameters

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

Returns true if the geometry has no polygonal components.



#### Note

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



#### Note

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

2.2.0 [New](#) [Function](#) [Syntax](#) [Parameters](#) [Returns](#) [Examples](#) [Comments](#) [Source](#)



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



This function supports M coordinates.

### See Also

[ST\\_ForcePolygonCW](#) , [ST\\_ForcePolygonCCW](#) , [ST\\_IsPolygonCW](#)

## 8.4.21 ST\_IsPolygonCW

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

### Synopsis

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

**Notes**

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

Returns true if the geometry has no polygonal components.

**Note**

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

**Note**

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

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



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



This function supports M coordinates.

**Related Functions**

[ST\\_ForcePolygonCW](#) , [ST\\_ForcePolygonCCW](#) , [ST\\_IsPolygonCW](#)

**8.4.22 ST\_IsRing**

`ST_IsRing` — Tests if a `LineString` is closed and simple.

**Synopsis**

boolean `ST_IsRing`(geometry g);

**Notes**

`LINESTRING`; `ST_IsClosed`(`ST_StartPoint`((g)) ~ `ST_Endpoint`((g))) & `ST_IsSimple`((g)); (PostGIS 2.2.0; TRUE; `ST_IsSimple`((g));



This method implements the [OGC Simple Features Implementation Specification for SQL 1.1](#). 2.1.5.1



This method implements the SQL/MM specification. SQL-MM 3: 7.1.6

**Note**

SQL-MM; `ST_IsRing`(NULL) & `ST_IsRing`(`LINESTRING(0 0, 0 0)`); `ST_IsRing`(`LINESTRING(0 0, 0 0)`); `ST_IsRing`(`LINESTRING(0 0, 0 0)`); `ST_IsRing`(`LINESTRING(0 0, 0 0)`);

**ST\_IsSimple**

```

SELECT ST_IsRing(the_geom), ST_IsClosed(the_geom), ST_IsSimple(the_geom)
FROM (SELECT 'LINESTRING(0 0, 0 1, 1 1, 1 0, 0 0)::geometry AS the_geom) AS foo;
  st_isring | st_isclosed | st_issimple
-----+-----+-----
t          | t           | t
(1 row)

SELECT ST_IsRing(the_geom), ST_IsClosed(the_geom), ST_IsSimple(the_geom)
FROM (SELECT 'LINESTRING(0 0, 0 1, 1 0, 1 1, 0 0)::geometry AS the_geom) AS foo;
  st_isring | st_isclosed | st_issimple
-----+-----+-----
f          | t           | f
(1 row)

```

**ST\_IsSimple**

**ST\_IsSimple**, **ST\_IsClosed**, **ST\_IsSimple**, **ST\_StartPoint**, **ST\_EndPoint**

**8.4.23 ST\_IsSimple**

**ST\_IsSimple** — Returns true if the geometry is simple, that is, if it does not contain any self-intersections or any other topological anomalies that would prevent it from being a valid simple feature. The function returns true if the geometry is a point, a line, or a polygon that does not self-intersect. The function returns false if the geometry is a multi-point, a multi-line, or a multi-polygon that contains any self-intersections or other topological anomalies. The function returns true if the geometry is a point, a line, or a polygon that does not self-intersect. The function returns false if the geometry is a multi-point, a multi-line, or a multi-polygon that contains any self-intersections or other topological anomalies.

**Synopsis**

boolean **ST\_IsSimple**(geometry geomA);

**ST\_IsSimple**

Returns true if the geometry is simple, that is, if it does not contain any self-intersections or any other topological anomalies that would prevent it from being a valid simple feature. The function returns true if the geometry is a point, a line, or a polygon that does not self-intersect. The function returns false if the geometry is a multi-point, a multi-line, or a multi-polygon that contains any self-intersections or other topological anomalies. The function returns true if the geometry is a point, a line, or a polygon that does not self-intersect. The function returns false if the geometry is a multi-point, a multi-line, or a multi-polygon that contains any self-intersections or other topological anomalies. **"OpenGIS Simple Features Implementation Specification for SQL 1.1, s2.1.1.1 (Ensuring OpenGIS compliancy of geometries)"**

**Note**

SQL-MM **ST\_IsSimple**(NULL) returns true. PostGIS **ST\_IsSimple**(NULL) returns false.



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



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



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

**Example 1**

```
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)
```

**Example 2****ST\_IsValid****8.4.24 ST\_M**

ST\_M — Returns the M coordinate of a Point.

**Synopsis**

float ST\_M(geometry a\_point);

**Parameters**

**M**: M coordinate of the point. If the point is 2D, the M coordinate is NULL. If the point is 3D, the M coordinate is the z-index. If the point is 4D, the M coordinate is the fourth coordinate.

**Note**

The M coordinate is returned as a float. If the point is 2D, the M coordinate is NULL. If the point is 3D, the M coordinate is the z-index. If the point is 4D, the M coordinate is the fourth coordinate.

This method implements the [OGC Simple Features Implementation Specification for SQL 1.1](#).

This method implements the SQL/MM specification.



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

**Example 1**

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

**ST\_MemSize**

**ST\_GeomFromEWKT, ST\_X, ST\_Y, ST\_Z**

### 8.4.25 ST\_MemSize

ST\_MemSize — ST\_Geometry

#### Synopsis

integer ST\_NRings(geometry geomA);

**ST\_MemSize**

ST\_Geometry

This complements the PostgreSQL built-in [database object functions](#) `pg_column_size`, `pg_size_pretty`, `pg_relation_size`, `pg_total_relation_size`.

#### Note



`pg_relation_size` which gives the byte size of a table may return byte size lower than `ST_MemSize`. This is because `pg_relation_size` does not add toasted table contribution and large geometries are stored in TOAST tables.

`pg_total_relation_size` - includes, the table, the toasted tables, and the indexes.

`pg_column_size` returns how much space a geometry would take in a column considering compression, so may be lower than `ST_MemSize`



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



This method supports Circular Strings and Curves



This function supports Polyhedral surfaces.



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

Changed: 2.2.0 name changed to `ST_MemSize` to follow naming convention.

**ST\_MemSize**

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

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

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

73

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

## 8.4.26 ST\_NDims

**ST\_NDims** — ST\_Geometry

### Synopsis

integer **ST\_NDims**(geometry g1);

### ;

2D: (x,y), 3D: (x,y,z), 4D: (x,y,z,m), 5D: (x,y,z,m,s)



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

### ;

```
SELECT ST_NDims(ST_GeomFromText('POINT(1 1)')) As d2point,
       ST_NDims(ST_GeomFromEWKT('POINT(1 1 2)')) As d3point,
       ST_NDims(ST_GeomFromEWKT('POINTM(1 1 0.5)')) As d2pointm;
```

```

d2point | d3point | d2pointm
-----+-----+-----
      2 |        3 |        3
```

### ;

[ST\\_CoordDim](#), [ST\\_Dimension](#), [ST\\_GeomFromEWKT](#)

## 8.4.27 ST\_NPoints

**ST\_NPoints** — ST\_Geometry

### Synopsis

integer **ST\_NPoints**(geometry g1);

**ST\_NPoints**

ST\_NPoints(geom) — Returns the number of points in the geometry. If the geometry is a curve, the number of points is the number of vertices in the curve.

ST\_NPoints(geom, z\_index) — Returns the number of points in the geometry. If the geometry is a curve, the number of points is the number of vertices in the curve. If the geometry is a surface, the number of points is the number of vertices in the surface.

**Note**

1.3.4 ST\_NPoints(geom, z\_index) — Returns the number of points in the geometry. If the geometry is a curve, the number of points is the number of vertices in the curve. If the geometry is a surface, the number of points is the number of vertices in the surface.

- ✔ This function supports 3d and will not drop the z-index.
- ✔ This method supports Circular Strings and Curves
- ✔ This function supports Polyhedral surfaces.

**ST\_NPoints**

```
SELECT ST_NPoints(ST_GeomFromText('LINESTRING(77.29 29.07,77.42 29.26,77.27 29.31,77.29 29.07)'));
-- ST_NPoints(geom);
4

-- ST_NPoints(geom, z_index);
SELECT ST_NPoints(ST_GeomFromEWKT('LINESTRING(77.29 29.07 1,77.42 29.26 0,77.27 29.31 -1,77.29 29.07 3)'));
-- ST_NPoints(geom, z_index);
4
```

**ST\_NPoints****ST\_NumPoints****8.4.28 ST\_NRings**

ST\_NRings(geom) — Returns the number of rings in the geometry. If the geometry is a surface, the number of rings is the number of holes in the surface.

**Synopsis**

integer ST\_NRings(geometry geomA);

**ST\_NumInteriorRings**

NumInteriorRings — Returns the number of interior rings of a polygon. If the polygon is a simple polygon, the result is 0. If the polygon is a multi-part polygon, the result is the number of interior rings of the first polygon. If the polygon is a circular string, the result is the number of interior rings of the circular string. If the polygon is a curve, the result is the number of interior rings of the curve.



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



This method supports Circular Strings and Curves

**SQL/MM specification**

```
SELECT ST_NRings(the_geom) As Nrings, ST_NumInteriorRings(the_geom) As ninterrings
FROM (SELECT ST_GeomFromText('POLYGON((1 2, 3 4, 5 5, 6, 1 2))') As the_geom) As foo;
```

| nrings | ninterrings |
|--------|-------------|
| 1      | 0           |

(1 row)

**ST\_NumInteriorRings****ST\_NumInteriorRings****8.4.29 ST\_NumGeometries**

ST\_NumGeometries — Returns the number of geometries in a multi-part geometry. If the geometry is a simple polygon, the result is 1. If the geometry is a multi-part polygon, the result is the number of polygons. If the geometry is a circular string, the result is the number of circular strings. If the geometry is a curve, the result is the number of curves.

**Synopsis**

integer **ST\_NumGeometries**(geometry geom);

**SQL/MM specification**

ST\_NumGeometries — Returns the number of geometries in a multi-part geometry. If the geometry is a simple polygon, the result is 1. If the geometry is a multi-part polygon, the result is the number of polygons. If the geometry is a circular string, the result is the number of circular strings. If the geometry is a curve, the result is the number of curves.

2.0.0 &#x0ac; &#xd56d;: 2.0.0 &#xbc84; &#xc804; &#xbd80; &#xd130; &#xb2e4; &#xba74; &#xccb4; &#xd45c; &#xba0c; &#xc0bc; &#xc01; &#xd615; &#xbc0f; TIN &#xc744; &#xc9c0; &#xc6d0; &#xd558; &#xae30; &#xc2dc; &#xc791; &#xd588; &#xc2b5;

&#xbcc0; &#xacbd; &#xc0ac; &#xd56d;: 2.0.0 &#xbbf8; &#xb9cc; &#xbc84; &#xc804; &#xc5d0; &#xc11c; &#xb3c4; &#xd615; &#xc778; &#xc9d1; &#xd569; &#xc774; &#xb098; &#xba40; &#xd2f0; &#xc720; &#xd615; &#xc774; &#xc544; &#xb2d0; &#xacbd; &#xc6b0; NULL &#xc744; &#xbc18; &#xd658; &#xd588; &#xc2b5; &#xb2c8; &#xb2e4;. 2.0.0 &#xbc84; &#xc804; &#xbd80; &#xd130; &#xd3f4; &#xb77c; &#xc778; &#xc2a4; &#xd2b8; &#xb9c1; &#xd3ec; &#xc778; &#xd2b8; &#xac19; &#xc740; &#xb2e8; &#xc77c; &#xb3c4; &#xb300; &#xd574; 1 &#xc744; &#xbc18; &#xd658; &#xd569; &#xb2c8; &#xb2e4;.



This method implements the SQL/MM specification. SQL-MM 3: 9.1.4

-  This function supports 3d and will not drop the z-index.
-  This function supports Polyhedral surfaces.
-  This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).

**Examples**

```
--Prior versions would have returned NULL for this -- in 2.0.0 this returns 1
SELECT ST_NumGeometries(ST_GeomFromText('LINESTRING(77.29 29.07,77.42 29.26,77.27 29.31,77.29 29.07)'));
--result
1

--Geometry Collection Example - multis count as one geom in a collection
SELECT ST_NumGeometries(ST_GeomFromEWKT('GEOMETRYCOLLECTION(MULTIPOINT((-2 3),(-2 2)),
LINESTRING(5 5 ,10 10),
POLYGON((-7 4.2,-7.1 5,-7.1 4.3,-7 4.2)))'));
--result
3
```

**ST\_GeometryN****ST\_GeometryN, ST\_Multi****8.4.30 ST\_NumInteriorRings**

**ST\_NumInteriorRings** — Returns the number of interior rings (holes) in a polygon. If the polygon is not a simple polygon, the function returns NULL.

**Synopsis**

integer **ST\_NumInteriorRings**(geometry a\_polygon);

**Examples**

```
-- Returns the number of interior rings (holes) in a polygon.
SELECT ST_NumInteriorRings(ST_GeomFromText('POLYGON((0 0,10 0,10 10,0 10,0 0),((2 2,8 2,8 8,2 8,2 2)))'));
--result
1
```



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

```
-- Returns the number of interior rings (holes) in a polygon.
SELECT ST_NumInteriorRings(ST_GeomFromText('POLYGON((0 0,10 0,10 10,0 10,0 0),((2 2,8 2,8 8,2 8,2 2)))'));
--result
1
```

**Examples**

```
-- Returns the number of interior rings (holes) in a polygon.
SELECT gid, field1, field2, ST_NumInteriorRings(the_geom) AS numholes
FROM sometable;

-- Returns the number of interior rings (holes) in a polygon.
```

```
--
SELECT gid, field1, field2, SUM(ST_NumInteriorRings(the_geom)) AS numholes
FROM (SELECT gid, field1, field2, (ST_Dump(the_geom)).geom As the_geom
      FROM sometable) As foo
GROUP BY gid, field1, field2;
```

**ST\_NumInteriorRing**

**ST\_NumInteriorRing, ST\_PointN**

### 8.4.31 ST\_NumInteriorRing

**ST\_NumInteriorRing** — Returns the number of interior rings (holes) in a polygon. The function returns 0 for a simple polygon and 1 for a polygon with one hole. The function returns NULL for a geometry that is not a polygon.

#### Synopsis

integer **ST\_NumInteriorRing**(geometry a\_polygon);

**ST\_NumInteriorRings**

**ST\_NumInteriorRings, ST\_PointN**

### 8.4.32 ST\_NumPatches

**ST\_NumPatches** — Returns the number of patches in a geometry. The function returns 0 for a simple polygon and 1 for a polygon with one hole. The function returns NULL for a geometry that is not a polygon.

#### Synopsis

integer **ST\_NumPatches**(geometry g1);

**ST\_NumPatches**

2.0.0 &#x2013; Returns the number of patches in a geometry. The function returns 0 for a simple polygon and 1 for a polygon with one hole. The function returns NULL for a geometry that is not a polygon.



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



This method implements the [OGC Simple Features Implementation Specification for SQL 1.1](#).



This method implements the SQL/MM specification. SQL-MM ISO/IEC 13249-3: 8.5



This function supports Polyhedral surfaces.



### 8.4.34 ST\_PatchN

ST\_PatchN — ST\_Geometry

#### Synopsis

geometry ST\_PatchN(geometry geomA, integer n);

**Notes:**

POLYHEDRALSURFACE, POLYHEDRALSURFACEM  
 1-  
 N  
 (&#x74;)&#x74; &#xbc18;&#xd658;&#xd569;&#xb2c8;&#xb2e4;. &#xadf8;  
 &#xc678;&#xc758; &#xacbd;&#xc6b0; NULL&#x74; &#xbc18;&#xd658;&#xd569;&#xb2c8;&#xb2e4;. &#xc774; &#xd568;&#x  
 &#xb2e4;&#xba74;&#xc4; &#xd45c;&#xba74;&#xc5d0; &#xb300;&#xd574; ST\_GeometryN&#xacfc; &#xb3d9;&#xc77c;&#xd55  
 &#xb2f5;&#xc744; &#xbc18;&#xd658;&#xd569;&#xb2c8;&#xb2e4;. ST\_GeometryN&#x74; &#xc774;&#xc6a9;&#xd558;&#xb2  
 &#xd3b8;&#xc774; &#xb354; &#xbe60;&#xb985;&#xb2c8;&#xb2e4;.



#### Note

1-  
 &#xc778;&#xb371;&#xc2a4;&#xb294; 1-  
 &#bc18;&#xc785;&#xb2c8;&#xb2e4;.



#### Note

&#xaa8;&#xb4e0; &#xb3c4;&#xd615;&#xb4e4;&#xc744; &#xcd94;&#xcd9c;&#xd558;&#xc0;&#xc790;  
 &#xd55c;&#xb2e4;&#xba74; ST\_Dump &#xd568;&#xc218;&#xac00; &#xb354; &#xd6a8;&#xc728;&#xc801;&#xc785;&#xb2c8;&#xb2e4;.

2.0.0 &#xc84;&#xc804;&#xbd80;&#xd130; &#xc0ac;&#xc6a9;&#xd560; &#xc218; &#xc788;&#xc2b5;&#xb2c8;&#xb2e4;.



This method implements the SQL/MM specification. SQL-MM ISO/IEC 13249-3: 8.5



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



This function supports Polyhedral surfaces.

**Notes:**

```
-- &#xb2e4;&#xba74;&#xc4; &#xd45c;&#xba74;&#xc758; &#xb450; &#xc88;&#xc9f8; &#xc744; &#xcd94;&#xcd9c;&#xd569;&#xb2c8;&#xb2e4; .
SELECT ST_AsEWKT(ST_PatchN(geom, 2)) As geomewkt
FROM (
VALUES (ST_GeomFromEWKT('POLYHEDRALSURFACE( ((0 0 0, 0 0 1, 0 1 1, 0 1 0, 0 0 0)),
((0 0 0, 0 1 0, 1 1 0, 1 0 0, 0 0 0)), ((0 0 0, 1 0 0, 1 0 1, 0 0 1, 0 0 0)),
((1 1 0, 1 1 1, 1 0 1, 1 0 0, 1 1 0)),
((0 1 0, 0 1 1, 1 1 1, 1 1 0, 0 1 0)), ((0 0 1, 1 0 1, 1 1 1, 0 1 1, 0 0 1)) )' ) )
As foo(geom);

geomewkt
-----
POLYGON((0 0 0,0 1 0,1 1 0,1 0 0,0 0 0))
```





### 8.4.36 ST\_Points

`ST_Points` — Returns a `MultiPoint` containing all the coordinates of a geometry. Duplicate points are preserved, including the start and end points of ring geometries. (If desired, duplicate points can be removed by calling `ST_RemoveRepeatedPoints` on the result).

#### Synopsis

```
geometry ST_Points( geometry geom );
```

#### Notes

Returns a `MultiPoint` containing all the coordinates of a geometry. Duplicate points are preserved, including the start and end points of ring geometries. (If desired, duplicate points can be removed by calling `ST_RemoveRepeatedPoints` on the result).

To obtain information about the position of each coordinate in the parent geometry use `ST_NumPoints`.

M and Z coordinates are preserved if present.



This method supports Circular Strings and Curves



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

2.3.0

#### Example

```
SELECT ST_AsText(ST_Points('POLYGON Z ((30 10 4,10 30 5,40 40 6, 30 10))'));
--result
MULTIPOINT Z ((30 10 4),(10 30 5),(40 40 6),(30 10 4))
```

#### See also

`ST_RemoveRepeatedPoints`, `ST_PointN`

### 8.4.37 ST\_StartPoint

`ST_StartPoint` — Returns the first point of a `LineString`.

#### Synopsis

```
geometry ST_StartPoint(geometry geomA);
```

#### Notes

`LINESTRING`, `CIRCULARLINESTRING`, `POINT`, `LINESTRING`, `CIRCULARLINESTRING`, `NULL`



This method implements the SQL/MM specification. SQL-MM 3: 7.1.3



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



This method supports Circular Strings and Curves

**Note**

Enhanced: 3.2.0 returns a point for all geometries. Prior behavior returns NULLs if input was not a LineString.

```

SELECT ST_StartPoint('LINESTRING(0 1, 0 2)::geometry') AS st_startpoint;
-----
POINT(0 1)

```

2.0.0 &#xc84;&#xc804;&#xbd80;&#xd130; &#xb2e8;&#xc77c; &#xb3c4;&#xd615; &#xba40;&#xd2f0;&#xb77c;&#xc778;&#xc2a4;&#xd2b8;&#xb9c1;&#xc744; &#xc9c0;&#xc6d0;&#xd558;&#xc9c0; &#xc54a;&#xc2b5;&#xb2c8;&#xb2e4;. PostGIS &#xc608;&#xc804; &#xc84;&#xc804;&#xc774;&#xb77c;&#xba74; &#xb2e8;&#xc77c; &#xb77c;&#xc778; &#xba40;&#xd2f0;&#xb77c;&#xc778;&#xc2a4;&#xd2b8;&#xb9c1;&#xc744; &#xc785;&#xb825;&#xc1b;&#xb294; &#xacbd;&#xc6b0; &#xc2dc;&#xc791;&#xc810;&#xc744; &#xc18;&#xd658;&#xd588;&#xc744; &#xac81;&#xb2c8;&#xb2e4;. 2.0.0 &#xc84;&#xc804;&#xc740; &#xb2e4;&#xb978; &#xaa8;&#xb4e0; &#xba40;&#xd2f0;&#xb77c;&#xc778;&#xc2a4;&#xd2b8;&#xb9c1;&#xc98;&#xb7fc; NULL&#xc744; &#xc18;&#xd658;&#xd560; &#xbf0;&#xc785;&#xb2c8;&#xb2e4;. &#xad6c;&#xc2dd; &#xc2b5;&#xc131;&#xc740; &#xbb38;&#xc11c;&#xd654;&#xb418;&#xc9c0; &#xc54a;&#xc740; &#xae30;&#xb2a5;&#xc774;&#xc9c0;&#xb9cc;, &#xc0ac;&#xc6a9;&#xc790; &#xb370;&#xc774;&#xd130;&#xb97c; &#xb77c;&#xc778;&#xc2a4;&#xd2b8;&#xb9c1;&#xc73c;&#xb85c; &#xc800;&#xc7a5;&#xd588;&#xb2e4;&#xace0; &#xac00;&#xc815;&#xd55c; &#xc0ac;&#xc6a9;&#xc790;&#xc758; &#xacbd;&#xc6b0; &#xd604;&#xc7ac; 2.0 &#xc84;&#xc804;&#xc5d0;&#xc11c; NULL&#xc774; &#xc18;&#xd658;&#xb420; &#xc218;&#xb3c4; &#xc788;&#xc2b5;&#xb2c8;&#xb2e4;.


**&#xc608;&#xc2dc;**

Start point of a LineString

```

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

```

Start point of a non-LineString is NULL

```

SELECT ST_StartPoint('POINT(0 1)::geometry') IS NULL AS is_null;
-----
is_null
-----
t

```

Start point of a 3D LineString

```

SELECT ST_AsEWKT(ST_StartPoint('LINESTRING(0 1 1, 0 2 2)::geometry'));
-----
st_asewkt
-----
POINT(0 1 1)

```

ST\_LineString &#xb610;&#xb294; ST\_CircularString &#xac12;&#xc5d0; &#xc788;&#xb294; &#xd3ec;&#xc778;&#xd2b8;&#xc758 &#xac1c;&#xc218;&#xb97c; &#xc18;&#xd658;&#xd569;&#xb2c8;&#xb2e4;.

```

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

```

**&#xc38;&#xace0;**

**ST\_EndPoint, ST\_PointN**

## 8.4.38 ST\_Summary

ST\_Summary — Returns a text summary of the geometry.

### Synopsis

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

### Options

Options: `geom` (geometry), `geog` (geography).

Options: `geom` (geometry), `geog` (geography), `summary` (summary type), `summary_type` (summary type).

- M: M (MultiPoint)
- Z: Z (MultiLineString)
- B: B (MultiPolygon)
- G: G (MultiSurface)
- S: S (MultiSurface)



This method supports Circular Strings and Curves



This function supports Polyhedral surfaces.



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

1.2.2 Summary (MultiPoint):

2.0.0 Summary (MultiLineString):

2.1.0 Summary (MultiPolygon):

2.2.0 Summary (MultiSurface):

### Examples

```
=# SELECT ST_Summary(ST_GeomFromText('LINESTRING(0 0, 1 1)')) as geom,
           ST_Summary(ST_GeogFromText('POLYGON((0 0, 1 1, 1 2, 1 1, 0 0))')) geog;
 geom | geog
-----+-----
 LineString[B] with 2 points | Polygon[BGS] with 1 rings
 | ring 0 has 5 points
 :
(1 row)

=# SELECT ST_Summary(ST_GeogFromText('LINESTRING(0 0 1, 1 1 1)')) As geog_line,
```

```

ST_Summary(ST_GeomFromText('SRID=4326;POLYGON((0 0 1, 1 1 2, 1 2 3, 1 1 1, 0 0 1)) ←
  ') As geom_poly;
;
-----+-----
geog_line | geom_poly
-----+-----
LineString[ZBGS] with 2 points | Polygon[ZBS] with 1 rings
: ring 0 has 5 points
:
(1 row)

```

**Related Functions:**

[PostGIS\\_DropBBox](#), [PostGIS\\_AddBBox](#), [ST\\_Force3DM](#), [ST\\_Force3DZ](#), [ST\\_Force2D](#), [geography](#)  
[ST\\_IsValid](#), [ST\\_IsValidReason](#), [ST\\_IsValidDetail](#)

### 8.4.39 ST\_X

**ST\_X** — Returns the X coordinate of a Point.

#### Synopsis

```
float ST_X(geometry a_point);
```

**Parameters:**

**a\_point**: X, Y, Z, M, or 3D/4D point geometry. If the geometry is NULL, the function returns NULL. If the geometry is a Point, the function returns the X coordinate. If the geometry is a LineString, the function returns the X coordinate of the first point. If the geometry is a Polygon, the function returns the X coordinate of the first point in the first ring.



#### Note

To get the minimum and maximum X value of geometry coordinates use the functions [ST\\_XMin](#) and [ST\\_XMax](#).



This method implements the SQL/MM specification. SQL-MM 3: 6.1.3



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

**Examples:**

```

SELECT ST_X(ST_GeomFromEWKT('POINT(1 2 3 4)'));
 st_x
-----
      1
(1 row)

SELECT ST_Y(ST_Centroid(ST_GeomFromEWKT('LINESTRING(1 2 3 4, 1 1 1 1)')));
 st_y
-----
    1.5
(1 row)

```



## Synopsis

```
float ST_Z(geometry a_point);
```

### Parameters

`Z`: Z coordinate of the point. `NULL`: NULL. `0`: 2D. `1`: 3D-M. `2`: 3D-Z. `3`: 4D.



#### Note

To get the minimum and maximum Z value of geometry coordinates use the functions [ST\\_ZMin](#) and [ST\\_ZMax](#).



This method implements the SQL/MM specification.



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

### Examples

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

### See also

[ST\\_GeomFromEWKT](#), [ST\\_M](#), [ST\\_X](#), [ST\\_Y](#), [ST\\_ZMax](#), [ST\\_ZMin](#)

## 8.4.42 ST\_Zmflag

`ST_Zmflag` — ST\_Geometry Z-index flag

### Synopsis

```
smallint ST_Zmflag(geometry geomA);
```

### Parameters

`ST_Geometry`: ST\_Geometry Z-index flag. `0`: 2D. `1`: 3D-M. `2`: 3D-Z. `3`: 4D.



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



This method supports Circular Strings and Curves

**Examples**

```

SELECT ST_Zmflag(ST_GeomFromEWKT('LINESTRING(1 2, 3 4)'));
 st_zmflag
-----
                0

SELECT ST_Zmflag(ST_GeomFromEWKT('LINESTRINGM(1 2 3, 3 4 3)'));
 st_zmflag
-----
                1

SELECT ST_Zmflag(ST_GeomFromEWKT('CIRCULARSTRING(1 2 3, 3 4 3, 5 6 3)'));
 st_zmflag
-----
                2

SELECT ST_Zmflag(ST_GeomFromEWKT('POINT(1 2 3 4)'));
 st_zmflag
-----
                3

```

**See also**

[ST\\_CoordDim](#), [ST\\_NDims](#), [ST\\_Dimension](#)

## 8.5 [ST\\_AddPoint](#) (editor)

### 8.5.1 [ST\\_AddPoint](#)

[ST\\_AddPoint](#) — Add a point to a LineString

**Synopsis**

geometry [ST\\_AddPoint](#)(geometry linestring, geometry point);

geometry [ST\\_AddPoint](#)(geometry linestring, geometry point, integer position = -1);

**Notes**

Adds a point to a LineString before the index *position* (using a 0-based index). If the *position* parameter is omitted or is -1 the point is appended to the end of the LineString.

1.1.0 [New](#): [Added](#) [Support](#) for 3D geometry.



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

**Examples**

Add a point to the end of a 3D line

```

SELECT ST_AsEWKT(ST_AddPoint('LINESTRING(0 0 1, 1 1 1)', ST_MakePoint(1, 2, 3)));

 st_asewkt
-----
LINESTRING(0 0 1,1 1 1,1 2 3)

```

Guarantee all lines in a table are closed by adding the start point of each line to the end of the line only for those that are not closed.

```
UPDATE sometable
SET geom = ST_AddPoint(geom, ST_StartPoint(geom))
FROM sometable
WHERE ST_IsClosed(geom) = false;
```

**Warning**

[ST\\_RemovePoint](#), [ST\\_SetPoint](#)

## 8.5.2 ST\_CollectionExtract

`ST_CollectionExtract` — Given a geometry collection, returns a multi-geometry containing only elements of a specified type.

### Synopsis

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

**Warning**

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

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

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

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

For atomic geometry inputs, the geometry is returned unchanged if the input type matches the requested type. Otherwise, the result is an EMPTY geometry of the specified type. If required, these can be converted to multi-geometries using [ST\\_Multi](#).



### Warning

MultiPolygon results are not checked for validity. If the polygon components are adjacent or overlapping the result will be invalid. (For example, this can occur when applying this function to an [ST\\_Split](#) result.) This situation can be checked with [ST\\_IsValid](#) and repaired with [ST\\_MakeValid](#).

1.5.0 [Warning](#): Prior to 1.5.3 this function returned atomic inputs unchanged, no matter type. In 1.5.3 non-matching single geometries returned a NULL result. In 2.0.0 non-matching single geometries return an EMPTY result of the requested type.



### Note

Prior to 1.5.3 this function returned atomic inputs unchanged, no matter type. In 1.5.3 non-matching single geometries returned a NULL result. In 2.0.0 non-matching single geometries return an EMPTY result of the requested type.

**Extract highest-dimension type:**

Extract highest-dimension type:

```
SELECT ST_AsText(ST_CollectionExtract(
    'GEOMETRYCOLLECTION( POINT(0 0), LINESTRING(1 1, 2 2) )');
    st_astext
    -----
MULTILINESTRING((1 1, 2 2))
```

**Extract points (type 1 == POINT):**

```
SELECT ST_AsText(ST_CollectionExtract(
    'GEOMETRYCOLLECTION(GEOMETRYCOLLECTION(POINT(0 0))),
    1));
    st_astext
    -----
MULTIPOINT((0 0))
```

**Extract lines (type 2 == LINESTRING):**

```
SELECT ST_AsText(ST_CollectionExtract(
    'GEOMETRYCOLLECTION(GEOMETRYCOLLECTION(LINESTRING(0 0, 1 1)),LINESTRING(2 2, 3 3))');
    st_astext
    -----
MULTILINESTRING((0 0, 1 1), (2 2, 3 3))
```

**ST\_CollectionHomogenize, ST\_Multi, ST\_IsValid, ST\_MakeValid**

**ST\_CollectionHomogenize, ST\_Multi, ST\_IsValid, ST\_MakeValid**

**8.5.3 ST\_CollectionHomogenize**

**ST\_CollectionHomogenize** — Returns the simplest representation of a geometry collection.

**Synopsis**

geometry **ST\_CollectionHomogenize**(geometry collection);

**Behavior**

Homogeneous (uniform) collections are returned as the appropriate multi-geometry. Heterogeneous (mixed) collections are flattened into a single GeometryCollection. Collections containing a single atomic element are returned as that element. Atomic geometries are returned unchanged. If required, these can be converted to a multi-geometry using **ST\_Multi**.

- Homogeneous (uniform) collections are returned as the appropriate multi-geometry.
- Heterogeneous (mixed) collections are flattened into a single GeometryCollection.
- Collections containing a single atomic element are returned as that element.
- Atomic geometries are returned unchanged. If required, these can be converted to a multi-geometry using **ST\_Multi**.

**Warning**

This function does not ensure that the result is valid. In particular, a collection containing adjacent or overlapping Polygons will create an invalid MultiPolygon. This situation can be checked with [ST\\_IsValid](#) and repaired with [ST\\_MakeValid](#).

2.0.0

**Single-element collection converted to an atomic geometry**

Single-element collection converted to an atomic geometry

```
SELECT ST_AsText(ST_CollectionHomogenize('GEOMETRYCOLLECTION(POINT(0 0))'));
```

```
st_astext
-----
POINT(0 0)
```

Nested single-element collection converted to an atomic geometry:

```
SELECT ST_AsText(ST_CollectionHomogenize('GEOMETRYCOLLECTION(MULTIPOINT((0 0)))'));
```

```
st_astext
-----
POINT(0 0)
```

Collection converted to a multi-geometry:

```
SELECT ST_AsText(ST_CollectionHomogenize('GEOMETRYCOLLECTION(POINT(0 0),POINT(1 1))'));
```

```
st_astext
-----
MULTIPOINT((0 0),(1 1))
```

Nested heterogeneous collection flattened to a GeometryCollection:

```
SELECT ST_AsText(ST_CollectionHomogenize('GEOMETRYCOLLECTION(POINT(0 0), GEOMETRYCOLLECTION ←
(LINESTRING(1 1, 2 2))')));
```

```
st_astext
-----
GEOMETRYCOLLECTION(POINT(0 0),LINESTRING(1 1,2 2))
```

Collection of Polygons converted to an (invalid) MultiPolygon:

```
SELECT ST_AsText(ST_CollectionHomogenize('GEOMETRYCOLLECTION (POLYGON ((10 50, 50 50, 50 ←
10, 10 10, 10 50)), POLYGON ((90 50, 90 10, 50 10, 50 50, 90 50)))'));
```

```
st_astext
-----
MULTIPOLYGON(((10 50,50 50,50 10,10 10,10 50)),((90 50,90 10,50 10,50 50,90 50)))
```

**ST\_CollectionExtract, ST\_Multi, ST\_IsValid, ST\_MakeValid**

[ST\\_CollectionExtract](#), [ST\\_Multi](#), [ST\\_IsValid](#), [ST\\_MakeValid](#)

## 8.5.4 ST\_CurveToLine

ST\_CurveToLine — Converts a geometry containing curves to a linear geometry.

### Synopsis

geometry **ST\_CurveToLine**(geometry curveGeom, float tolerance, integer tolerance\_type, integer flags);

### Usage

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

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

The `tolerance\_type` argument determines interpretation of the `tolerance` argument. It can take the following values:

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

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

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

Availability: 1.3.0

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

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



This method implements the [OGC Simple Features Implementation Specification for SQL 1.1](#).



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



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



This method supports Circular Strings and Curves

### Examples

```
SELECT ST_AsText(ST_CurveToLine(ST_GeomFromText('CIRCULARSTRING(220268 150415,220227 150505,220227 150406)')));
```

--Result --

```
LINESTRING(220268 150415,220269.95064912 150416.539364228,220271.823415575 150418.17258804,220273.613787707 150419.895736857,220275.317452352 150421.704659462,220276.930305234 150423.594998003,220278.448460847 150425.562198489,220279.868261823 150427.60152176,220281.186287736 150429.708054909,220282.399363347 150431.876723113,
```

220283.50456625 150434.10230186,220284.499233914 150436.379429536,220285.380970099 ↔  
150438.702620341,220286.147650624 150441.066277505,  
220286.797428488 150443.464706771,220287.328738321 150445.892130112,220287.740300149 ↔  
150448.342699654,  
220288.031122486 150450.810511759,220288.200504713 150453.289621251,220288.248038775 ↔  
150455.77405574,  
220288.173610157 150458.257830005,220287.977398166 150460.734960415,220287.659875492 ↔  
150463.199479347,  
220287.221807076 150465.64544956,220286.664248262 150468.066978495,220285.988542259 ↔  
150470.458232479,220285.196316903 150472.81345077,  
220284.289480732 150475.126959442,220283.270218395 150477.39318505,220282.140985384 ↔  
150479.606668057,  
220280.90450212 150481.762075989,220279.5637474 150483.85421628,220278.12195122 ↔  
150485.87804878,  
220276.582586992 150487.828697901,220274.949363179 150489.701464356,220273.226214362 ↔  
150491.491836488,  
220271.417291757 150493.195501133,220269.526953216 150494.808354014,220267.559752731 ↔  
150496.326509628,  
220265.520429459 150497.746310603,220263.41389631 150499.064336517,220261.245228106 ↔  
150500.277412127,  
220259.019649359 150501.38261503,220256.742521683 150502.377282695,220254.419330878 ↔  
150503.259018879,  
220252.055673714 150504.025699404,220249.657244448 150504.675477269,220247.229821107 ↔  
150505.206787101,  
220244.779251566 150505.61834893,220242.311439461 150505.909171266,220239.832329968 ↔  
150506.078553494,  
220237.347895479 150506.126087555,220234.864121215 150506.051658938,220232.386990804 ↔  
150505.855446946,  
220229.922471872 150505.537924272,220227.47650166 150505.099855856,220225.054972724 ↔  
150504.542297043,  
220222.663718741 150503.86659104,220220.308500449 150503.074365683,  
220217.994991777 150502.167529512,220215.72876617 150501.148267175,  
220213.515283163 150500.019034164,220211.35987523 150498.7825509,  
220209.267734939 150497.441796181,220207.243902439 150496,  
220205.293253319 150494.460635772,220203.420486864 150492.82741196,220201.630114732 ↔  
150491.104263143,  
220199.926450087 150489.295340538,220198.313597205 150487.405001997,220196.795441592 ↔  
150485.437801511,  
220195.375640616 150483.39847824,220194.057614703 150481.291945091,220192.844539092 ↔  
150479.123276887,220191.739336189 150476.89769814,  
220190.744668525 150474.620570464,220189.86293234 150472.297379659,220189.096251815 ↔  
150469.933722495,  
220188.446473951 150467.535293229,220187.915164118 150465.107869888,220187.50360229 ↔  
150462.657300346,  
220187.212779953 150460.189488241,220187.043397726 150457.710378749,220186.995863664 ↔  
150455.22594426,  
220187.070292282 150452.742169995,220187.266504273 150450.265039585,220187.584026947 ↔  
150447.800520653,  
220188.022095363 150445.35455044,220188.579654177 150442.933021505,220189.25536018 ↔  
150440.541767521,  
220190.047585536 150438.18654923,220190.954421707 150435.873040558,220191.973684044 ↔  
150433.60681495,  
220193.102917055 150431.393331943,220194.339400319 150429.237924011,220195.680155039 ↔  
150427.14578372,220197.12195122 150425.12195122,  
220198.661315447 150423.171302099,220200.29453926 150421.298535644,220202.017688077 ↔  
150419.508163512,220203.826610682 150417.804498867,  
220205.716949223 150416.191645986,220207.684149708 150414.673490372,220209.72347298 ↔  
150413.253689397,220211.830006129 150411.935663483,  
220213.998674333 150410.722587873,220216.22425308 150409.61738497,220218.501380756 ↔  
150408.622717305,220220.824571561 150407.740981121,  
220223.188228725 150406.974300596,220225.586657991 150406.324522731,220227 150406)

```

--3d example
SELECT ST_AsEWKT(ST_CurveToLine(ST_GeomFromEWKT('CIRCULARSTRING(220268 150415 1,220227 150505 2,220227 150406 3)')));
Output
-----
LINESTRING(220268 150415 1,220269.95064912 150416.539364228 1.0181172856673,
220271.823415575 150418.17258804 1.03623457133459,220273.613787707 150419.895736857 1.05435185700189,...AD INFINITUM ...
220225.586657991 150406.324522731 1.32611114201132,220227 150406 3)

--use only 2 segments to approximate quarter circle
SELECT ST_AsText(ST_CurveToLine(ST_GeomFromText('CIRCULARSTRING(220268 150415,220227 150505,220227 150406)'),2));
st_astext
-----
LINESTRING(220268 150415,220287.740300149 150448.342699654,220278.12195122 150485.87804878,
220244.779251566 150505.61834893,220207.243902439 150496,220187.50360229 150462.657300346,
220197.12195122 150425.12195122,220227 150406)

-- Ensure approximated line is no further than 20 units away from
-- original curve, and make the result direction-neutral
SELECT ST_AsText(ST_CurveToLine(
'CIRCULARSTRING(0 0,100 -100,200 0)::geometry,
20, -- Tolerance
1, -- Above is max distance between curve and line
1 -- Symmetric flag
));
st_astext
-----
LINESTRING(0 0,50 -86.6025403784438,150 -86.6025403784439,200 -1.1331077795296e-13,200 0)

```

**ST\_LineToCurve**

**ST\_LineToCurve**

### 8.5.5 ST\_Scroll

ST\_Scroll — Change start point of a closed LineString.

#### Synopsis

geometry **ST\_Scroll**(geometry linestring, geometry point);

**ST\_Scroll**

Changes the start/end point of a closed LineString to the given vertex *point*.

Availability: 3.2.0



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



This function supports M coordinates.

**Example**

Make a closed line start at its 3rd vertex

```
SELECT ST_AsEWKT(ST_Scroll(LINESTRING(0 0 0 1, 10 0 2 0, 5 5 4 2, 0 0 0 1), ' ←
    POINT(5 5 4 2)'));

st_asewkt
-----
SRID=4326;LINESTRING(5 5 4 2,0 0 0 1,10 0 2 0,5 5 4 2)
```

**See also**

[ST\\_Normalize](#)

## 8.5.6 ST\_FlipCoordinates

**ST\_FlipCoordinates** — Returns a version of a geometry with X and Y axis flipped.

**Synopsis**

geometry **ST\_FlipCoordinates**(geometry geom);

**Description**

Returns a version of the given geometry with X and Y axis flipped. Useful for fixing geometries which contain coordinates expressed as latitude/longitude (Y,X).

2.0.0 and later

-  This method supports Circular Strings and Curves
-  This function supports 3d and will not drop the z-index.
-  This function supports M coordinates.
-  This function supports Polyhedral surfaces.
-  This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).

**Example**

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

**See also**

[ST\\_SwapOrdinates](#)



## Synopsis

geometry **ST\_Force3D**(geometry geomA, float Zvalue = 0.0);

### Details

Forces the geometries into XYZ mode. This is an alias for `ST_Force3DZ`. If a geometry has no Z component, then a *Zvalue* Z coordinate is tacked on.

2.0.0: `ST_Force3D` (geometry geomA, float Zvalue = 0.0);

2.1.0: `ST_Force3D` (geometry geomA, float Zvalue = 0.0);

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



This function supports Polyhedral surfaces.



This method supports Circular Strings and Curves



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

### Examples

```
-- ST_Force3D (geometry geomA, float Zvalue = 0.0);
SELECT ST_AsEWKT(ST_Force3D(ST_GeomFromEWKT('CIRCULARSTRING(1 1 2, 2 3 2, 4 5 2, 6 7 2, 5 6 2)')));
st_asewkt
-----
CIRCULARSTRING(1 1 2,2 3 2,4 5 2,6 7 2,5 6 2)

SELECT ST_AsEWKT(ST_Force3D('POLYGON((0 0,0 5,5 0,0 0),(1 1,3 1,1 3,1 1))'));
st_asewkt
-----
POLYGON((0 0 0,0 5 0,5 0 0,0 0 0),(1 1 0,3 1 0,1 3 0,1 1 0))
```

### See Also

[ST\\_AsEWKT](#), [ST\\_Force2D](#), [ST\\_Force3DM](#), [ST\\_Force3DZ](#)

## 8.5.9 ST\_Force3DZ

`ST_Force3DZ` — `ST_Force3DZ` (geometry geomA, float Zvalue = 0.0);

## Synopsis

geometry **ST\_Force3DZ**(geometry geomA, float Zvalue = 0.0);

**ST\_Force3DZ**

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

2.0.0: `ST_Force3DZ(geometry geomA, float Mvalue = 0.0)`  
 surface)

2.1.0: `ST_Force3DZ(geometry geomA, float Mvalue = 0.0, float Zvalue)`  
 ST\_Force\_3DZ

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



This function supports Polyhedral surfaces.



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



This method supports Circular Strings and Curves

**ST\_Force3DZ**

```
-- ST_Force3DZ(geom, float Mvalue, float Zvalue)
SELECT ST_AsEWKT(ST_Force3DZ(ST_GeomFromEWKT('CIRCULARSTRING(1 1 2, 2 3 2, 4 5 2, 6 7 2, 5 6 2)')));
st_asewkt
-----
CIRCULARSTRING(1 1 2,2 3 2,4 5 2,6 7 2,5 6 2)

SELECT ST_AsEWKT(ST_Force3DZ('POLYGON((0 0,0 5,5 0,0 0),(1 1,3 1,1 3,1 1))'));
st_asewkt
-----
POLYGON((0 0,0 5,5 0,0 0),(1 1,3 1,1 3,1 1))
```

**ST\_Force3D**

[ST\\_AsEWKT](#), [ST\\_Force2D](#), [ST\\_Force3DM](#), [ST\\_Force3D](#)

**8.5.10 ST\_Force3DM**

`ST_Force3DM(geometry geomA, float Mvalue = 0.0)` — XYM

**Synopsis**

geometry **ST\_Force3DM**(geometry geomA, float Mvalue = 0.0);

**ST\_Force3DM**

Forces the geometries into XYM mode. If a geometry has no M component, then a *Mvalue* M coordinate is tacked on. If it has a Z component, then Z is removed

2.1.0: `ST_Force3DM(geometry geomA, float Mvalue)`  
 ST\_Force\_3DM

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



This method supports Circular Strings and Curves

**8.5.10 ST\_Force3DM**

```
-- 3D CircularString: 3D CircularString (Z and M values)
SELECT ST_AsEWKT(ST_Force3DM(ST_GeomFromEWKT('CIRCULARSTRING(1 1 2, 2 3 2, 4 5 2, 6 7 2, 5 6 2)')));
st_asewkt
-----
CIRCULARSTRINGM(1 1 0,2 3 0,4 5 0,6 7 0,5 6 0)

SELECT ST_AsEWKT(ST_Force3DM('POLYGON((0 0 1,0 5 1,5 0 1,0 0 1),(1 1 1,3 1 1,1 3 1,1 1 1))'));
st_asewkt
-----
POLYGONM((0 0 0,0 5 0,5 0 0,0 0 0),(1 1 0,3 1 0,1 3 0,1 1 0))
```

**8.5.11 ST\_Force4D**

**ST\_AsEWKT, ST\_Force2D, ST\_Force3DM, ST\_Force3D, ST\_GeomFromEWKT**

**8.5.11 ST\_Force4D**

ST\_Force4D — 4D Geometry (XYZM) with Z and M values.

**Synopsis**

geometry **ST\_Force4D**(geometry geomA, float Zvalue = 0.0, float Mvalue = 0.0);

**8.5.11.1 Details**

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

2.1.0: Added support for 3D geometries. 2.0.x: Added support for 4D geometries. 1.5.0: Added support for 4D geometries.

Changed: 3.1.0. Added support for supplying non-zero Z and M values.



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



This method supports Circular Strings and Curves

**8.5.11.2 Examples**

```
-- 4D CircularString: 4D CircularString (Z and M values)
SELECT ST_AsEWKT(ST_Force4D(ST_GeomFromEWKT('CIRCULARSTRING(1 1 2, 2 3 2, 4 5 2, 6 7 2, 5 6 2)')));
st_asewkt
-----
CIRCULARSTRING(1 1 2 0,2 3 2 0,4 5 2 0,6 7 2 0,5 6 2 0)
```

```
SELECT ST_AsEWKT(ST_Force4D('MULTILINESTRING((0 0 1,0 5 2,5 0 3,0 0 4),(1 1 1,3 1 1,1 3 1,1 1 1))'));
st_asewkt
-----
MULTILINESTRING((0 0 0 1,0 5 0 2,5 0 0 3,0 0 0 4),(1 1 0 1,3 1 0 1,1 3 0 1,1 1 0 1))
```

**Notes:**

[ST\\_AsEWKT](#), [ST\\_Force2D](#), [ST\\_Force3DM](#), [ST\\_Force3D](#)

### 8.5.12 ST\_ForcePolygonCCW

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

#### Synopsis

geometry `ST_ForcePolygonCCW` ( geometry geom );

**Notes:**

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

Availability: 2.4.0



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



This function supports M coordinates.

**Notes:**

[ST\\_ForcePolygonCW](#) , [ST\\_IsPolygonCCW](#) , [ST\\_IsPolygonCW](#)

### 8.5.13 ST\_ForceCollection

`ST_ForceCollection` — Forces all polygons in a collection to be oriented counter-clockwise for their exterior rings and clockwise for their interior rings.

#### Synopsis

geometry `ST_ForceCollection`(geometry geomA);

**Notes:**

Forces all polygons in a collection to be oriented counter-clockwise for their exterior rings and clockwise for their interior rings. Non-polygonal geometries are returned unchanged. WKB output is in the same format as the input. Availability: 2.0.0

Availability: 2.0.0. This function is available in the `ST_ForceCollection` function. Availability: 2.0.0. This function is available in the `ST_ForceCollection` function.



### 8.5.14 ST\_ForcePolygonCW

ST\_ForcePolygonCW — Orients all exterior rings clockwise and all interior rings counter-clockwise.

#### Synopsis

geometry ST\_ForcePolygonCW ( geometry geom );

#### Notes

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

Availability: 2.4.0



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



This function supports M coordinates.

#### See also

[ST\\_ForcePolygonCCW](#) , [ST\\_IsPolygonCCW](#) , [ST\\_IsPolygonCW](#)

### 8.5.15 ST\_ForceSFS

ST\_ForceSFS — Forces SFS 1.1 geometries to use a consistent orientation for their exterior rings and interior rings.

#### Synopsis

geometry ST\_ForceSFS(geometry geomA);  
geometry ST\_ForceSFS(geometry geomA, text version);

#### Notes



This function supports Polyhedral surfaces.



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



This method supports Circular Strings and Curves



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

### 8.5.16 ST\_ForceRHR

ST\_ForceRHR — Forces Right-Hand Rule (RHR) geometries to use a consistent orientation for their exterior rings and interior rings.

#### Synopsis

geometry ST\_ForceRHR(geometry g);

**ST\_ForceRHR**

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

**Note**

The above definition of the Right-Hand-Rule conflicts with definitions used in other contexts. To avoid confusion, it is recommended to use [ST\\_ForcePolygonCW](#).

2.0.0 and will not drop the z-index. surface)



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



This function supports Polyhedral surfaces.

**ST\_AsEWKT**

```
SELECT ST_AsEWKT (
  ST_ForceRHR (
    'POLYGON((0 0 2, 5 0 2, 0 5 2, 0 0 2), (1 1 2, 1 3 2, 3 1 2, 1 1 2))'
  )
);
st_asewkt
-----
POLYGON((0 0 2,0 5 2,5 0 2,0 0 2), (1 1 2,3 1 2,1 3 2,1 1 2))
(1 row)
```

**ST\_ForcePolygonCCW**

[ST\\_ForcePolygonCCW](#) , [ST\\_ForcePolygonCW](#) , [ST\\_IsPolygonCCW](#) , [ST\\_IsPolygonCW](#) , [ST\\_BuildArea](#), [ST\\_Polygonize](#), [ST\\_Reverse](#)

**8.5.17 ST\_ForceCurve**

**ST\_ForceCurve** — Forces the orientation of the vertices in a polygon to follow a Right-Hand-Rule, in which the area that is bounded by the polygon is to the right of the boundary. In particular, the exterior ring is orientated in a clockwise direction and the interior rings in a counter-clockwise direction. This function is a synonym for [ST\\_ForcePolygonCW](#)

**Synopsis**

geometry **ST\_ForceCurve**(geometry g);

**ST\_ForceCurve**

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

ST\_AsText(  
 ST\_ForceCurve(  
 'POLYGON((0 0 2, 5 0 2, 0 5 2, 0 0 2),(1 1 2, 1 3 2, 3 1 2, 1 1 2))'::geometry  
 )  
 );  
 st\_astext

2.2.0 ST\_AsText(  
 ST\_ForceCurve(  
 'POLYGON((0 0 2, 5 0 2, 0 5 2, 0 0 2),(1 1 2, 1 3 2, 3 1 2, 1 1 2))'::geometry  
 )  
 );  
 st\_astext



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



This method supports Circular Strings and Curves

## ST\_AsText

```
SELECT ST_AsText (
  ST_ForceCurve (
    'POLYGON((0 0 2, 5 0 2, 0 5 2, 0 0 2),(1 1 2, 1 3 2, 3 1 2, 1 1 2))'::geometry
  )
);
st_astext
-----
CURVEPOLYGON Z ((0 0 2,5 0 2,0 5 2,0 0 2),(1 1 2,1 3 2,3 1 2,1 1 2))
(1 row)
```

## ST\_LineToCurve

### ST\_LineToCurve

## 8.5.18 ST\_LineToCurve

ST\_LineToCurve — Converts a linear geometry to a curved geometry.

### Synopsis

geometry **ST\_LineToCurve**(geometry geomANoncircular);

## ST\_LineToCurve

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



### Note

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

Availability: 1.3.0



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



This method supports Circular Strings and Curves

**ST\_CurveToLine**

```

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

curvedastext | non_curvedastext
-----|-----
CURVEPOLYGON(CIRCULARSTRING(4 3,3.12132034355964 0.878679656440359, | POLYGON((4
  3,3.94235584120969 2.41472903395162,3.77163859753386 1.85194970290473,
1 0,-1.12132034355965 5.12132034355963,4 3)) | 3.49440883690764
  1.33328930094119,3.12132034355964 0.878679656440359, | 2.66671069905881
  | 0.505591163092366,2.14805029
  0.228361402466141, | 1.58527096604839
  | 0.0576441587903094,1
  0, | 0.414729033951621
  | 0.0576441587903077,-0.14805029
  0.228361402466137, | -0.666710699058802
  | 0.505591163092361,-1.12132034
  0.878679656440353, | -1.49440883690763
  | 1.33328930094119,-1.77163859
  1.85194970290472 | --ETC--
  | 3.94235584120969
  3.58527096604839,4
  3))
--3D example
SELECT ST_AsText(ST_LineToCurve(geom)) As curved, ST_AsText(geom) AS not_curved
FROM (SELECT ST_Translate(ST_Force3D(ST_Boundary(ST_Buffer(ST_Point(1,3), 2,2))),0,0,3) AS
  geom) AS foo;

          curved | not_curved
-----|-----
CIRCULARSTRING Z (3 3 3,-1 2.99999999999999 3,3 3 3) | LINESTRING Z (3 3 3,2.4142135623731
  1.58578643762691 3,1 1 3, | -0.414213562373092 1.5857864376269
  | 3,-1 2.99999999999999 3,
  | -0.414213562373101 4.41421356237309
  3, | 0.999999999999999 5
  | 3,2.41421356237309 4.4142135623731
  3,3 3 3)
(1 row)

```

**ST\_CurveToLine**

[ST\\_CurveToLine](#)

## 8.5.19 ST\_Multi

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

### Synopsis

geometry ST\_Multi(geometry geom);

#### Parameters

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

#### Example

```
SELECT ST_AsText(ST_Multi('POLYGON ((10 30, 30 30, 30 10, 10 10, 10 30))'));
           st_astext
-----
MULTIPOLYGON(((10 30,30 30,30 10,10 10,10 30)))
```

#### See Also

[ST\\_AsText](#)

## 8.5.20 ST\_Normalize

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

### Synopsis

geometry ST\_Normalize(geometry geom);

#### Parameters

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

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

**SQL**

```
SELECT ST_AsText(ST_Normalize(ST_GeomFromText (
  'GEOMETRYCOLLECTION (
    POINT(2 3),
    MULTILINESTRING((0 0, 1 1), (2 2, 3 3)),
    POLYGON (
      (0 10,0 0,10 0,10 10,0 10),
      (4 2,2 2,2 4,4 4,4 2),
      (6 8,8 8,8 6,6 6,6 8)
    )
  )'
)))
```

st\_astext

```
GEOMETRYCOLLECTION(POLYGON((0 0,0 10,10 10,10 0,0 0),(6 6,8 6,8 8,6 8,6 6),(2 2,4 2,4 4,2 4,2 2)),MULTILINESTRING((2 2,3 3),(0 0,1 1)),POINT(2 3))
(1 row)
```

**SQL**[ST\\_Equals](#),

## 8.5.21 ST\_QuantizeCoordinates

ST\_QuantizeCoordinates — Sets least significant bits of coordinates to zero

**Synopsis**

geometry **ST\_QuantizeCoordinates** ( geometry g , int prec\_x , int prec\_y , int prec\_z , int prec\_m );

**SQL**

`ST_QuantizeCoordinates` determines the number of bits (N) required to represent a coordinate value with a specified number of digits after the decimal point, and then sets all but the N most significant bits to zero. The resulting coordinate value will still round to the original value, but will have improved compressibility. This can result in a significant disk usage reduction provided that the geometry column is using a [compressible storage type](#). The function allows specification of a different number of digits after the decimal point in each dimension; unspecified dimensions are assumed to have the precision of the x dimension. Negative digits are interpreted to refer digits to the left of the decimal point, (i.e., `prec_x=-2` will preserve coordinate values to the nearest 100).

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

Availability: 2.5.0

**Technical Background**

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

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

**Note**

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

**Example**

```
SELECT ST_AsText(ST_QuantizeCoordinates('POINT (100.123456 0)::geometry, 4));
st_astext
-----
POINT(100.123455047607 0)
```

```
WITH test AS (SELECT 'POINT (123.456789123456 123.456789123456)::geometry AS geom)
SELECT
  digits,
  encode(ST_QuantizeCoordinates(geom, digits), 'hex'),
  ST_AsText(ST_QuantizeCoordinates(geom, digits))
FROM test, generate_series(15, -15, -1) AS digits;
```

| digits | encode                                       | st_astext                                  |
|--------|--|--|
| 15     | 01010000005f9a72083cdd5e405f9a72083cdd5e40   | POINT(123.456789123456 123.456789123456) ← |
| 14     | 01010000005f9a72083cdd5e405f9a72083cdd5e40   | POINT(123.456789123456 123.456789123456) ← |
| 13     | 01010000005f9a72083cdd5e405f9a72083cdd5e40   | POINT(123.456789123456 123.456789123456) ← |
| 12     | 01010000005c9a72083cdd5e405c9a72083cdd5e40   | POINT(123.456789123456 123.456789123456) ← |
| 11     | 0101000000409a72083cdd5e40409a72083cdd5e40   | POINT(123.456789123456 123.456789123456) ← |
| 10     | 0101000000009a72083cdd5e40009a72083cdd5e40   | POINT(123.456789123455 123.456789123455) ← |
| 9      | 0101000000009072083cdd5e40009072083cdd5e40   | POINT(123.456789123418 123.456789123418) ← |
| 8      | 0101000000008072083cdd5e40008072083cdd5e40   | POINT(123.45678912336 123.45678912336) ←   |
| 7      | 0101000000000070083cdd5e40000070083cdd5e40   | POINT(123.456789121032 123.456789121032) ← |
| 6      | 0101000000000040083cdd5e40000040083cdd5e40   | POINT(123.456789076328 123.456789076328) ← |
| 5      | 0101000000000000083cdd5e40000000083cdd5e40   | POINT(123.456789016724 123.456789016724) ← |
| 4      | 0101000000000000003cdd5e40000000003cdd5e40   | POINT(123.456787109375 123.456787109375) ← |
| 3      | 010100000000000000003cdd5e40000000003cdd5e40 | POINT(123.456787109375 123.456787109375) ← |





## Synopsis

geometry **ST\_Reverse**(geometry g1);

### Enhanced:

Enhanced: 2.4.0 support for curves was introduced.



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



This function supports Polyhedral surfaces.

### Example

```
SELECT ST_AsText(geom) as line, ST_AsText(ST_Reverse(geom)) As reverseline
FROM
(SELECT ST_MakeLine(ST_Point(1,2),
                   ST_Point(1,10)) As geom) as foo;
--result
      line          |      reverseline
-----+-----
LINESTRING(1 2,1 10) | LINESTRING(1 10,1 2)
```

## 8.5.25 ST\_Segmentize

**ST\_Segmentize** — Segmentize geometry into equal length segments.

### Synopsis

geometry **ST\_Segmentize**(geometry geom, float max\_segment\_length);  
 geography **ST\_Segmentize**(geography geog, float max\_segment\_length);

### Enhanced:

Enhanced: 3.0.0 Segmentize geometry now uses equal length segments  
 Enhanced: 2.3.0 Segmentize geography now uses equal length segments  
 Enhanced: 2.1.0 Segmentize geometry now uses equal length segments  
 Enhanced: 2.1.0 Segmentize geography now uses equal length segments

1.2.2 Segmentize geometry now uses equal length segments

Enhanced: 3.0.0 Segmentize geometry now uses equal length segments

Enhanced: 2.3.0 Segmentize geography now uses equal length segments

Enhanced: 2.1.0 Segmentize geometry now uses equal length segments  
 Enhanced: 2.1.0 Segmentize geography now uses equal length segments

Enhanced: 2.1.0 Segmentize geometry now uses equal length segments  
 Enhanced: 2.1.0 Segmentize geography now uses equal length segments

```
ST_GeomFromText, ST_GeogFromText
SELECT ST_Segmentize('LINESTRING(1 2, 3 4)'::geometry, 0.5);
SELECT ST_Segmentize('MULTILINESTRING((-29 -27,-30 -29.7,-36 -31,-45 -33), (-45 -33,-46 -32))')
AS st_astext;
```

**Note**

```
SELECT ST_Segmentize(ST_GeomFromText('POLYGON((-29 28, -30 40, -29 28))'), 10)
AS st_astext;
```

**ST\_Segmentize**

```
SELECT ST_AsText(ST_Segmentize(
ST_GeomFromText('MULTILINESTRING((-29 -27,-30 -29.7,-36 -31,-45 -33), (-45 -33,-46 -32))')
, 5)
);
st_astext
```

```
MULTILINESTRING((-29 -27,-30 -29.7,-34.886615700134 -30.758766735029,-36 -31,
-40.8809353009198 -32.0846522890933,-45 -33),
(-45 -33,-46 -32))
(1 row)
```

```
SELECT ST_AsText(ST_Segmentize(ST_GeomFromText('POLYGON((-29 28, -30 40, -29 28))'), 10));
st_astext
-----
POLYGON((-29 28,-29.8304547985374 37.9654575824488,-30 40,-29.1695452014626
30.0345424175512,-29 28))
(1 row)
```

**ST\_LineSubstring****ST\_LineSubstring****8.5.26 ST\_SetPoint**

**ST\_SetPoint** — Returns a geometry with a point added to a linestring at the specified position.

**Synopsis**

geometry **ST\_SetPoint**(geometry linestring, integer zerobasedposition, geometry point);

**ST\_SetPoint**

**ST\_SetPoint** — Returns a geometry with a point added to a linestring at the specified position.

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



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

### Examples

```
--Change first point in line string from -1 3 to -1 1
SELECT ST_AsText(ST_SetPoint('LINESTRING(-1 2,-1 3)', 0, 'POINT(-1 1)'));
      st_astext
-----
LINESTRING(-1 1,-1 3)

---Change last point in a line string (lets play with 3d linestring this time)
SELECT ST_AsEWKT(ST_SetPoint(foo.geom, ST_NumPoints(foo.geom) - 1, ST_GeomFromEWKT('POINT (-1 1 3)'))
FROM (SELECT ST_GeomFromEWKT('LINESTRING(-1 2 3,-1 3 4, 5 6 7)') As geom) As foo;
      st_asewkt
-----
LINESTRING(-1 2 3,-1 3 4,-1 1 3)

SELECT ST_AsText(ST_SetPoint(g, -3, p))
FROM ST_GeomFromText('LINESTRING(0 0, 1 1, 2 2, 3 3, 4 4)') AS g
      , ST_PointN(g,1) as p;
      st_astext
-----
LINESTRING(0 0,1 1,0 0,3 3,4 4)
```

### See Also

[ST\\_AddPoint](#), [ST\\_NPoints](#), [ST\\_NumPoints](#), [ST\\_PointN](#), [ST\\_RemovePoint](#)

## 8.5.27 ST\_ShiftLongitude

`ST_ShiftLongitude` — Shifts the longitude coordinates of a geometry between -180..180 and 0..360.

### Synopsis

geometry `ST_ShiftLongitude`(geometry geom);

### Details

Reads every point/vertex in a geometry, and shifts its longitude coordinate from -180..0 to 180..360 and vice versa if between these ranges. This function is symmetrical so the result is a 0..360 representation of a -180..180 data and a -180..180 representation of a 0..360 data.



## Synopsis

geometry **ST\_WrapX**(geometry geom, float8 wrap, float8 move);

### Notes

This function splits the input geometries and then moves every resulting component falling on the right (for negative 'move') or on the left (for positive 'move') of given 'wrap' line in the direction specified by the 'move' parameter, finally re-unioning the pieces together.



#### Note

This function will not drop the z-index. For example, if you have a 3D geometry with a z-index of 10, and you use ST\_WrapX, the resulting geometry will still have a z-index of 10. This is important for 3D applications where the z-index represents elevation or depth.

Availability: 2.3.0 requires GEOS



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

### Usage

```
-- Move all components of the given geometries whose bounding box
-- falls completely on the left of x=0 to +360
select ST_WrapX(geom, 0, 360);

-- Move all components of the given geometries whose bounding box
-- falls completely on the left of x=-30 to +360
select ST_WrapX(geom, -30, 360);
```

### ST\_ShiftLongitude

## ST\_ShiftLongitude

### 8.5.29 ST\_SnapToGrid

**ST\_SnapToGrid** — Snap a geometry to a grid. The grid is defined by the origin and size of the grid cells.

## Synopsis

geometry **ST\_SnapToGrid**(geometry geomA, float originX, float originY, float sizeX, float sizeY);  
 geometry **ST\_SnapToGrid**(geometry geomA, float sizeX, float sizeY);  
 geometry **ST\_SnapToGrid**(geometry geomA, float size);  
 geometry **ST\_SnapToGrid**(geometry geomA, geometry pointOrigin, float sizeX, float sizeY, float sizeZ, float sizeM);

**&#xc124;&#xba85;**

&#xbcc0;&#xc885; 1, 2, 3: &#xc785;&#xb825; &#xb3c4;&#xd615;&#xc758; &#xbaa8;&#xb4e0; &#xd3ec;&#xc778;&#xd2b8;&#xb4dc  
 &#xc6d0;&#xc810;&#xacfc; &#xc140;(cell) &#xd06c;&#xae30;&#xb85c; &#xc815;&#xc758;&#xb41c; &#xadf8;&#xb9ac;&#xb4dc  
 &#xc2a4;&#xb0c5;(snap) &#xc2dc;&#xd0b5;&#xb2c8;&#xb2e4;. &#xb3d9;&#xc77c;&#xd55c; &#xc140;&#xc5d0; &#xb5a8;&#xc5  
 &#xc5f0;&#xc18d;&#xb41c; &#xd3ec;&#xc778;&#xd2b8;&#xb4e4;&#xc744; &#xc81c;&#xac70;&#xd558;&#xba70;. &#xcd9c;&#xc7  
 &#xd3ec;&#xc778;&#xd2b8;&#xb4e4;&#xc774; &#xc8fc;&#xc5b4;&#xc9c4; &#xc720;&#xd615;&#xc758; &#xb3c4;&#xd615;&#xc7  
 &#xc815;&#xc758;&#xd558;&#xae30;&#xc5d0; &#xbd80;&#xc871;&#xd560; &#xacbd;&#xc6b0; &#xacb0;&#xad6d; NULL &#xc7  
 &#xbc18;&#xd658;&#xd569;&#xb2c8;&#xb2e4;. &#xb3c4;&#xd615; &#xc9d1;&#xd569; &#xc548;&#xc5d0;&#xc11c; &#xc774;&  
 &#xbd95;&#xad34;&#xb41c; &#xb3c4;&#xd615;&#xc740; &#xc9d1;&#xd569;&#xc5d0;&#xc11c; &#xc81c;&#xc678;&#xb429;&#xc7  
 &#xc815;&#xd655;&#xb3c4;&#xb97c; &#xb0ae;&#xcd94;&#xb294; &#xb370; &#xc720;&#xc6a9;&#xd569;&#xb2c8;&#xb2e4;.

&#xbcc0;&#xc885; 4: 1.1.0 &#xbc84;&#xc804;&#xc5d0;&#xc11c; &#xcd94;&#xac00;&#xb418;&#xc5c8;&#xc2b5;&#xb2c8;&#xb2  
 &#xc785;&#xb825; &#xb3c4;&#xd615;&#xc758; &#xbaa8;&#xb4e0; &#xd3ec;&#xc778;&#xd2b8;&#xb97c; &#xc6d0;&#xc810; (&  
 &#xbc88;&#xc9f8; &#xc778;&#xc218;. &#xd3ec;&#xc778;&#xd2b8;&#xc5ec;&#xc57c; &#xd569;&#xb2c8;&#xb2e4;) &#xbc0f;  
 &#xc140; &#xd06c;&#xae30;&#xb85c; &#xc815;&#xc758;&#xb41c; &#xadf8;&#xb9ac;&#xb4dc;&#xb85c; &#xc2a4;&#xb0c5;&#xc7  
 &#xc0ac;&#xc6a9;&#xc790;&#xac00; &#xadf8;&#xb9ac;&#xb4dc;&#xb85c; &#xc2a4;&#xb0c5;&#xc2dc;&#xd0a4;&#xace0;  
 &#xc2f6;&#xc9c0; &#xc54a;&#xc740; &#xcc28;&#xc6d0;&#xc758; &#xacbd;&#xc6b0;. &#xc140; &#xd06c;&#xae30;&#xb97c;  
 0 &#xc73c;&#xb85c; &#xc124;&#xc815;&#xd558;&#xc2ed;&#xc2dc;&#xc624;.



**Note**

&#xbc18;&#xd658;&#xb418;&#xb294; &#xb3c4;&#xd615;&#xc774; &#xb2e8;&#xc21c;&#xc131;&#xc744;  
 &#xc783;&#xc744; &#xc218;&#xb3c4; &#xc788;&#xc2b5;&#xb2c8;&#xb2e4;(ST\_IsSimple &#xcc38;&#xc870;).

**Note**

1.1.0 &#xbc84;&#xc804; &#xb3c4;&#xd615;&#xc758; &#xbaa8;&#xb4e0; &#xd3ec;&#xc778;&#xd2b8;&#xb97c; &#xc6d0;&#xc810; (&  
 &#xd568;&#xc218;&#xac00; &#xd56d;&#xc0c1; 2 &#xcc28;&#xc6d0; &#xb3c4;&#xd615;&#xc744;  
 &#xbc18;&#xd658;&#xd588;&#xc2b5;&#xb2c8;&#xb2e4;. 1.1.0 &#xbc84;&#xc804;&#xbd80;&#xd130;  
 &#xbc18;&#xd658;&#xb418;&#xb294; &#xb3c4;&#xd615;&#xc774;. &#xb354; &#xb192;&#xc740;  
 &#xcc28;&#xc6d0;&#xac12;&#xc740; &#xac74;&#xb4dc;&#xb9ac;&#xc9c0; &#xc54a;&#xc740; &#xcc44;. &  
 &#xc785;&#xb825; &#xb3c4;&#xd615;&#xacfc; &#xb3d9;&#xc77c;&#xd55c; &#xcc28;&#xc6d0;&#xc218;&#xb97c;  
 &#xac00;&#xc9c0;&#xac8c; &#xb410;&#xc2b5;&#xb2c8;&#xb2e4;. &#xbaa8;&#xb4e0; &#xadf8;&#xb9ac;&#xb4dc;  
 &#xcc28;&#xc6d0;&#xc744; &#xc815;&#xc758;&#xd558;&#xb824;&#xba74; &#xb450; &#xbc88;&#xc9f8;  
 &#xb3c4;&#xd615; &#xc778;&#xc790;&#xb97c; &#xc785;&#xb825;&#xb1b;&#xb294; &#xbc84;&#xc804;&#xc744;  
 &#xc774;&#xc6a9;&#xd558;&#xc2ed;&#xc2dc;&#xc624;.

1.0.0RC1 &#xbc84;&#xc804;&#xbd80;&#xd130; &#xc0ac;&#xc6a9;&#xd560; &#xc218; &#xc788;&#xc2b5;&#xb2c8;&#xb2e4;.

1.1.0 &#xbc84;&#xc804;&#xbd80;&#xd130; Z &#xbc0f; M &#xc744; &#xc9c0;&#xc6d0;&#xd569;&#xb2c8;&#xb2e4;.



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

**&#xc608;&#xc2dc;**

```
--Snap your geometries to a precision grid of 10^-3
UPDATE mytable
  SET geom = ST_SnapToGrid(geom, 0.001);

SELECT ST_AsText(ST_SnapToGrid(
  ST_GeomFromText('LINESTRING(1.1115678 2.123, 4.111111 3.2374897, ←
    4.11112 3.23748667)'),
  0.001)
);
      st_astext
```

```

-----
LINESTRING(1.112 2.123,4.111 3.237)
--Snap a 4d geometry
SELECT ST_AsEWKT(ST_SnapToGrid(
    ST_GeomFromEWKT('LINESTRING(-1.1115678 2.123 2.3456 1.11111,
        4.111111 3.2374897 3.1234 1.1111, -1.11111112 2.123 2.3456 1.111112)'),
    ST_GeomFromEWKT('POINT(1.12 2.22 3.2 4.4444)'),
    0.1, 0.1, 0.1, 0.01) );
                                                    st_asewkt
-----
LINESTRING(-1.08 2.12 2.3 1.1144,4.12 3.22 3.1 1.1144,-1.08 2.12 2.3 1.1144)

--With a 4d geometry - the ST_SnapToGrid(geom,size) only touches x and y coords but keeps m ←
and z the same
SELECT ST_AsEWKT(ST_SnapToGrid(ST_GeomFromEWKT('LINESTRING(-1.1115678 2.123 3 2.3456,
    4.111111 3.2374897 3.1234 1.1111)'),
    0.01) );
                                                    st_asewkt
-----
LINESTRING(-1.11 2.12 3 2.3456,4.11 3.24 3.1234 1.1111)

```

**ST\_Snap**

[ST\\_Snap](#), [ST\\_AsEWKT](#), [ST\\_AsText](#), [ST\\_GeomFromText](#), [ST\\_GeomFromEWKT](#), [ST\\_Simplify](#)

### 8.5.30 ST\_Snap

**ST\_Snap** — Snaps the vertices and segments of a geometry to another Geometry's vertices. A snap distance tolerance is used to control where snapping is performed. The result geometry is the input geometry with the vertices snapped. If no snapping occurs then the input geometry is returned unchanged.

#### Synopsis

geometry **ST\_Snap**(geometry input, geometry reference, float tolerance);

**ST\_Snap**

Snaps the vertices and segments of a geometry to another Geometry's vertices. A snap distance tolerance is used to control where snapping is performed. The result geometry is the input geometry with the vertices snapped. If no snapping occurs then the input geometry is returned unchanged.

**ST\_Snap**(geometry input, geometry reference, float tolerance);

**ST\_Snap**(geometry input, geometry reference, float tolerance);



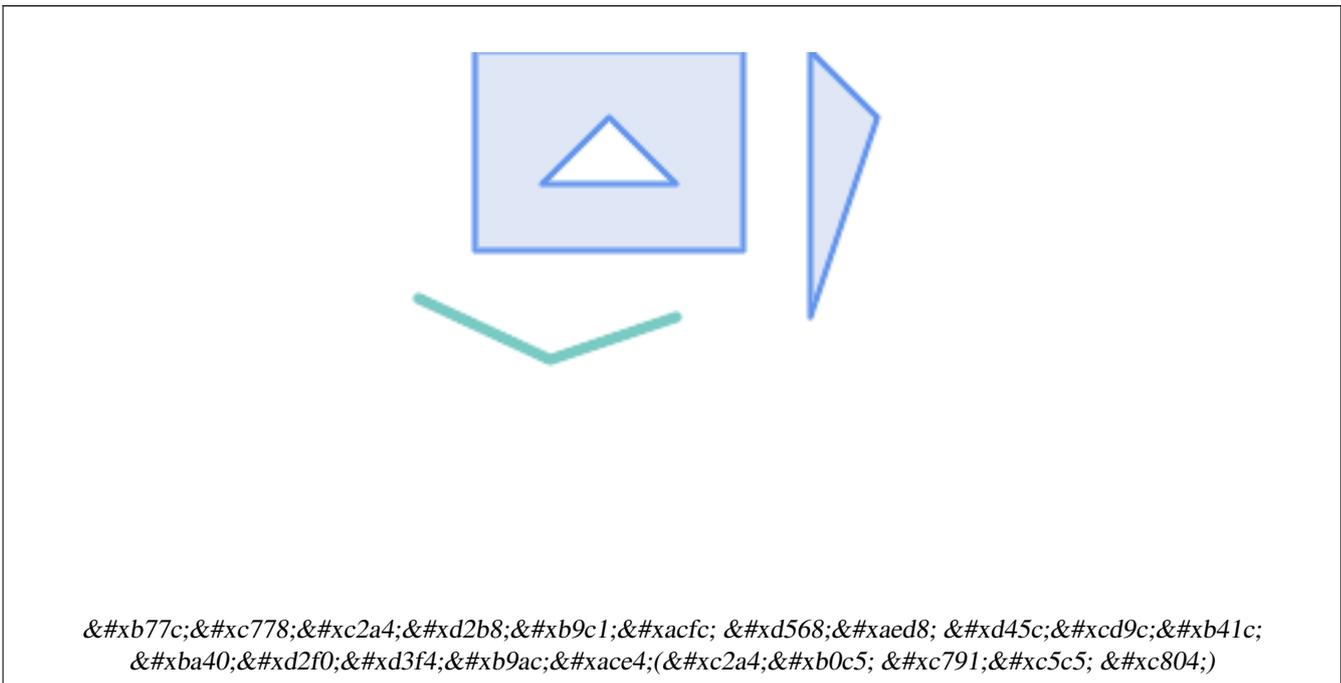
**Note**

ST\_IsSimple  
(ST\_IsValid  
ST\_IsSimple  
ST\_IsValid

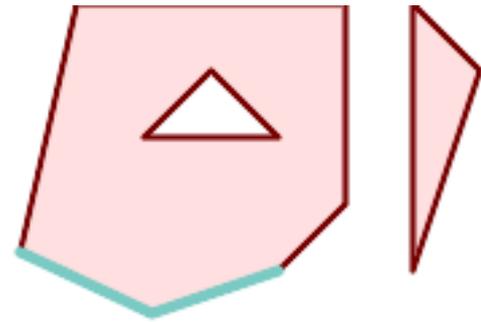
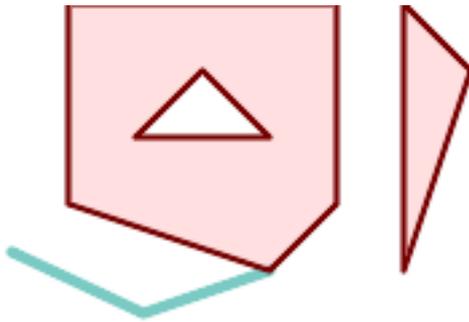
GEOS 2.0.0

2.0.0

ST\_IsSimple



ST\_IsSimple  
ST\_IsValid



```

&#xd5c8;&#xc6a9; &#xac70;&#xb9ac; 1.01&#xc744;
&#xae30;&#xc900;&#xc73c;&#xb85c;
&#xb77c;&#xc778;&#xc2a4;&#xd2b8;&#xb9c1;&#xc5d0;
&#xc2a4;&#xb0c5;&#xb41c;
&#xba40;&#xd2f0;&#xd3f4;&#xb9ac;&#xace4;.
&#xc0c8;&#xb85c;&#xc6b4;
&#xba40;&#xd2f0;&#xd3f4;&#xb9ac;&#xace4;&#xc740;
&#xc38;&#xc870;
&#xb77c;&#xc778;&#xc2a4;&#xd2b8;&#xb9c1;&#xafc;
&#xd568;&#xaed8;
&#xd45c;&#xcd9c;&#xb429;&#xb2c8;&#xb2e4;.
    
```

```

SELECT ST_AsText (ST_Snap (poly, line, ←
    ST_Distance (poly, line) * 1.01)) AS polysnapped
FROM (SELECT
    ST_GeomFromText ('MULTIPOLYGON (
        ((26 125, 26 200, 126 200, 126 125, ←
        26 125 ),
        ( 51 150, 101 150, 76 175, 51 150 ) ←
        ),
        (( 151 100, 151 200, 176 175, 151 ←
        100 )))') As poly,
    ST_GeomFromText ('LINESTRING (5 ←
    107, 54 84, 101 100)') As line
    ) As foo;
    
```

polysnapped

```

MULTIPOLYGON(((26 125,26 200,126 200,126 ←
    125,101 100,26 125),
(51 150,101 150,76 175,51 150)),((151 ←
    100,151 200,176 175,151 100)))
    
```

```

&#xd5c8;&#xc6a9; &#xac70;&#xb9ac; 1.25&#xb97c;
&#xae30;&#xc900;&#xc73c;&#xb85c;
&#xb77c;&#xc778;&#xc2a4;&#xd2b8;&#xb9c1;&#xc5d0;
&#xc2a4;&#xb0c5;&#xb41c;
&#xba40;&#xd2f0;&#xd3f4;&#xb9ac;&#xace4;.
&#xc0c8;&#xb85c;&#xc6b4;
&#xba40;&#xd2f0;&#xd3f4;&#xb9ac;&#xace4;&#xc740;
&#xc38;&#xc870;
&#xb77c;&#xc778;&#xc2a4;&#xd2b8;&#xb9c1;&#xafc;
&#xd568;&#xaed8;
&#xd45c;&#xcd9c;&#xb429;&#xb2c8;&#xb2e4;.
    
```

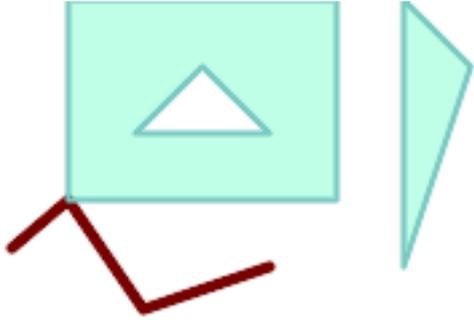
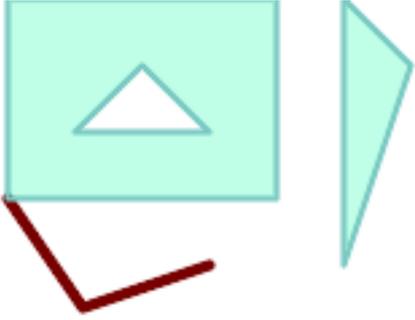
```

SELECT ST_AsText (
    ST_Snap (poly, line, ST_Distance (poly, ←
    line) * 1.25)
) AS polysnapped
FROM (SELECT
    ST_GeomFromText ('MULTIPOLYGON (
        (( 26 125, 26 200, 126 200, 126 125, ←
        26 125 ),
        ( 51 150, 101 150, 76 175, 51 150 ) ←
        ),
        (( 151 100, 151 200, 176 175, 151 ←
        100 )))') As poly,
    ST_GeomFromText ('LINESTRING (5 ←
    107, 54 84, 101 100)') As line
    ) As foo;
    
```

polysnapped

```

MULTIPOLYGON(((5 107,26 200,126 200,126 ←
    125,101 100,54 84,5 107),
(51 150,101 150,76 175,51 150)),((151 ←
    100,151 200,176 175,151 100)))
    
```

|   |  |
|---|--|
|  <pre style="font-family: monospace; font-size: 0.9em;"> &amp;#xd5c8;&amp;#xc6a9; &amp;#xac70;&amp;#xb9ac; 1.01&amp;#xc744; &amp;#xae30;&amp;#xc900;&amp;#xc73c;&amp;#xb85c; &amp;#xc6d0;&amp;#xb798; &amp;#xba40;&amp;#xd2f0;&amp;#xd3f4;&amp;#xb9ac;&amp;#xace4;&amp;#xc5d0; &amp;#xc2a4;&amp;#xb0c5;&amp;#xb41c; &amp;#xb77c;&amp;#xc778;&amp;#xc2a4;&amp;#xd2b8;&amp;#xb9c1;. &amp;#xc0c8;&amp;#xb85c;&amp;#xc6b4; &amp;#xb77c;&amp;#xc778;&amp;#xc2a4;&amp;#xd2b8;&amp;#xb9c1;&amp;#xc740; &amp;#xcc38;&amp;#xc870; &amp;#xba40;&amp;#xd2f0;&amp;#xd3f4;&amp;#xb9ac;&amp;#xace4;&amp;#xacfc; &amp;#xd568;&amp;#xaed8; &amp;#xd45c;&amp;#xcd9c;&amp;#xb429;&amp;#xb2c8;&amp;#xb2e4;.  SELECT ST_AsText (   ST_Snap(line, poly, ST_Distance(poly, ←     line)*1.01) ) AS linesnapped FROM (SELECT   ST_GeomFromText ('MULTIPOLYGON (     ((26 125, 26 200, 126 200, 126 125, ←     26 125),     (51 150, 101 150, 76 175, 51 150 )) ←   ',   ((151 100, 151 200, 176 175, 151 ←   100)))') As poly,   ST_GeomFromText ('LINESTRING (5 ←   107, 54 84, 101 100)') As line ) As foo;            linesnapped ----- LINESTRING(5 107,26 125,54 84,101 100)         </pre> |  <pre style="font-family: monospace; font-size: 0.9em;"> &amp;#xd5c8;&amp;#xc6a9; &amp;#xac70;&amp;#xb9ac; 1.25&amp;#xb97c; &amp;#xae30;&amp;#xc900;&amp;#xc73c;&amp;#xb85c; &amp;#xc6d0;&amp;#xb798; &amp;#xba40;&amp;#xd2f0;&amp;#xd3f4;&amp;#xb9ac;&amp;#xace4;&amp;#xc5d0; &amp;#xc2a4;&amp;#xb0c5;&amp;#xb41c; &amp;#xb77c;&amp;#xc778;&amp;#xc2a4;&amp;#xd2b8;&amp;#xb9c1;. &amp;#xc0c8;&amp;#xb85c;&amp;#xc6b4; &amp;#xb77c;&amp;#xc778;&amp;#xc2a4;&amp;#xd2b8;&amp;#xb9c1;&amp;#xc740; &amp;#xcc38;&amp;#xc870; &amp;#xba40;&amp;#xd2f0;&amp;#xd3f4;&amp;#xb9ac;&amp;#xace4;&amp;#xacfc; &amp;#xd568;&amp;#xaed8; &amp;#xd45c;&amp;#xcd9c;&amp;#xb429;&amp;#xb2c8;&amp;#xb2e4;.  SELECT ST_AsText (   ST_Snap(line, poly, ST_Distance(poly, ←     line)*1.25) ) AS linesnapped FROM (SELECT   ST_GeomFromText ('MULTIPOLYGON (     (( 26 125, 26 200, 126 200, 126 125, ←     26 125 ),     (51 150, 101 150, 76 175, 51 150 )) ←   ',   ((151 100, 151 200, 176 175, 151 ←   100 ))') As poly,   ST_GeomFromText ('LINESTRING (5 ←   107, 54 84, 101 100)') As line ) As foo;            linesnapped ----- LINESTRING(26 125,54 84,101 100)         </pre> |
|---|--|

**&#xcc38;&#xace0;**

**ST\_SnapToGrid**



## Synopsis

```
boolean ST_IsValid(geometry g);
boolean ST_IsValid(geometry g, integer flags);
```

## Description

Tests if an `ST_Geometry` value is well-formed and valid in 2D according to the OGC rules. For geometries with 3 and 4 dimensions, the validity is still only tested in 2 dimensions. For geometries that are invalid, a PostgreSQL NOTICE is emitted providing details of why it is not valid.

For the version with the `flags` parameter, supported values are documented in [ST\\_IsValidDetail](#). This version does not print a NOTICE explaining invalidity.

For more information on the definition of geometry validity, refer to [Section 4.4](#).



### Note

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

Performed by the GEOS module.

The version accepting flags is available starting with 2.0.0.



This method implements the [OGC Simple Features Implementation Specification for SQL 1.1](#).



This method implements the SQL/MM specification. SQL-MM 3: 5.1.9



### Note

Neither OGC-SFS nor SQL-MM specifications include a flag argument for `ST_IsValid`. The flag is a PostGIS extension.

## Examples

```
SELECT ST_IsValid(ST_GeomFromText('LINESTRING(0 0, 1 1)')) As good_line,
       ST_IsValid(ST_GeomFromText('POLYGON((0 0, 1 1, 1 2, 1 1, 0 0))')) As bad_poly
--results
NOTICE: Self-intersection at or near point 0 0
 good_line | bad_poly
-----+-----
 t         | f
```

## See Also

[ST\\_IsSimple](#), [ST\\_IsValidReason](#), [ST\\_IsValidDetail](#),

### 8.6.2 ST\_IsValidDetail

`ST_IsValidDetail` — Returns a `valid_detail` row stating if a geometry is valid or if not a reason and a location.

## Synopsis

```
valid_detail ST_IsValidDetail(geometry geom, integer flags);
```

## Description

Returns a `valid_detail` row, containing a boolean (`valid`) stating if a geometry is valid, a varchar (`reason`) stating a reason why it is invalid and a geometry (`location`) pointing out where it is invalid.

Useful to improve on the combination of [ST\\_IsValid](#) and [ST\\_IsValidReason](#) to generate a detailed report of invalid geometries.

The optional `flags` parameter is a bitfield. It can have the following values:

- 0: Use usual OGC SFS validity semantics.
- 1: Consider certain kinds of self-touching rings (inverted shells and exverted holes) as valid. This is also known as "the ESRI flag", since this is the validity model used by those tools. Note that this is invalid under the OGC model.

Performed by the GEOS module.

Availability: 2.0.0

## Examples

```
--First 3 Rejects from a successful quintuplet experiment
SELECT gid, reason(ST_IsValidDetail(geom)), ST_AsText(location(ST_IsValidDetail(geom))) as
  location
FROM
  (SELECT ST_MakePolygon(ST_ExteriorRing(e.buff), array_agg(f.line)) As geom, gid
  FROM (SELECT ST_Buffer(ST_Point(x1*10,y1), z1) As buff, x1*10 + y1*100 + z1*1000 As gid
        FROM generate_series(-4,6) x1
        CROSS JOIN generate_series(2,5) y1
        CROSS JOIN generate_series(1,8) z1
        WHERE x1 > y1*0.5 AND z1 < x1*y1) As e
        INNER JOIN (SELECT ST_Translate(ST_ExteriorRing(ST_Buffer(ST_Point(x1*10,y1), z1)),
          y1*1, z1*2) As line
        FROM generate_series(-3,6) x1
        CROSS JOIN generate_series(2,5) y1
        CROSS JOIN generate_series(1,10) z1
        WHERE x1 > y1*0.75 AND z1 < x1*y1) As f
  ON (ST_Area(e.buff) > 78 AND ST_Contains(e.buff, f.line))
  GROUP BY gid, e.buff) As quintuplet_experiment
WHERE ST_IsValid(geom) = false
ORDER BY gid
LIMIT 3;
```

| gid  | reason            | location    |
|------|-------------------|-------------|
| 5330 | Self-intersection | POINT(32 5) |
| 5340 | Self-intersection | POINT(42 5) |
| 5350 | Self-intersection | POINT(52 5) |

```
--simple example
SELECT * FROM ST_IsValidDetail('LINESTRING(220227 150406,2220227 150407,222020 150410)');
```

| valid | reason | location |
|-------|--------|----------|
| t     |        |          |

**See Also**[ST\\_IsValid](#), [ST\\_IsValidReason](#)**8.6.3 ST\_IsValidReason****ST\_IsValidReason** — Returns text stating if a geometry is valid, or a reason for invalidity.**Synopsis**

```
text ST_IsValidReason(geometry geomA);
text ST_IsValidReason(geometry geomA, integer flags);
```

**Description**

Returns text stating if a geometry is valid, or if invalid a reason why.

Useful in combination with [ST\\_IsValid](#) to generate a detailed report of invalid geometries and reasons.Allowed flags are documented in [ST\\_IsValidDetail](#).

Performed by the GEOS module.

Availability: 1.4

Availability: 2.0 version taking flags.

**Examples**

```
-- invalid bow-tie polygon
SELECT ST_IsValidReason(
  'POLYGON ((100 200, 100 100, 200 200,
    200 100, 100 200))'::geometry) as validity_info;
validity_info
-----
Self-intersection[150 150]
```

```
--First 3 Rejects from a successful quintuplet experiment
SELECT gid, ST_IsValidReason(geom) as validity_info
FROM
  (SELECT ST_MakePolygon(ST_ExteriorRing(e.buff), array_agg(f.line)) As geom, gid
  FROM (SELECT ST_Buffer(ST_Point(x1*10,y1), z1) As buff, x1*10 + y1*100 + z1*1000 As gid
    FROM generate_series(-4,6) x1
    CROSS JOIN generate_series(2,5) y1
    CROSS JOIN generate_series(1,8) z1
    WHERE x1 > y1*0.5 AND z1 < x1*y1) As e
    INNER JOIN (SELECT ST_Translate(ST_ExteriorRing(ST_Buffer(ST_Point(x1*10,y1), z1)), ←
      y1*1, z1*2) As line
    FROM generate_series(-3,6) x1
    CROSS JOIN generate_series(2,5) y1
    CROSS JOIN generate_series(1,10) z1
    WHERE x1 > y1*0.75 AND z1 < x1*y1) As f
  ON (ST_Area(e.buff) > 78 AND ST_Contains(e.buff, f.line))
  GROUP BY gid, e.buff) As quintuplet_experiment
WHERE ST_IsValid(geom) = false
ORDER BY gid
LIMIT 3;

gid |          validity_info
```

```
-----+-----
5330 | Self-intersection [32 5]
5340 | Self-intersection [42 5]
5350 | Self-intersection [52 5]

--simple example
SELECT ST_IsValidReason('LINESTRING(220227 150406,2220227 150407,222020 150410)');

st_isvalidreason
-----
Valid Geometry
```

## See Also

[ST\\_IsValid](#), [ST\\_Summary](#)

## 8.6.4 ST\_MakeValid

`ST_MakeValid` — Attempts to make an invalid geometry valid without losing vertices.

### Synopsis

```
geometry ST_MakeValid(geometry input);
geometry ST_MakeValid(geometry input, text params);
```

### Description

The function attempts to create a valid representation of a given invalid geometry without losing any of the input vertices. Valid geometries are returned unchanged.

Supported inputs are: POINTS, MULTIPOINTS, LINESTRINGS, MULTILINESTRINGS, POLYGONS, MULTIPOLYGONS and GEOMETRYCOLLECTIONS containing any mix of them.

In case of full or partial dimensional collapses, the output geometry may be a collection of lower-to-equal dimension geometries, or a geometry of lower dimension.

Single polygons may become multi-geometries in case of self-intersections.

The `params` argument can be used to supply an options string to select the method to use for building valid geometry. The options string is in the format "method=linework|structure keepcollapsed=true|false".

The "method" key has two values.

- "linework" is the original algorithm, and builds valid geometries by first extracting all lines, noding that linework together, then building a value output from the linework.
- "structure" is an algorithm that distinguishes between interior and exterior rings, building new geometry by unioning exterior rings, and then differencing all interior rings.

The "keepcollapsed" key is only valid for the "structure" algorithm, and takes a value of "true" or "false". When set to "false", geometry components that collapse to a lower dimensionality, for example a one-point linestring would be dropped.

Performed by the GEOS module.

Availability: 2.0.0

Enhanced: 2.0.1, speed improvements

Enhanced: 2.1.0, added support for GEOMETRYCOLLECTION and MULTIPOINT.

Enhanced: 3.1.0, added removal of Coordinates with NaN values.

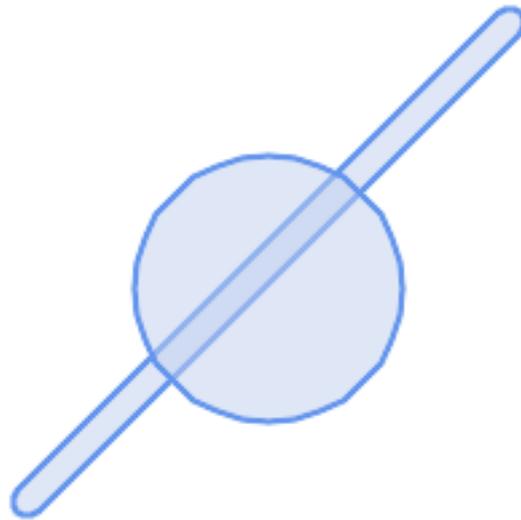
Enhanced: 3.2.0, added algorithm options, 'linework' and 'structure' which requires GEOS  $\geq$  3.10.0.



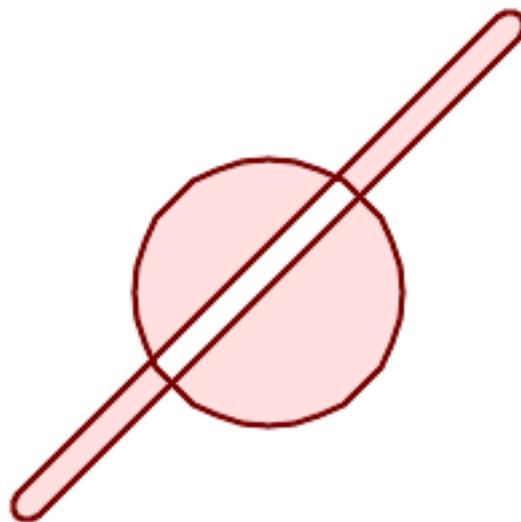
This function supports 3d and will not drop the z-index.

## Examples

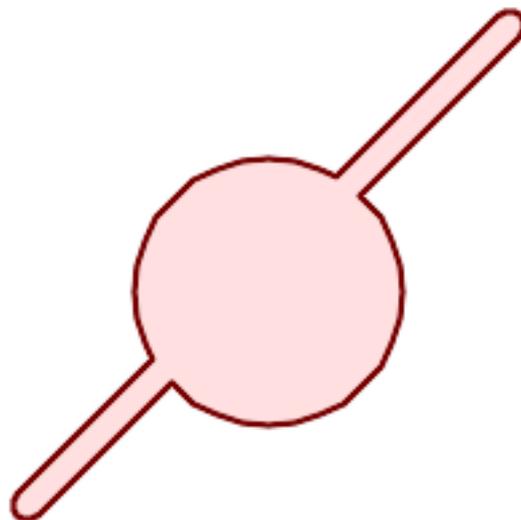
---



*before\_geom: MULTIPOLYGON of 2 overlapping polygons*



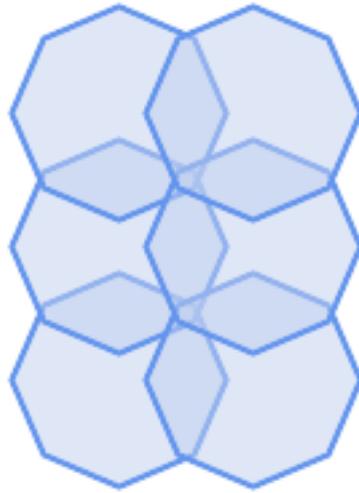
*after\_geom: MULTIPOLYGON of 4 non-overlapping polygons*



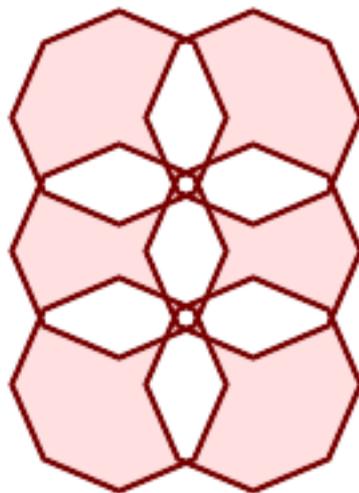
*after\_geom\_structure: MULTIPOLYGON of 1 non-overlapping polygon*

```
SELECT f.geom AS before_geom, ST_MakeValid(f.geom) AS after_geom, ST_MakeValid(f.geom, ←
    'method=structure') AS after_geom_structure
FROM (SELECT 'MULTIPOLYGON(((186 194,187 194,188 195,189 195,190 195,
191 195,192 195,193 194,194 194,194 194,193 195,192 195,191
```

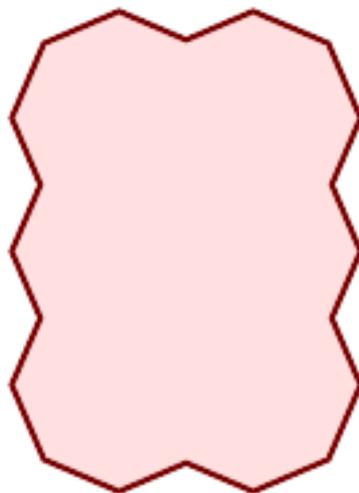




*before\_geom: MULTIPOLYGON of 6 overlapping polygons*



*after\_geom: MULTIPOLYGON of 14 Non-overlapping polygons*



*after\_geom\_structure: MULTIPOLYGON of 1 Non-overlapping polygon*

```
SELECT c.geom AS before_geom,
       ST_MakeValid(c.geom) AS after_geom,
       ST_MakeValid(c.geom, 'method=structure') AS after_geom_structure
FROM (SELECT 'MULTIPOLYGON(((91 50,79 22,51 10,23 22,11 50,23 78,51 90,79 78,91 ↵
```

## Examples

```
SELECT ST_AsText(ST_MakeValid(
  'LINESTRING(0 0, 0 0)',
  'method=structure keepcollapsed=true'
));

st_astext
-----
POINT(0 0)

SELECT ST_AsText(ST_MakeValid(
  'LINESTRING(0 0, 0 0)',
  'method=structure keepcollapsed=false'
));

st_astext
-----
LINESTRING EMPTY
```

## See Also

[ST\\_IsValid](#), [ST\\_GeomCollFromText](#), [ST\\_CollectionExtract](#)

## 8.7 Spatial Reference System Functions

### 8.7.1 ST\_SetSRID

`ST_SetSRID` — Set the SRID on a geometry.

#### Synopsis

geometry **ST\_SetSRID**(geometry geom, integer srid);

#### Description

Sets the SRID on a geometry to a particular integer value. Useful in constructing bounding boxes for queries.



#### Note

This function does not transform the geometry coordinates in any way - it simply sets the meta data defining the spatial reference system the geometry is assumed to be in. Use [ST\\_Transform](#) if you want to transform the geometry into a new projection.



This method implements the [OGC Simple Features Implementation Specification for SQL 1.1](#).



This method supports Circular Strings and Curves

## Examples

-- Mark a point as WGS 84 long lat --

```
SELECT ST_SetSRID(ST_Point(-123.365556, 48.428611),4326) As wgs84long_lat;
-- the ewkt representation (wrap with ST_AsEWKT) -
SRID=4326;POINT(-123.365556 48.428611)
```

-- Mark a point as WGS 84 long lat and then transform to web mercator (Spherical Mercator) --

```
SELECT ST_Transform(ST_SetSRID(ST_Point(-123.365556, 48.428611),4326),3785) As spere_merc;
-- the ewkt representation (wrap with ST_AsEWKT) -
SRID=3785;POINT(-13732990.8753491 6178458.96425423)
```

## See Also

Section [4.5](#), [ST\\_SRID](#), [ST\\_Transform](#), [UpdateGeometrySRID](#)

## 8.7.2 ST\_SRID

**ST\_SRID** — Returns the spatial reference identifier for a geometry.

### Synopsis

integer **ST\_SRID**(geometry g1);

### Description

Returns the spatial reference identifier for the ST\_Geometry as defined in spatial\_ref\_sys table. Section [4.5](#)



#### Note

spatial\_ref\_sys table is a table that catalogs all spatial reference systems known to PostGIS and is used for transformations from one spatial reference system to another. So verifying you have the right spatial reference system identifier is important if you plan to ever transform your geometries.



This method implements the [OGC Simple Features Implementation Specification for SQL 1.1](#). s2.1.1.1



This method implements the SQL/MM specification. SQL-MM 3: 5.1.5



This method supports Circular Strings and Curves

## Examples

```
SELECT ST_SRID(ST_GeomFromText('POINT(-71.1043 42.315)',4326));
--result
4326
```

## See Also

Section [4.5](#), [ST\\_SetSRID](#), [ST\\_Transform](#), [ST\\_SRID](#), [ST\\_SRID](#)

### 8.7.3 ST\_Transform

ST\_Transform — Return a new geometry with coordinates transformed to a different spatial reference system.

#### Synopsis

```
geometry ST_Transform(geometry g1, integer srid);
geometry ST_Transform(geometry geom, text to_proj);
geometry ST_Transform(geometry geom, text from_proj, text to_proj);
geometry ST_Transform(geometry geom, text from_proj, integer to_srid);
```

#### Description

Returns a new geometry with its coordinates transformed to a different spatial reference system. The destination spatial reference `to_srid` may be identified by a valid SRID integer parameter (i.e. it must exist in the `spatial_ref_sys` table). Alternatively, a spatial reference defined as a PROJ.4 string can be used for `to_proj` and/or `from_proj`, however these methods are not optimized. If the destination spatial reference system is expressed with a PROJ.4 string instead of an SRID, the SRID of the output geometry will be set to zero. With the exception of functions with `from_proj`, input geometries must have a defined SRID.

ST\_Transform is often confused with [ST\\_SetSRID](#). ST\_Transform actually changes the coordinates of a geometry from one spatial reference system to another, while ST\_SetSRID() simply changes the SRID identifier of the geometry.

**Note**

Requires PostGIS be compiled with PROJ support. Use [PostGIS\\_Full\\_Version](#) to confirm you have PROJ support compiled in.

**Note**

If using more than one transformation, it is useful to have a functional index on the commonly used transformations to take advantage of index usage.

**Note**

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

Enhanced: 2.0.0 support for Polyhedral surfaces was introduced.

Enhanced: 2.3.0 support for direct PROJ.4 text was introduced.



This method implements the SQL/MM specification. SQL-MM 3: 5.1.6



This method supports Circular Strings and Curves



This function supports Polyhedral surfaces.

#### Examples

Change Massachusetts state plane US feet geometry to WGS 84 long lat

```

SELECT ST_AsText(ST_Transform(ST_GeomFromText('POLYGON((743238 2967416,743238 2967450,
      743265 2967450,743265.625 2967416,743238 2967416)'),2249),4326)) As wgs_geom;

wgs_geom
-----
POLYGON((-71.1776848522251 42.3902896512902,-71.1776843766326 42.3903829478009,
-71.1775844305465 42.3903826677917,-71.1775825927231 42.3902893647987,-71.177684
8522251 42.3902896512902));
(1 row)

--3D Circular String example
SELECT ST_AsEWKT(ST_Transform(ST_GeomFromEWKT('SRID=2249;CIRCULARSTRING(743238 2967416 ↵
      1,743238 2967450 2,743265 2967450 3,743265.625 2967416 3,743238 2967416 4)'),4326));

st_asewkt
-----
SRID=4326;CIRCULARSTRING(-71.1776848522251 42.3902896512902 1,-71.1776843766326 ↵
      42.3903829478009 2,
-71.1775844305465 42.3903826677917 3,
-71.1775825927231 42.3902893647987 3,-71.1776848522251 42.3902896512902 4)

```

Example of creating a partial functional index. For tables where you are not sure all the geometries will be filled in, its best to use a partial index that leaves out null geometries which will both conserve space and make your index smaller and more efficient.

```

CREATE INDEX idx_geom_26986_parcel
ON parcels
USING gist
(ST_Transform(geom, 26986))
WHERE geom IS NOT NULL;

```

Examples of using PROJ.4 text to transform with custom spatial references.

```

-- Find intersection of two polygons near the North pole, using a custom Gnomonic projection
-- See http://boundlessgeo.com/2012/02/flattening-the-peel/
WITH data AS (
  SELECT
    ST_GeomFromText('POLYGON((170 50,170 72,-130 72,-130 50,170 50)'), 4326) AS p1,
    ST_GeomFromText('POLYGON((-170 68,-170 90,-141 90,-141 68,-170 68)'), 4326) AS p2,
    '+proj=gnom +ellps=WGS84 +lat_0=70 +lon_0=-160 +no_defs'::text AS gnom
)
SELECT ST_AsText(
  ST_Transform(
    ST_Intersection(ST_Transform(p1, gnom), ST_Transform(p2, gnom)),
    gnom, 4326))
FROM data;

st_astext
-----
POLYGON((-170 74.053793645338,-141 73.4268621378904,-141 68,-170 68,-170 74.053793645338) ↵
)

```

### Configuring transformation behavior

Sometimes coordinate transformation involving a grid-shift can fail, for example if PROJ.4 has not been built with grid-shift files or the coordinate does not lie within the range for which the grid shift is defined. By default, PostGIS will throw an error if a grid shift file is not present, but this behavior can be configured on a per-SRID basis either by testing different `to_proj` values of PROJ.4 text, or altering the `proj4text` value within the `spatial_ref_sys` table.

For example, the `proj4text` parameter `+datum=NAD87` is a shorthand form for the following `+nadgrids` parameter:

```
+nadgrids=@conus,@alaska,@ntv2_0.gsb,@ntv1_can.dat
```

The @ prefix means no error is reported if the files are not present, but if the end of the list is reached with no file having been appropriate (ie. found and overlapping) then an error is issued.

If, conversely, you wanted to ensure that at least the standard files were present, but that if all files were scanned without a hit a null transformation is applied you could use:

```
+nadgrids=@conus,@alaska,@ntv2_0.gsb,@ntv1_can.dat,null
```

The null grid shift file is a valid grid shift file covering the whole world and applying no shift. So for a complete example, if you wanted to alter PostGIS so that transformations to SRID 4267 that didn't lie within the correct range did not throw an ERROR, you would use the following:

```
UPDATE spatial_ref_sys SET proj4text = '+proj=longlat +ellps=clrk66 +nadgrids=@conus, ↵
@alaska,@ntv2_0.gsb,@ntv1_can.dat,null +no_defs' WHERE srid = 4267;
```

### See Also

Section 4.5, [ST\\_SetSRID](#), [ST\\_SRID](#), [UpdateGeometrySRID](#)

## 8.8 Geometry Input

### 8.8.1 Well-Known Text (WKT)

#### 8.8.1.1 ST\_BdPolyFromText

ST\_BdPolyFromText — [ST\\_BdPolyFromText](#) (text WKT, integer srid);

#### Synopsis

geometry **ST\_BdPolyFromText**(text WKT, integer srid);

**Notes:**

[ST\\_BdPolyFromText](#) (text WKT, integer srid);

#### Note



WKT('POLYGON((0 0, 1 0, 1 1, 0 1, 0 0))'); -- returns POLYGON((0 0, 1 0, 1 1, 0 1, 0 0))

This method implements the [OGC Simple Features Implementation Specification for SQL 1.1](#). s3.2.6.2

GEOS [ST\\_BdPolyFromText](#)(text WKT, integer srid);

1.1.0 [ST\\_BdPolyFromText](#)(text WKT, integer srid);

**ST\_BuildArea, ST\_BdMPolyFromText**

**ST\_BuildArea, ST\_BdMPolyFromText**

### 8.8.1.2 ST\_BdMPolyFromText

ST\_BdMPolyFromText — WKT geometry text, integer srid;

#### Synopsis

geometry ST\_BdMPolyFromText(text WKT, integer srid);

**Note**

WKT geometry text, integer srid;

#### Note

WKT geometry text, integer srid; ST\_BuildArea()



 This method implements the [OGC Simple Features Implementation Specification for SQL 1.1](#) s3.2.6.2

GEOS

1.1.0

**ST\_BuildArea, ST\_BdPolyFromText**

**ST\_BuildArea, ST\_BdPolyFromText**

### 8.8.1.3 ST\_GeogFromText

ST\_GeogFromText — WKT geometry text, integer srid;

#### Synopsis

geography ST\_GeogFromText(text EWKT);

**WKT**

```
WKT ('SRID=4326;POINT(-77.0092 38.889588)');
```

**SQL**

```
--- SRID=4326;POINT(-77.0092 38.889588) ←
ALTER TABLE sometable ADD COLUMN geog geography(POINT, 4326);
UPDATE sometable SET geog = ST_GeogFromText('SRID=4326;POINT(-77.0092 38.889588)');

--- EPSG:4267, NAD27(1116614.831146, 5623592.771422) ←
SELECT ST_AsEWKT(ST_GeogFromText('SRID=4267;POINT(-77.0092 38.889588)'));
```

**ST\_GeogFromText**

[ST\\_AsText](#), [ST\\_GeographyFromText](#)

**8.8.1.4 ST\_GeographyFromText**

**ST\_GeographyFromText** — WKT ('SRID=4326;POINT(-77.0092 38.889588)')

**Synopsis**

geography **ST\_GeographyFromText**(text EWKT);

**WKT**

```
WKT ('SRID=4326;POINT(-77.0092 38.889588)');
```

**SQL**

[ST\\_GeogFromText](#), [ST\\_AsText](#)

**8.8.1.5 ST\_GeomCollFromText**

**ST\_GeomCollFromText** — Makes a collection Geometry from collection WKT with the given SRID. If SRID is not given, it defaults to 0.

**Synopsis**

geometry **ST\_GeomCollFromText**(text WKT, integer srid);  
 geometry **ST\_GeomCollFromText**(text WKT);

**ST\_GeomFromText**

Makes a collection Geometry from the Well-Known-Text (WKT) representation with the given SRID. If SRID is not given, it defaults to 0.

OGC [3.2.6.2 - Simple Features Implementation Specification for SQL](#); SRID; suite);

WKT; null;

**Note**

WKT; SRID; ST\_GeomFromText;



This method implements the [OGC Simple Features Implementation Specification for SQL 1.1](#). s3.2.6.2



This method implements the SQL/MM specification.

**ST\_GeomCollFromText**

```
SELECT ST_GeomCollFromText('GEOMETRYCOLLECTION(POINT(1 2),LINESTRING(1 2, 3 4))');
```

**ST\_GeomFromText, ST\_SRID****ST\_GeomFromText, ST\_SRID****8.8.1.6 ST\_GeomFromEWKT**

ST\_GeomFromEWKT — EWKT(Extended Well-Known Text); ST\_GeomFromText;

**Synopsis**

geometry **ST\_GeomFromEWKT**(text EWKT);

**ST\_GeomFromEWKT**

OGC EWKT(Extended Well-Known Text); PostGIS ST\_GeomFromEWKT;

**Note**

EWKT; OGC; SRID; PostGIS;

2.0.0 and will not drop the z-index. surface) TIN;

- ✓ This function supports 3d and will not drop the z-index.
- ✓ This method supports Circular Strings and Curves
- ✓ This function supports Polyhedral surfaces.
- ✓ This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).

### Examples

```
SELECT ST_GeomFromEWKT('SRID=4269;LINESTRING(-71.160281 42.258729,-71.160837 42.259113,-71.161144 42.25932)');
SELECT ST_GeomFromEWKT('SRID=4269;MULTILINESTRING((-71.160281 42.258729,-71.160837 42.259113,-71.161144 42.25932)');

SELECT ST_GeomFromEWKT('SRID=4269;POINT(-71.064544 42.28787)');

SELECT ST_GeomFromEWKT('SRID=4269;POLYGON((-71.1776585052917 42.3902909739571,-71.1776820268866 42.3903701743239,-71.1776063012595 42.3903825660754,-71.1775826583081 42.3903033653531,-71.1776585052917 42.3902909739571)');

SELECT ST_GeomFromEWKT('SRID=4269;MULTIPOLYGON((( -71.1031880899493 42.3152774590236, -71.1031627617667 42.3152960829043,-71.102923838298 42.3149156848307, -71.1023097974109 42.3151969047397,-71.1019285062273 42.3147384934248, -71.102505233663 42.3144722937587,-71.10277487471 42.3141658254797, -71.103113945163 42.3142739188902,-71.10324876416 42.31402489987, -71.1033002961013 42.3140393340215,-71.1033488797549 42.3139495090772, -71.103396240451 42.3138632439557,-71.1041521907712 42.3141153348029, -71.1041411411543 42.3141545014533,-71.1041287795912 42.3142114839058, -71.1041188134329 42.3142693656241,-71.1041112482575 42.3143272556118, -71.1041072845732 42.3143851580048,-71.1041057218871 42.3144430686681, -71.1041065602059 42.3145009876017,-71.1041097995362 42.3145589148055, -71.1041166403905 42.3146168544148,-71.1041258822717 42.3146748022936, -71.1041375307579 42.3147318674446,-71.1041492906949 42.3147711126569, -71.1041598612795 42.314808571739,-71.1042515013869 42.3151287620809, -71.1041173835118 42.3150739481917,-71.1040809891419 42.3151344119048, -71.1040438678912 42.3151191367447,-71.1040194562988 42.3151832057859, -71.1038734225584 42.3151140942995,-71.1038446938243 42.3151006300338, -71.1038315271889 42.315094347535,-71.1037393329282 42.315054824985, -71.1035447555574 42.3152608696313,-71.1033436658644 42.3151648370544, -71.1032580383161 42.3152269126061,-71.103223066939 42.3152517403219, -71.1031880899493 42.3152774590236)), ((-71.1043632495873 42.315113108546,-71.1043583974082 42.3151211109857,-71.1043443253471 42.3150676015829,-71.1043850704575 42.3150793250568,-71.1043632495873 42.315113108546)))');
```

```
-- 3D CircularString
SELECT ST_GeomFromEWKT('CIRCULARSTRING(220268 150415 1,220227 150505 2,220227 150406 3)');
```

```
-- PolyhedralSurface
SELECT ST_GeomFromEWKT('POLYHEDRALSURFACE(
  ((0 0 0, 0 0 1, 0 1 1, 0 1 0, 0 0 0)),
  ((0 0 0, 0 1 0, 1 1 0, 1 0 0, 0 0 0)),
  ((0 0 0, 1 0 0, 1 0 1, 0 0 1, 0 0 0)),
  ((1 1 0, 1 1 1, 1 0 1, 1 0 0, 1 1 0)),
```

```
((0 1 0, 0 1 1, 1 1 1, 1 1 0, 0 1 0)),
((0 0 1, 1 0 1, 1 1 1, 0 1 1, 0 0 1))
) ');
```

**ST\_AsEWKT, ST\_GeomFromText, ST\_GeomFromEWKT**

**ST\_AsEWKT, ST\_GeomFromText, ST\_GeomFromEWKT**

### 8.8.1.7 ST\_GeomFromMARC21

**ST\_GeomFromMARC21** — Takes MARC21/XML geographic data as input and returns a PostGIS geometry object.

#### Synopsis

geometry **ST\_GeomFromMARC21** ( text marcxml );

**ST\_GeomFromMARC21**

This function creates a PostGIS geometry from a MARC21/XML record, which can contain a POINT or a POLYGON. In case of multiple geographic data entries in the same MARC21/XML record, a MULTIPOINT or MULTIPOLYGON will be returned. If the record contains mixed geometry types, a GEOMETRYCOLLECTION will be returned. It returns NULL if the MARC21/XML record does not contain any geographic data (datafield:034).

LOC MARC21/XML versions supported:

- [MARC21/XML 1.1](#)

Availability: 3.3.0, requires libxml2 2.6+



#### Note

The MARC21/XML Coded Cartographic Mathematical Data currently does not provide any means to describe the Spatial Reference System of the encoded coordinates, so this function will always return a geometry with SRID 0.



#### Note

Returned POLYGON geometries will always be clockwise oriented.

**ST\_GeomFromMARC21**

Converting MARC21/XML geographic data containing a single POINT encoded as hddd . dddddd

```
SELECT
    ST_AsText (
        ST_GeomFromMARC21 ('
            <record xmlns="http://www.loc.gov/MARC21/slim">
                <leader>00000nz a2200000nc 4500</leader>
                <controlfield tag="001">040277569</controlfield>
                <datafield tag="034" ind1=" " ind2=" ">
                    <subfield code="d">W004.500000</subfield>
                    <subfield code="e">W004.500000</subfield>
                    <subfield code="f">N054.250000</subfield>
                    <subfield code="g">N054.250000</subfield>
                </datafield>
            </record>
        ')
```

```

        </datafield>
    </record>');

st_astext
-----
POINT(-4.5 54.25)
(1 row)

```

### Converting MARC21/XML geographic data containing a single POLYGON encoded as hdddmms

```

SELECT
    ST_AsText (
        ST_GeomFromMARC21 ('
            <record xmlns="http://www.loc.gov/MARC21/slim">
                <leader>01062cem a2200241 a 4500</leader>
                <controlfield tag="001"> 84696781 </controlfield>
                <datafield tag="034" ind1="1" ind2=" ">
                    <subfield code="a">a</subfield>
                    <subfield code="b">50000</subfield>
                    <subfield code="d">E0130600</subfield>
                    <subfield code="e">E0133100</subfield>
                    <subfield code="f">N0523900</subfield>
                    <subfield code="g">N0522300</subfield>
                </datafield>
            </record>');

    st_astext
    -----
POLYGON((13.1 52.65,13.516666666666667 52.65,13.516666666666667 ←
        52.38333333333333,13.1 52.38333333333333,13.1 52.65))
(1 row)

```

### Converting MARC21/XML geographic data containing a POLYGON and a POINT:

```

SELECT
    ST_AsText (
        ST_GeomFromMARC21 ('
            <record xmlns="http://www.loc.gov/MARC21/slim">
                <datafield tag="034" ind1="1" ind2=" ">
                    <subfield code="a">a</subfield>
                    <subfield code="b">50000</subfield>
                    <subfield code="d">E0130600</subfield>
                    <subfield code="e">E0133100</subfield>
                    <subfield code="f">N0523900</subfield>
                    <subfield code="g">N0522300</subfield>
                </datafield>
                <datafield tag="034" ind1=" " ind2=" ">
                    <subfield code="d">W004.500000</subfield>
                    <subfield code="e">W004.500000</subfield>
                    <subfield code="f">N054.250000</subfield>
                    <subfield code="g">N054.250000</subfield>
                </datafield>
            </record>');

    st_astext ←
    -----
GEOMETRYCOLLECTION(POLYGON((13.1 52.65,13.516666666666667 ←
        52.65,13.516666666666667 52.38333333333333,13.1 52.38333333333333,13.1 ←
        52.65)),POINT(-4.5 54.25))

```



- ✔ This method implements the [OGC Simple Features Implementation Specification for SQL 1.1](#), 3.2.6.2 - SRID; (conformance suite);
- ✔ This method implements the SQL/MM specification. SQL-MM 3: 5.1.40
- ✔ This method supports Circular Strings and Curves

**Note**

While not OGC-compliant, `ST_MakePoint` is faster than `ST_GeomFromText` and `ST_PointFromText`. It is also easier to use for numeric coordinate values. `ST_Point` is another option similar in speed to `ST_MakePoint` and is OGC-compliant, but doesn't support anything but 2D points.

**Warning**

PostGIS 2.0.0: `ST_GeomFromText('GEOMETRYCOLLECTION(EMPTY)')` PostGIS 2.0.0: `SQL/MM`

**Example**

```
SELECT ST_GeomFromText('LINESTRING(-71.160281 42.258729,-71.160837 42.259113,-71.161144 42.25932)');
SELECT ST_GeomFromText('LINESTRING(-71.160281 42.258729,-71.160837 42.259113,-71.161144 42.25932)',4269);

SELECT ST_GeomFromText('MULTILINESTRING((-71.160281 42.258729,-71.160837 42.259113,-71.161144 42.25932))');

SELECT ST_GeomFromText('POINT(-71.064544 42.28787)');

SELECT ST_GeomFromText('POLYGON((-71.1776585052917 42.3902909739571,-71.1776820268866 42.3903701743239,
-71.1776063012595 42.3903825660754,-71.1775826583081 42.3903033653531,-71.1776585052917 42.3902909739571))');

SELECT ST_GeomFromText('MULTIPOLYGON((( -71.1031880899493 42.3152774590236,
-71.1031627617667 42.3152960829043,-71.102923838298 42.3149156848307,
-71.1023097974109 42.3151969047397,-71.1019285062273 42.3147384934248,
-71.102505233663 42.3144722937587,-71.10277487471 42.3141658254797,
-71.103113945163 42.3142739188902,-71.10324876416 42.31402489987,
-71.1033002961013 42.3140393340215,-71.1033488797549 42.3139495090772,
-71.103396240451 42.3138632439557,-71.1041521907712 42.3141153348029,
-71.1041411411543 42.3141545014533,-71.1041287795912 42.3142114839058,
-71.1041188134329 42.3142693656241,-71.1041112482575 42.3143272556118,
-71.1041072845732 42.3143851580048,-71.1041057218871 42.3144430686681,
-71.1041065602059 42.3145009876017,-71.1041097995362 42.3145589148055,
-71.1041166403905 42.3146168544148,-71.1041258822717 42.3146748022936,
-71.1041375307579 42.3147318674446,-71.1041492906949 42.3147711126569,
-71.1041598612795 42.314808571739,-71.1042515013869 42.3151287620809,
-71.1041173835118 42.3150739481917,-71.1040809891419 42.3151344119048,
-71.1040438678912 42.3151191367447,-71.1040194562988 42.3151832057859,
```



**ST\_LineFromText**

```
SELECT ST_LineFromText('LINESTRING(1 2, 3 4)') AS aline, ST_LineFromText('POINT(1 2)') AS
  null_return;
aline          | null_return
-----
01020000000200000000000000000000F ... | t
```

**ST\_MLineFromText**

**ST\_MLineFromText**

### 8.8.1.11 ST\_MLineFromText

**ST\_MLineFromText** — WKT `LINESTRING(x1 y1, x2 y2, ...)`; SRID `SRID(x1 y1, x2 y2, ...)`; `ST_MultiLineString(x1 y1, x2 y2, ...)`.

#### Synopsis

geometry **ST\_MLineFromText**(text WKT, integer srid);  
 geometry **ST\_MLineFromText**(text WKT);

**ST\_MLineFromText**

Makes a Geometry from Well-Known-Text (WKT) with the given SRID. If SRID is not given, it defaults to 0.

OGC [Simple Features Implementation Specification for SQL 1.1](#): 3.2.6.2 - `SRID(x1 y1, x2 y2, ...)`; `ST_MultiLineString(x1 y1, x2 y2, ...)`.

WKT `LINESTRING(x1 y1, x2 y2, ...)`; SRID `SRID(x1 y1, x2 y2, ...)`; `ST_MultiLineString(x1 y1, x2 y2, ...)`.

#### Note



WKT `LINESTRING(x1 y1, x2 y2, ...)`; SRID `SRID(x1 y1, x2 y2, ...)`; `ST_MultiLineString(x1 y1, x2 y2, ...)`.



This method implements the [OGC Simple Features Implementation Specification for SQL 1.1](#): 3.2.6.2



This method implements the SQL/MM specification. SQL/MM 3: 9.4.4

**ST\_MLineFromText**

```
SELECT ST_MLineFromText('MULTILINESTRING((1 2, 3 4), (4 5, 6 7))');
```

**ST\_MLineFromText**

**ST\_MLineFromText**

### 8.8.1.12 ST\_MPointFromText

ST\_MPointFromText — Makes a Geometry from WKT with the given SRID. If SRID is not given, it defaults to 0.

#### Synopsis

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

#### Notes

Makes a Geometry from WKT with the given SRID. If SRID is not given, it defaults to 0.

OGC 3.2.6.2 - SRID; suite);

WKT; null;.

#### Note



WKT; ST\_GeomFromText;



This method implements the [OGC Simple Features Implementation Specification for SQL 1.1](#). 3.2.6.2



This method implements the SQL/MM specification. SQL-MM 3: 9.2.4

#### Examples

```
SELECT ST_MPointFromText ('MULTIPOINT((1 2), (3 4))');
SELECT ST_MPointFromText ('MULTIPOINT((-70.9590 42.1180), (-70.9611 42.1223))', 4326);
```

#### See also

[ST\\_GeomFromText](#)

### 8.8.1.13 ST\_MPolyFromText

ST\_MPolyFromText — Makes a MultiPolygon Geometry from WKT with the given SRID. If SRID is not given, it defaults to 0.

#### Synopsis

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



**ST\_PointFromText**

Constructs a PostGIS ST\_Geometry point object from the OGC Well-Known text representation. If SRID is not given, it defaults to unknown (currently 0). If geometry is not a WKT point representation, returns null. If completely invalid WKT, then throws an error.

**Note**

ST\_PointFromText(*geom\_text*, [*SRID*], [*spatial\_ref\_sys*])  
 Returns a geometry object from the OGC Well-Known Text (WKT) representation. If SRID is not given, it defaults to unknown (currently 0). If geometry is not a WKT point representation, returns null. If completely invalid WKT, then throws an error.


**Note**

ST\_GeomFromText(*geom\_text*, [*SRID*], [*spatial\_ref\_sys*])  
 Returns a geometry object from the OGC Well-Known Text (WKT) representation. If SRID is not given, it defaults to unknown (currently 0). If geometry is not a WKT point representation, returns null. If completely invalid WKT, then throws an error.



 This method implements the [OGC Simple Features Implementation Specification for SQL 1.1](#), 3.2.6.2 - *POINT* (SRID, *geom\_text*); *SRID*, *geom\_text* (conformance suite)

 This method implements the SQL/MM specification. SQL-MM 3: 6.1.8

**ST\_GeomFromText**

```
SELECT ST_GeomFromText ('POINT (-71.064544 42.28787) ');
SELECT ST_GeomFromText ('POINT (-71.064544 42.28787)', 4326);
```

**ST\_GeomFromText, ST\_MakePoint, ST\_Point, ST\_SRID**

[ST\\_GeomFromText](#), [ST\\_MakePoint](#), [ST\\_Point](#), [ST\\_SRID](#)

**8.8.1.15 ST\_PolygonFromText**

ST\_PolygonFromText — Makes a Geometry from WKT with the given SRID. If SRID is not given, it defaults to 0.

## Synopsis

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

### Notes

Makes a Geometry from WKT with the given SRID. If SRID is not given, it defaults to 0. Returns null if WKT is not a polygon.

OGC Simple Features Implementation Specification for SQL 1.1, 3.2.6.2 - SRID

#### Note



WKT SRID

```
ST_GeomFromText('POLYGON((-71.1776585052917 42.3902909739571,-71.1776820268866 42.3903701743239,-71.1776063012595 42.3903825660754,-71.1775826583081 42.3903033653531,-71.1776585052917 42.3902909739571))');
```

ST\_GeomFromText('POINT(1 2)')



This method implements the [OGC Simple Features Implementation Specification for SQL 1.1, 3.2.6.2](#)



This method implements the SQL/MM specification. SQL-MM 3: 8.3.6

### Examples

```
SELECT ST_PolygonFromText('POLYGON((-71.1776585052917 42.3902909739571,-71.1776820268866 42.3903701743239,-71.1776063012595 42.3903825660754,-71.1775826583081 42.3903033653531,-71.1776585052917 42.3902909739571))');
st_polygonfromtext
-----
010300000001000000050000006...
```

```
SELECT ST_PolygonFromText('POINT(1 2)') IS NULL as point_is_notpoly;
point_is_not_poly
-----
t
```

### ST\_GeomFromText

#### ST\_GeomFromText

#### 8.8.1.16 ST\_WKTToSQL

ST\_WKTToSQL — WKT(Well-Known Text) to SQL Geometry

## Synopsis

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

**ST\_GeogFromText**



This method implements the SQL/MM specification. SQL-MM 3: 5.1.34

**ST\_GeogFromWKB**

**ST\_GeogFromText**

## 8.8.2 Well-Known Binary (WKB)

### 8.8.2.1 ST\_GeogFromWKB

**ST\_GeogFromWKB** — WKB (Well-Known Binary) or EWKB (Extended Well-Known Binary) representation of a geometry.

#### Synopsis

geography **ST\_GeogFromWKB**(bytea wkb);

**ST\_GeogFromWKB**

**ST\_GeogFromWKB** (bytea wkb) returns a geography. The input wkb must be in SRID 4326 (WGS84) or SRID 31466 (NAD83). The input wkb must be in the same SRID as the output geography. The input wkb must be in the same SRID as the output geography. The input wkb must be in the same SRID as the output geography.

SRID 4326 (WGS84) or SRID 31466 (NAD83). The input wkb must be in the same SRID as the output geography. The input wkb must be in the same SRID as the output geography.



This method supports Circular Strings and Curves

**ST\_AsText**

```
-- bytea representation of a geometry
SELECT ST_AsText(
  ST_GeogFromWKB(E'\\001\\002\\000\\000\\000\\000\\002\\000\\000\\000\\037\\205\\353Q
  \\270~\\\\\\\\300\\323Mb\\020X\\231C@\\020X9\\264\\310~\\\\\\\\300)\\\\\\\\217\\302\\365\\230
  C@')
);
          st_astext
-----
LINESTRING(-113.98 39.198,-113.981 39.195)
(1 row)
```

**ST\_GeogFromText, ST\_AsBinary**

**ST\_GeogFromText, ST\_AsBinary**

### 8.8.2.2 ST\_GeomFromEWKB

ST\_GeomFromEWKB — EWKB(Extended Well-Known Binary) ST\_Geometry

#### Synopsis

geometry ST\_GeomFromEWKB(bytea EWKB);

#### Note

OGC EWKB(Extended Well-Known Binary) PostGIS ST\_Geom



#### Note

EWKB OGC SRID PostGIS

2.0.0 surface) TIN

- This function supports 3d and will not drop the z-index.
- This method supports Circular Strings and Curves
- This function supports Polyhedral surfaces.
- This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).

#### Note

NAD83 (SRID 4269) LINESTRING(-71.160281 42.258729,-71.160837 42.259113,-71.161144 42.25932)



#### Note

AsEWKB

```
SELECT ST_GeomFromEWKB(E'\\001\\002\\000\\000 \\255\\020\\000\\000\\003\\000\\000\\000\\344 ←
J=
\\013B\\312Q\\300n\\303(\\010\\036!E@' '\\277E' 'K
\\312Q\\300\\366{b\\235*!E@\\225|\\354.P\\312Q
\\300p\\231\\323e1!E@') ;
```







**Note**

`ST_GeomFromWKB` (bytea geom, integer srid);

`ST_GeomFromWKB` (bytea geom, integer srid);



This method implements the [OGC Simple Features Implementation Specification for SQL 1.1](#). s3.2.6.2



This method implements the SQL/MM specification. SQL-MM 3: 7.2.9

**Example**

```
SELECT
  ST_LineStringFromWKB (
    ST_AsBinary (ST_GeomFromText ('LINESTRING(1 2, 3 4)'))
  ) AS aline,
  ST_LineStringFromWKB (
    ST_AsBinary (ST_GeomFromText ('POINT(1 2)'))
  ) IS NULL AS null_return;
-----
01020000000200000000000000000000F ... | t
```

**Example**

`ST_GeomFromWKB`, `ST_LineFromWKB`

**8.8.2.6 ST\_PointFromWKB**

`ST_PointFromWKB` (bytea geom, integer srid);

**Synopsis**

geometry `ST_GeomFromWKB`(bytea geom);

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

**Example**

`ST_PointFromWKB` (bytea geom, integer srid);

`ST_PointFromWKB` (bytea geom, integer srid);



This method implements the [OGC Simple Features Implementation Specification for SQL 1.1](#). s3.2.7.2

-  This method implements the SQL/MM specification. SQL-MM 3: 6.1.9
-  This function supports 3d and will not drop the z-index.
-  This method supports Circular Strings and Curves

### ST\_AsText

```
SELECT
  ST_AsText (
    ST_PointFromWKB (
      ST_AsEWKB ('POINT(2 5) '::geometry)
    )
  );
st_astext
-----
POINT(2 5)
(1 row)

SELECT
  ST_AsText (
    ST_PointFromWKB (
      ST_AsEWKB ('LINESTRING(2 5, 2 6) '::geometry)
    )
  );
st_astext
-----
(1 row)
```

### ST\_GeomFromWKB, ST\_LineFromWKB

**ST\_GeomFromWKB, ST\_LineFromWKB**

#### 8.8.2.7 ST\_WKBToSQL

**ST\_WKBToSQL** — WKB(Well-Known Binary) `ST_WKBToSQL(geometry ST_Geometry, SRID, SRID, ST_GeomFromWKB)`

#### Synopsis

geometry **ST\_WKBToSQL**(bytea WKB);

### ST\_GeomFromWKB

-  This method implements the SQL/MM specification. SQL-MM 3: 5.1.36

### ST\_GeomFromWKB

**ST\_GeomFromWKB**

## 8.8.3 Other Formats

### 8.8.3.1 ST\_Box2dFromGeoHash

`ST_Box2dFromGeoHash` — GeoHash BOX2D based on full precision of the input GeoHash string.

#### Synopsis

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

#### GeoHash BOX2D

GeoHash BOX2D based on full precision of the input GeoHash string.

If no `precision` is specified `ST_Box2dFromGeoHash` returns a BOX2D based on full precision of the input GeoHash string.

```
precision ST_Box2dFromGeoHash(text geohash, integer precision=full_precision_of_geohash);
BOX2D ST_Box2dFromGeoHash(text geohash, integer precision=full_precision_of_geohash);
BOX2D ST_Box2dFromGeoHash(text geohash, integer precision=full_precision_of_geohash, integer precision=full_precision_of_geohash);
```

2.1.0 BOX2D based on full precision of the input GeoHash string.

#### SQL Examples

```
SELECT ST_Box2dFromGeoHash('9qqj7nmxcggy4d0dbxqz0');
           st_geomfromgeohash
-----
BOX(-115.172816 36.114646,-115.172816 36.114646)

SELECT ST_Box2dFromGeoHash('9qqj7nmxcggy4d0dbxqz0', 0);
           st_box2dfromgeohash
-----
BOX(-180 -90,180 90)

SELECT ST_Box2dFromGeoHash('9qqj7nmxcggy4d0dbxqz0', 10);
           st_box2dfromgeohash
-----
BOX(-115.17282128334 36.1146408319473,-115.172810554504 36.1146461963654)
```

#### ST\_GeoHash, ST\_GeomFromGeoHash, ST\_PointFromGeoHash

[ST\\_GeoHash](#), [ST\\_GeomFromGeoHash](#), [ST\\_PointFromGeoHash](#)

### 8.8.3.2 ST\_GeomFromGeoHash

`ST_GeomFromGeoHash` — GeoHash GEOMETRY based on full precision of the input GeoHash string.

#### Synopsis

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

**GeoHash**

GeoHash — Hexadecimal string representing a point in space. The string is composed of characters from the set `0-9` and `bcdefghjkmnpqrstuvwxyz`. The string is divided into two parts: a prefix and a suffix. The prefix is used to identify the general location of the point, and the suffix is used to identify the specific location of the point.

`precision` — Integer value representing the precision of the GeoHash. The precision is the number of characters in the suffix. The precision is used to determine the size of the bounding box of the point.

`precision` — Integer value representing the precision of the GeoHash. The precision is the number of characters in the suffix. The precision is used to determine the size of the bounding box of the point.

2.1.0 — Hexadecimal string representing a point in space. The string is composed of characters from the set `0-9` and `bcdefghjkmnpqrstuvwxyz`.

**ST\_GeomFromGeoHash**

```
SELECT ST_AsText(ST_GeomFromGeoHash('9qqj7nmxcggy4d0dbxqz0'));
           st_astext
-----
POLYGON((-115.172816 36.114646,-115.172816 36.114646,-115.172816 36.114646,-115.172816 36.114646,-115.172816 36.114646))

SELECT ST_AsText(ST_GeomFromGeoHash('9qqj7nmxcggy4d0dbxqz0', 4));
           st_astext
-----
POLYGON((-115.3125 36.03515625,-115.3125 36.2109375,-114.9609375 36.2109375,-114.9609375 36.03515625,-115.3125 36.03515625))

SELECT ST_AsText(ST_GeomFromGeoHash('9qqj7nmxcggy4d0dbxqz0', 10));
           st_astext
-----
POLYGON((-115.17282128334 36.1146408319473,-115.17282128334 36.1146461963654,-115.172810554504 36.1146461963654,-115.172810554504 36.1146408319473,-115.17282128334 36.1146408319473))
```

**ST\_GeomFromGML**

[ST\\_GeoHash](#), [ST\\_Box2dFromGeoHash](#), [ST\\_PointFromGeoHash](#)

**8.8.3.3 ST\_GeomFromGML**

`ST_GeomFromGML` — Returns a geometry object from a GML string. The GML string is a string representation of a geometry object in the GML format. The GML string is composed of characters from the set `0-9` and `abcdefghijklmnopqrstuvwxyz`.

**Synopsis**

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



**ST\_GeomFromGML**

```
SELECT ST_GeomFromGML('
    <gml:LineString srsName="EPSG:4269">
      <gml:coordinates>
        -71.16028,42.258729 -71.160837,42.259112 ↵
        -71.161143,42.25932
      </gml:coordinates>
    </gml:LineString
>');
```

**ST\_GeomFromGML**

```
SELECT ST_GeomFromGML('
    <gml:LineString xmlns:gml="http://www.opengis.net/gml"
      xmlns:xlink="http://www.w3.org/1999/xlink"
      srsName="urn:ogc:def:crs:EPSG::4269">
      <gml:pointProperty>
        <gml:Point gml:id="p1"
><gml:pos
>42.258729 -71.16028</gml:pos
></gml:Point>
        </gml:pointProperty>
        <gml:pos
>42.259112 -71.160837</gml:pos>
        <gml:pointProperty>
          <gml:Point xlink:type="simple" xlink:href="#p1"/>
        </gml:pointProperty>
      </gml:LineString
>'););
```

**ST\_AsEWKT**

```
SELECT ST_AsEWKT(ST_GeomFromGML('
<gml:PolyhedralSurface>
<gml:polygonPatches>
  <gml:PolygonPatch>
    <gml:exterior>
      <gml:LinearRing
><gml:posList srsDimension="3"
>0 0 0 0 1 0 1 1 0 1 0 0 0 0</gml:posList
></gml:LinearRing>
      </gml:exterior>
    </gml:PolygonPatch>
    <gml:PolygonPatch>
      <gml:exterior>
        <gml:LinearRing
><gml:posList srsDimension="3"
>0 0 0 1 0 1 1 0 1 0 0 0 0 0</gml:posList
></gml:LinearRing>
        </gml:exterior>
      </gml:PolygonPatch>
      <gml:PolygonPatch>
        <gml:exterior>
          <gml:LinearRing
><gml:posList srsDimension="3"
>0 0 0 1 0 0 1 0 1 0 0 1 0 0 0</gml:posList
></gml:LinearRing>
          </gml:exterior>
```

```

</gml:PolygonPatch>
<gml:PolygonPatch>
  <gml:exterior>
    <gml:LinearRing
><gml:posList srsDimension="3"
>1 1 0 1 1 1 1 0 1 1 0 0 1 1 0</gml:posList
</gml:LinearRing>
    </gml:exterior>
  </gml:PolygonPatch>
<gml:PolygonPatch>
  <gml:exterior>
    <gml:LinearRing
><gml:posList srsDimension="3"
>0 1 0 0 1 1 1 1 1 1 1 0 0 1 0</gml:posList
</gml:LinearRing>
    </gml:exterior>
  </gml:PolygonPatch>
<gml:PolygonPatch>
  <gml:exterior>
    <gml:LinearRing
><gml:posList srsDimension="3"
>0 0 1 1 0 1 1 1 1 0 1 1 0 0 1</gml:posList
</gml:LinearRing>
    </gml:exterior>
  </gml:PolygonPatch>
</gml:polygons>
</gml:PolyhedralSurface
>');

-- &#xacc0;&#xacc1; --
POLYHEDRALSURFACE(((0 0 0,0 0 1,0 1 1,0 1 0,0 0 0)),
((0 0 0,0 1 0,1 1 0,1 0 0,0 0 0)),
((0 0 0,1 0 0,1 0 1,0 0 1,0 0 0)),
((1 1 0,1 1 1,1 0 1,1 0 0,1 1 0)),
((0 1 0,0 1 1,1 1 1,1 1 0,0 1 0)),
((0 0 1,1 0 1,1 1 1,0 1 1,0 0 1)))

```

## ST\_GeomFromGeoJSON

Section [2.2.3](#), [ST\\_AsGML](#), [ST\\_GMLToSQL](#)

### 8.8.3.4 ST\_GeomFromGeoJSON

**ST\_GeomFromGeoJSON** — GeoJSON `<string>`; PostGIS `<geometry_type>`.

#### Synopsis

```

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

```

#### ST\_GeomFromGeoJSON

GeoJSON `<string>`; PostGIS `<geometry_type>`.





**ST\_AsText**

```
SELECT ST_AsText(ST_GeomFromTWKB(ST_ASTWKB('LINESTRING(126 34, 127 35)::geometry')));
```

```

      st_astext
-----
LINESTRING(126 34, 127 35)
(1 row)

```

```
SELECT ST_AsEWKT(
  ST_GeomFromTWKB(E'\x620002f7f40dbce4040105')
);
```

```

                                     st_asewkt
-----
LINESTRING(-113.98 39.198,-113.981 39.195)
(1 row)

```

**ST\_AsTWKB****ST\_AsTWKB****8.8.3.7 ST\_GMLToSQL**

**ST\_GMLToSQL** — GML geometry to SQL geometry. `ST_GeomFromGML(gml_text, srid)` returns a geometry object from a GML geometry string. `ST_GeomFromGML(gml_text, srid)` returns a geometry object from a GML geometry string.

**Synopsis**

geometry **ST\_GMLToSQL**(text geomgml);  
 geometry **ST\_GMLToSQL**(text geomgml, integer srid);

**Notes**

This method implements the SQL/MM specification. SQL-MM 3: 5.1.50 (2D/3D simple surface, TIN, TIN surface).

1.5 (2D/3D simple surface, TIN, TIN surface).  
 LibXML2 1.6 (2D/3D simple surface, TIN, TIN surface).

2.0 (2D/3D simple surface, TIN, TIN surface).  
 surface) TIN surface).

2.0 (2D/3D simple surface, TIN, TIN surface).  
 SRID (2D/3D simple surface, TIN, TIN surface).

**ST\_GeomFromGML**

Section 2.2.3, [ST\\_GeomFromGML](#), [ST\\_AsGML](#)

**8.8.3.8 ST\_LineFromEncodedPolyline**

**ST\_LineFromEncodedPolyline** — Encoded Polyline to SQL geometry. `ST_LineFromEncodedPolyline(encoded_polyline, srid)` returns a geometry object from an encoded polyline string.



**ST\_AsText**

```
SELECT ST_AsText(ST_PointFromGeoHash('9qqj7nmxcggy4d0dbxqz0'));
      st_astext
-----
POINT(-115.172816 36.114646)

SELECT ST_AsText(ST_PointFromGeoHash('9qqj7nmxcggy4d0dbxqz0', 4));
      st_astext
-----
POINT(-115.13671875 36.123046875)

SELECT ST_AsText(ST_PointFromGeoHash('9qqj7nmxcggy4d0dbxqz0', 10));
      st_astext
-----
POINT(-115.172815918922 36.1146435141563)
```

**ST\_GeoHash, ST\_Box2dFromGeoHash, ST\_GeomFromGeoHash**

**ST\_GeoHash, ST\_Box2dFromGeoHash, ST\_GeomFromGeoHash**

**8.8.3.10 ST\_FromFlatGeobufToTable**

**ST\_FromFlatGeobufToTable** — Creates a table based on the structure of FlatGeobuf data.

**Synopsis**

geometry **ST\_BdPolyFromText**(text WKT, integer srid);

**Creates a table based on the structure of FlatGeobuf data.**

Creates a table based on the structure of FlatGeobuf data. (<http://flatgeobuf.org>).

`schema` Schema name.

`table` Table name.

`data` Input FlatGeobuf data.

Availability: 3.2.0

**8.8.3.11 ST\_FromFlatGeobuf**

**ST\_FromFlatGeobuf** — Reads FlatGeobuf data.

**Synopsis**

setof anyelement **ST\_FromFlatGeobuf**(anyelement Table reference, bytea FlatGeobuf input data);

**Creates a table based on the structure of FlatGeobuf data.**

Reads FlatGeobuf data (<http://flatgeobuf.org>). NOTE: PostgreSQL bytea cannot exceed 1GB.

`tabletype` reference to a table type.

`data` input FlatGeobuf data.

Availability: 3.2.0

## 8.9 Geometry Output

### 8.9.1 Well-Known Text (WKT)

#### 8.9.1.1 ST\_AsEWKT

`ST_AsEWKT` — Well-Known Text (WKT) representation of the geometry prefixed with the SRID. The optional `maxdecimaldigits` argument may be used to reduce the maximum number of decimal digits after floating point used in output (defaults to 15).

#### Synopsis

```
text ST_AsEWKT(geometry g1);
text ST_AsEWKT(geometry g1, integer maxdecimaldigits=15);
text ST_AsEWKT(geography g1);
text ST_AsEWKT(geography g1, integer maxdecimaldigits=15);
```

#### Warning

Returns the Well-Known Text representation of the geometry prefixed with the SRID. The optional `maxdecimaldigits` argument may be used to reduce the maximum number of decimal digits after floating point used in output (defaults to 15).

To perform the inverse conversion of EWKT representation to PostGIS geometry use `ST_GeomFromEWKT`.



#### Warning

Using the `maxdecimaldigits` parameter can cause output geometry to become invalid. To avoid this use `ST_ReducePrecision` with a suitable gridsize first.



#### Note

The WKT spec does not include the SRID. To get the OGC WKT format use `ST_AsText`.



#### Warning

WKT format does not maintain precision so to prevent floating truncation, use `ST_AsBinary` or `ST_AsEWKB` format for transport.

Enhanced: 3.1.0 support for optional precision parameter.

2.0.0 support for optional precision parameter. The optional `maxdecimaldigits` argument may be used to reduce the maximum number of decimal digits after floating point used in output (defaults to 15).



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



This method supports Circular Strings and Curves



This function supports Polyhedral surfaces.



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





**ST\_AsBinary**

Returns the OGC/ISO **Well-Known Binary** (WKB) representation of the geometry. The first function variant defaults to encoding using server machine endian. The second function variant takes a text argument specifying the endian encoding, either little-endian ('NDR') or big-endian ('XDR').

WKB format is useful to read geometry data from the database and maintaining full numeric precision. This avoids the precision rounding that can happen with text formats such as WKT.

To perform the inverse conversion of WKB to PostGIS geometry use [ST\\_GeomFromWKB](#).

**Note**

The OGC/ISO WKB format does not include the SRID. To get the EWKB format which does include the SRID use [ST\\_AsEWKB](#)

**Note**

The default behavior in PostgreSQL 9.0 has been changed to output bytea in hex encoding. If your GUI tools require the old behavior, then SET bytea\_output='escape' in your database.

```
ST_AsBinary(geometry, 'NDR')::bytea;
ST_AsBinary(geometry, 'XDR')::bytea;
```

```
ST_AsBinary(geometry, 'NDR')::text;
ST_AsBinary(geometry, 'XDR')::text;
```

```
ST_AsBinary(geometry, 'NDR', SRID)::bytea;
ST_AsBinary(geometry, 'XDR', SRID)::bytea;
```

```
ST_AsBinary(geometry, 'NDR', SRID)::text;
ST_AsBinary(geometry, 'XDR', SRID)::text;
```

```
ST_AsBinary(geometry, 'NDR', SRID, 'WKT')::bytea;
ST_AsBinary(geometry, 'XDR', SRID, 'WKT')::bytea;
ST_AsBinary(geometry, 'NDR', SRID, 'WKT')::text;
ST_AsBinary(geometry, 'XDR', SRID, 'WKT')::text;
ST_AsBinary(geometry, 'NDR', SRID, 'WKT', 'ESCAPE')::bytea;
ST_AsBinary(geometry, 'XDR', SRID, 'WKT', 'ESCAPE')::bytea;
ST_AsBinary(geometry, 'NDR', SRID, 'WKT', 'ESCAPE')::text;
ST_AsBinary(geometry, 'XDR', SRID, 'WKT', 'ESCAPE')::text;
ST_AsBinary(geometry, 'NDR', SRID, 'WKT', 'ESCAPE', 'ESCAPE')::bytea;
ST_AsBinary(geometry, 'XDR', SRID, 'WKT', 'ESCAPE', 'ESCAPE')::bytea;
ST_AsBinary(geometry, 'NDR', SRID, 'WKT', 'ESCAPE', 'ESCAPE')::text;
ST_AsBinary(geometry, 'XDR', SRID, 'WKT', 'ESCAPE', 'ESCAPE')::text;
```



This method implements the [OGC Simple Features Implementation Specification for SQL 1.1](#). s2.1.1.1



This method implements the SQL/MM specification. SQL-MM 3: 5.1.37



This method supports Circular Strings and Curves



This function supports Polyhedral surfaces.



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



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





**Example 8.9.3.1**

```
SELECT ST_AsHEXEWKB(ST_GeomFromText('POLYGON((0 0,0 1,1 1,1 0,0 0))',4326));
-- 0103000020E6100000010000000500  ←
   00000000000000000000000000000000
   00000000000000000000000000000000F03F
   000000000000F03F000000000000F03F000000000000F03
   F000000000000000000000000000000000000000000000000
```

**8.9.3 Other Formats****8.9.3.1 ST\_AsEncodedPolyline**

**ST\_AsEncodedPolyline** — Returns the geometry as an Encoded Polyline. This format is used by Google Maps with precision=5 and by Open Source Routing Machine with precision=5 and 6.

**Synopsis**

```
text ST_AsEncodedPolyline(geometry geom, integer precision=5);
```

**Options**

Returns the geometry as an Encoded Polyline. This format is used by Google Maps with precision=5 and by Open Source Routing Machine with precision=5 and 6.

Optional `precision` specifies how many decimal places will be preserved in Encoded Polyline. Value should be the same on encoding and decoding, or coordinates will be incorrect.

2.2.0 Returns the geometry as an Encoded Polyline. This format is used by Google Maps with precision=5 and by Open Source Routing Machine with precision=5 and 6.

**Example 8.9.3.1****Options**

```
SELECT ST_AsEncodedPolyline(GeomFromEWKT('SRID=4326;LINESTRING(-120.2 38.5,-120.95 40.7,-126.453 43.252)'));
-- 0103000020E6100000010000000500  ←
   00000000000000000000000000000000
   00000000000000000000000000000000F03F
   000000000000F03F000000000000F03F000000000000F03
   F000000000000000000000000000000000000000000000000
```

Returns the geometry as an Encoded Polyline. This format is used by Google Maps with precision=5 and by Open Source Routing Machine with precision=5 and 6.

```
-- 0103000020E6100000010000000500  ←
   00000000000000000000000000000000
   00000000000000000000000000000000F03F
   000000000000F03F000000000000F03F000000000000F03
   F000000000000000000000000000000000000000000000000
   SELECT ST_AsEncodedPolyline(
     ST_Segmentize(
       ST_GeogFromText('LINESTRING(-71.0519 42.4935,-122.4483 37.64)'),
       100000)::geometry) As encodedFlightPath;
```

```

<script type="text/javascript" src="http://maps.googleapis.com/maps/api/js?libraries=
  geometry"
></script>
<script type="text/javascript">
  flightPath = new google.maps.Polyline({
    path: google.maps.geometry.encoding.decodePath("$encodedFlightPath"),
    map: map,
    strokeColor: '#0000CC',
    strokeOpacity: 1.0,
    strokeWeight: 4
  });
</script>

```

**ST\_LineFromEncodedPolyline, ST\_Segmentize**

**ST\_LineFromEncodedPolyline, ST\_Segmentize**

### 8.9.3.2 ST\_AsFlatGeobuf

ST\_AsFlatGeobuf — Return a FlatGeobuf representation of a set of rows.

#### Synopsis

```

bytea ST_AsFlatGeobuf(anyelement set row);
bytea ST_AsFlatGeobuf(anyelement row, bool index);
bytea ST_AsFlatGeobuf(anyelement row, bool index, text geom_name);

```

**Return Value**

Return a FlatGeobuf representation (<http://flatgeobuf.org>) of a set of rows corresponding to a FeatureCollection. NOTE: PostgreSQL bytea cannot exceed 1GB.

**row** row data with at least a geometry column.

**index** toggle spatial index creation. Default is false.

**geom\_name** is the name of the geometry column in the row data. If NULL it will default to the first found geometry column.

Availability: 3.2.0

### 8.9.3.3 ST\_AsGeobuf

ST\_AsGeobuf — Return a Geobuf representation of a set of rows.

#### Synopsis

```

bytea ST_AsGeobuf(anyelement set row);
bytea ST_AsGeobuf(anyelement row, text geom_name);

```

**Warning**

Return a Geobuf representation (<https://github.com/mapbox/geobuf>) of a set of rows corresponding to a FeatureCollection. Every input geometry is analyzed to determine maximum precision for optimal storage. Note that Geobuf in its current form cannot be streamed so the full output will be assembled in memory.

row row data with at least a geometry column.

geom\_name is the name of the geometry column in the row data. If NULL it will default to the first found geometry column.

Availability: 2.4.0

**Example**

```
SELECT encode(ST_AsGeobuf(q, 'geom'), 'base64')
  FROM (SELECT ST_GeomFromText('POLYGON((0 0,0 1,1 1,1 0,0 0))') AS geom) AS q;
 st_asgeobuf
-----
GAAiEAoOCgwIBBoIAAAAAGIAAAE=
```

**8.9.3.4 ST\_AsGeoJSON**

ST\_AsGeoJSON — Return a geometry as a GeoJSON element.

**Synopsis**

text **ST\_AsGeoJSON**(record feature, text geomcolumnname, integer maxdecimaldigits=9, boolean pretty\_bool=false);

text **ST\_AsGeoJSON**(geometry geom, integer maxdecimaldigits=9, integer options=8);

text **ST\_AsGeoJSON**(geography geog, integer maxdecimaldigits=9, integer options=0);

**Warning**

Returns a geometry as a GeoJSON "geometry", or a row as a GeoJSON "feature". (See the [GeoJSON specifications RFC 7946](#)). 2D and 3D Geometries are both supported. GeoJSON only support SFS 1.1 geometry types (no curve support for example).

The `maxdecimaldigits` argument may be used to reduce the maximum number of decimal places used in output (defaults to 9). If you are using EPSG:4326 and are outputting the geometry only for display, `maxdecimaldigits=6` can be a good choice for many maps.

**Warning**

Using the `maxdecimaldigits` parameter can cause output geometry to become invalid. To avoid this use **ST\_ReducePrecision** with a suitable gridsize first.

The `options` argument can be used to add BBOX or CRS in GeoJSON output:

- 0: means no option
- 1: GeoJSON BBOX
- 2: GeoJSON Short CRS (&#xc608;; EPSG:4326)
- 4: GeoJSON Long CRS (&#xc608;; urn:ogc:def:crs:EPSG::4326)
- 8: GeoJSON Short CRS if not EPSG:4326 (default)

The GeoJSON specification states that polygons are oriented using the Right-Hand Rule, and some clients require this orientation. This can be ensured by using `ST_ForcePolygonCCW`. The specification also requires that geometry be in the WGS84 coordinate system (SRID = 4326). If necessary geometry can be projected into WGS84 using `ST_Transform`: `ST_Transform(geom, 4326)`.

GeoJSON can be tested and viewed online at [geojson.io](http://geojson.io) and [geojsonlint.com](http://geojsonlint.com). It is widely supported by web mapping frameworks:

- [OpenLayers GeoJSON Example](#)
- [Leaflet GeoJSON Example](#)
- [Mapbox GL GeoJSON Example](#)

1.3.4

1.5.0  
 2.0.0  
 (default arg)  
 (named arg)

Changed: 3.0.0 support records as input

Changed: 3.0.0 output SRID if not EPSG:4326.



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

## Generate a FeatureCollection:

Generate a FeatureCollection:

```
SELECT json_build_object(
  'type', 'FeatureCollection',
  'features', json_agg(ST_AsGeoJSON(t.*)::json)
)
FROM ( VALUES (1, 'one', 'POINT(1 1)::geometry),
              (2, 'two', 'POINT(2 2)'),
              (3, 'three', 'POINT(3 3)')
       ) as t(id, name, geom);
```

```
{"type": "FeatureCollection", "features": [{"type": "Feature", "geometry": {"type": "Point", "coordinates": [1,1]}, "properties": {"id": 1, "name": "one"}}, {"type": "Feature", "geometry": {"type": "Point", "coordinates": [2,2]}, "properties": {"id": 2, "name": "two"}}, {"type": "Feature", "geometry": {"type": "Point", "coordinates": [3,3]}, "properties": {"id": 3, "name": "three"}}]}
```

Generate a Feature:

```
SELECT ST_AsGeoJSON(t.*)
FROM (VALUES (1, 'one', 'POINT(1 1)::geometry)) AS t(id, name, geom);
```

st\_asgeojson

```
{"type": "Feature", "geometry": {"type": "Point", "coordinates": [1,1]}, "properties": {"id": 1, "name": "one"}}
```

An alternate way to generate Features with an id property is to use JSONB functions and operators:

```
SELECT jsonb_build_object(
  'type',      'Feature',
  'id',        id,
  'geometry',  ST_AsGeoJSON(geom)::jsonb,
  'properties', to_jsonb( t.* ) - 'id' - 'geom'
) AS json
FROM (VALUES (1, 'one', 'POINT(1 1)::geometry)) AS t(id, name, geom);
```

```
json
```

```
-----
{"id": 1, "type": "Feature", "geometry": {"type": "Point", "coordinates": [1, 1]}, " ←
  properties": {"name": "one"}}
```

Don't forget to transform your data to WGS84 longitude, latitude to conform with the GeoJSON specification:

```
SELECT ST_AsGeoJSON(ST_Transform(geom,4326)) from fe_edges limit 1;
```

```
st_asgeojson
```

```
-----
{"type": "MultiLineString", "coordinates": [[[-89.734634999999997, 31.492072000000000],
[-89.734955999999997, 31.492237999999997]]]}
```

3D geometries are supported:

```
SELECT ST_AsGeoJSON('LINESTRING(1 2 3, 4 5 6)');
```

```
-----
{"type": "LineString", "coordinates": [[1,2,3],[4,5,6]]}
```

**ST\_GeomFromGeoJSON**

**ST\_GeomFromGeoJSON**, **ST\_ForcePolygonCCW**, **ST\_Transform**

### 8.9.3.5 ST\_AsGML

**ST\_AsGML** — **ST\_AsGML**(geometry geom, integer maxdecimaldigits=15, integer options=0);  
**ST\_AsGML**(geography geog, integer maxdecimaldigits=15, integer options=0, text nprefix=null, text id=null);  
**ST\_AsGML**(integer version, geometry geom, integer maxdecimaldigits=15, integer options=0, text nprefix=null, text id=null);  
**ST\_AsGML**(integer version, geography geog, integer maxdecimaldigits=15, integer options=0, text nprefix=null, text id=null);

#### Synopsis

text **ST\_AsGML**(geometry geom, integer maxdecimaldigits=15, integer options=0);  
text **ST\_AsGML**(geography geog, integer maxdecimaldigits=15, integer options=0, text nprefix=null, text id=null);  
text **ST\_AsGML**(integer version, geometry geom, integer maxdecimaldigits=15, integer options=0, text nprefix=null, text id=null);  
text **ST\_AsGML**(integer version, geography geog, integer maxdecimaldigits=15, integer options=0, text nprefix=null, text id=null);

**ST\_AsGML**

Return the geometry as a Geography Markup Language (GML) element. The version parameter, if specified, may be either 2 or 3. If no version parameter is specified then the default is assumed to be 2. The `maxdecimaldigits` argument may be used to reduce the maximum number of decimal places used in output (defaults to 15).



**Note**

ST\_AsGML (3, ST\_GeomFromText('POLYGON((0 0,0 1,1 1,1 0,0 0))',4326));



This method implements the SQL/MM specification. SQL-MM IEC 13249-3: 17.2



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



This function supports Polyhedral surfaces.



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

**ST\_AsGML (3, ST\_GeomFromText('POLYGON((0 0,0 1,1 1,1 0,0 0))',4326));**

```
SELECT ST_AsGML(ST_GeomFromText('POLYGON((0 0,0 1,1 1,1 0,0 0))',4326));
      st_asgml
      -----
      <gml:Polygon srsName="EPSG:4326"
><gml:outerBoundaryIs
><gml:LinearRing
><gml:coordinates
>0,0 0,1 1,1 1,0 0,0</gml:coordinates
></gml:LinearRing
></gml:outerBoundaryIs
></gml:Polygon
>
```

**ST\_AsGML (3, ST\_GeomFromText('POINT(5.234234233242 6.34534534534)',4326), 5, 17);**

```
-- ST_AsGML (3, ST_GeomFromText('POINT(5.234234233242 6.34534534534)',4326), 5, 17);
      st_asgml
      -----
      <gml:Point srsName="urn:ogc:def:crs:EPSG::4326"
><gml:pos
>6.34535 5.23423</gml:pos
></gml:Point
>
```

```
-- ST_AsGML (3, ST_GeomFromText('LINESTRING(1 2, 3 4, 10 20)',4326), 5, 32);
      st_asgml
      -----
      <gml:Envelope srsName="EPSG:4326">
      <gml:lowerCorner
>1 2</gml:lowerCorner>
      <gml:upperCorner
>10 20</gml:upperCorner>
      </gml:Envelope
>
```

```

-- &#xc0c1;&#xc790;(envelope)&#xb97c; &#xcd9c;&#xb825;&#xd558;&#xace0; (32), ←
&#xc88c;&#xd45c;&#xb97c; &#xb4a4;&#xc9d1;&#xace0; (&#xacbd;&#xb3c4; &#xc704;&#xb3c4; ←
&#xb300;&#xc2e0; &#xc704;&#xb3c4; &#xacbd;&#xb3c4;) (16), long SRS&#xb97c; ←
&#xc0ac;&#xc6a9; (1) = 32 | 16 | 1 = 49 --
SELECT ST_AsGML(3, ST_GeomFromText('LINESTRING(1 2, 3 4, 10 20)',4326), 5, 49);
    st_asgml
    -----
<gml:Envelope srsName="urn:ogc:def:crs:EPSG::4326">
  <gml:lowerCorner
>2 1</gml:lowerCorner>
  <gml:upperCorner
>20 10</gml:upperCorner>
</gml:Envelope
>

-- &#xb2e4;&#xba74;&#xccb4; &#xc608;&#xc2dc; --
SELECT ST_AsGML(3, ST_GeomFromEWKT('POLYHEDRALSURFACE( ((0 0 0, 0 0 1, 0 1 1, 0 1 0, 0 0 0) ←
),
((0 0 0, 0 1 0, 1 1 0, 1 0 0, 0 0 0)), ((0 0 0, 1 0 0, 1 0 1, 0 0 1, 0 0 0)),
((1 1 0, 1 1 1, 1 0 1, 1 0 0, 1 1 0)),
((0 1 0, 0 1 1, 1 1 1, 1 1 0, 0 1 0)), ((0 0 1, 1 0 1, 1 1 1, 0 1 1, 0 0 1)) )'));
    st_asgml
    -----
<gml:PolyhedralSurface>
<gml:polygonPatches>
  <gml:PolygonPatch>
    <gml:exterior>
      <gml:LinearRing>
        <gml:posList srsDimension="3"
>0 0 0 0 0 1 0 1 1 0 1 0 0 0 0</gml:posList>
        </gml:LinearRing>
      </gml:exterior>
    </gml:PolygonPatch>
  <gml:PolygonPatch>
    <gml:exterior>
      <gml:LinearRing>
        <gml:posList srsDimension="3"
>0 0 0 0 1 0 1 1 0 1 0 0 0 0 0</gml:posList>
        </gml:LinearRing>
      </gml:exterior>
    </gml:PolygonPatch>
  <gml:PolygonPatch>
    <gml:exterior>
      <gml:LinearRing>
        <gml:posList srsDimension="3"
>0 0 0 1 0 0 1 0 1 0 0 1 0 0 0</gml:posList>
        </gml:LinearRing>
      </gml:exterior>
    </gml:PolygonPatch>
  <gml:PolygonPatch>
    <gml:exterior>
      <gml:LinearRing>
        <gml:posList srsDimension="3"
>1 1 0 1 1 1 1 0 1 1 0 0 1 1 0</gml:posList>
        </gml:LinearRing>
      </gml:exterior>
    </gml:PolygonPatch>
  <gml:PolygonPatch>
    <gml:exterior>
      <gml:LinearRing>
        <gml:posList srsDimension="3"

```

```
>0 1 0 0 1 1 1 1 1 1 0 0 1 0</gml:posList>
      </gml:LinearRing>
    </gml:exterior>
  </gml:PolygonPatch>
  <gml:PolygonPatch>
    <gml:exterior>
      <gml:LinearRing>
        <gml:posList srsDimension="3"
>0 0 1 1 0 1 1 1 1 0 1 1 0 0 1</gml:posList>
      </gml:LinearRing>
    </gml:exterior>
  </gml:PolygonPatch>
</gml:polygonPatches>
</gml:PolyhedralSurface
>
```

**Warning**

**ST\_GeomFromGML**

### 8.9.3.6 ST\_AsKML

**ST\_AsKML** — KML 2 (GML 2), KML 3 (GML 3), and KML 4 (GML 4).

#### Synopsis

text **ST\_AsKML**(geometry geom, integer maxdecimaldigits=15, text nprefix=NULL);  
text **ST\_AsKML**(geography geog, integer maxdecimaldigits=15, text nprefix=NULL);

**Warning**

Keyhole Markup Language (KML) is a file format for storing geographic data. It is used to describe geographic features such as points, lines, polygons, and surfaces. The format is based on XML and is used to describe geographic data in a way that can be understood by a variety of software applications. The format is used to describe geographic data in a way that can be understood by a variety of software applications.



#### Warning

Using the *maxdecimaldigits* parameter can cause output geometry to become invalid. To avoid this use **ST\_ReducePrecision** with a suitable gridsize first.



#### Note

PostGIS 3.3.6dev Proj 4: GML 2 (GML 2), GML 3 (GML 3), and GML 4 (GML 4).  
PostGIS 3.3.6dev Proj 4: GML 2 (GML 2), GML 3 (GML 3), and GML 4 (GML 4).  
PostGIS 3.3.6dev Proj 4: GML 2 (GML 2), GML 3 (GML 3), and GML 4 (GML 4).

**Note**

1.2.2 `ST_AsKML` `ST_GeomFromText('POLYGON((0 0,0 1,1 1,1 0,0 0))',4326);` `SRID=4326;LINESTRING(1 2 3, 4 5 6);` 1.3.2

**Note**

`ST_AsKML` `ST_GeomFromText('POLYGON((0 0,0 1,1 1,1 0,0 0))',4326);` 2.0.0 `SRID=4326;LINESTRING(1 2 3, 4 5 6);`

**Note**

Changed: 3.0.0 - Removed the "versioned" variant signature

**Note**

`ST_AsKML` `ST_GeomFromText('POLYGON((0 0,0 1,1 1,1 0,0 0))',4326);` `SRID=4326;LINESTRING(1 2 3, 4 5 6);`



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

**3D Support**

```
SELECT ST_AsKML(ST_GeomFromText('POLYGON((0 0,0 1,1 1,1 0,0 0))',4326));

      st_askml
      -
      <Polygon
><outerBoundaryIs
><LinearRing
><coordinates
>0,0 0,1 1,1 1,0 0,0</coordinates
></LinearRing
></outerBoundaryIs
></Polygon>

-- 3D LINESTRING
      SELECT ST_AsKML('SRID=4326;LINESTRING(1 2 3, 4 5 6)');
      <LineString
><coordinates
>1,2,3 4,5,6</coordinates
></LineString>
```

**ST\_AsSVG, ST\_AsGML**

[ST\\_AsSVG](#), [ST\\_AsGML](#)

### 8.9.3.7 ST\_AsLatLonText

ST\_AsLatLonText —

#### Synopsis

text ST\_AsLatLonText(geometry pt, text format=);

#### Note



#### Note

ST\_AsLatLonText(*geometry* pt, *text format*=);

X(*lon*) Y(*lat*)

180 *lon* 90 *lat*

90 *lon* 180 *lat*

ST\_AsLatLonText(*geometry* pt, *text format*=);

ST\_AsLatLonText(*geometry* pt, 'D'); -- cardinal direction

ST\_AsLatLonText(*geometry* pt, 'M'); -- magnetic declination

ST\_AsLatLonText(*geometry* pt, 'S'); -- spherical coordinates

ST\_AsLatLonText(*geometry* pt, 'C'); -- Cartesian coordinates

ST\_AsLatLonText(*geometry* pt, 'U'); -- UTM coordinates

ST\_AsLatLonText(*geometry* pt, 'W'); -- WGS84 coordinates

ST\_AsLatLonText(*geometry* pt, 'U'); -- UTM coordinates

ST\_AsLatLonText(*geometry* pt, 'W'); -- WGS84 coordinates

ST\_AsLatLonText(*geometry* pt, 'U'); -- UTM coordinates

ST\_AsLatLonText(*geometry* pt, 'W'); -- WGS84 coordinates

#### Note

ST\_AsLatLonText(*geometry* pt, 'U');

```
SELECT (ST_AsLatLonText('POINT (-3.2342342 -2.32498)'));
st_aslatlon
-----
2\textdegree{}19'29.928"S 3\textdegree{}14'3.243"W
```

ST\_AsLatLonText(*geometry* pt, 'U');

```
SELECT (ST_AsLatLonText('POINT (-3.2342342 -2.32498)', 'D\textdegree{}M'S.SSS"C'));
      st_aslatlon
-----
2\textdegree{}19'29.928"S 3\textdegree{}14'3.243"W
```

D, M, S, C &#xc0f;.&#xc774; &#xc544;&#xb2cc; &#xb2e4;&#xb978; &#xb38;&#xc790;&#xb4e4;&#xc740; &#xadf8;&#xb0e5; &#xb34;&#xc2dc;&#xb429;&#xb2c8;&#xb2e4;.

```
SELECT (ST_AsLatLonText('POINT (-3.2342342 -2.32498)', 'D degrees, M minutes, S seconds to the C'));
      st_aslatlon
-----
2 degrees, 19 minutes, 30 seconds to the S 3 degrees, 14 minutes, 3 seconds to the W
```

&#xae30;&#xbcf8;&#xb29;&#xd5a5; &#xb300;&#xc2e0; &#xbd80;&#xd638;&#xac00; &#xbd99;&#xc740; &#xb3c4;&#xb97c; &#xc0ac;&#xc6a9;&#xd569;&#xb2c8;&#xb2e4;.

```
SELECT (ST_AsLatLonText('POINT (-3.2342342 -2.32498)', 'D\textdegree{}M'S.SSS"));
      st_aslatlon
-----
-2\textdegree{}19'29.928" -3\textdegree{}14'3.243"
```

&#xc18c;&#xc218;&#xc810;&#xc774; &#xbd99;&#xc740; &#xb3c4;&#xb97c; &#xc0ac;&#xc6a9;&#xd569;&#xb2c8;&#xb2e4;.

```
SELECT (ST_AsLatLonText('POINT (-3.2342342 -2.32498)', 'D.DDDD degrees C'));
      st_aslatlon
-----
2.3250 degrees S 3.2342 degrees W
```

&#xc9c0;&#xb098;&#xce58;&#xac8c; &#xd070; &#xac12;&#xc740; &#xc815;&#xaddd;&#xd654;&#xb429;&#xb2c8;&#xb2e4;.

```
SELECT (ST_AsLatLonText('POINT (-302.2342342 -792.32498)'));
      st_aslatlon
-----
72\textdegree{}19'29.928"S 57\textdegree{}45'56.757"E
```

### 8.9.3.8 ST\_AsMARC21

ST\_AsMARC21 — Returns geometry as a MARC21/XML record with a geographic datafield (034).

#### Synopsis

```
text ST_AsMARC21 ( geometry geom , text format='hddmms' );
```

#### &#xc124;&#xba85;

This function returns a MARC21/XML record with **Coded Cartographic Mathematical Data** representing the bounding box of a given geometry. The *format* parameter allows to encode the coordinates in subfields \$d,\$e,\$f and \$g in all formats supported by the MARC21/XML standard. Valid formats are:

- cardinal direction, degrees, minutes and seconds (default): hddmms
- decimal degrees with cardinal direction: hddd.d
- decimal degrees without cardinal direction: ddd.d
- decimal minutes with cardinal direction: hddmm.m

- decimal minutes without cardinal direction: dddmm.mmmmm
- decimal seconds with cardinal direction: hddmmss.sss

The decimal sign may be also a comma, e.g. hddmm,mmmm.

The precision of decimal formats can be limited by the number of characters after the decimal sign, e.g. hddmm.mm for decimal minutes with a precision of two decimals.

This function ignores the Z and M dimensions.

LOC MARC21/XML versions supported:

- [MARC21/XML 1.1](#)

Availability: 3.3.0



#### Note

This function does not support non lon/lat geometries, as they are not supported by the MARC21/XML standard (Coded Cartographic Mathematical Data).



#### Note

The MARC21/XML Standard does not provide any means to annotate the spatial reference system for Coded Cartographic Mathematical Data, which means that this information will be lost after conversion to MARC21/XML.

## Converting a POINT to MARC21/XML

Converting a POINT to MARC21/XML formatted as hddmmss (default)

```
SELECT ST_AsMARC21 ('SRID=4326;POINT(-4.504289 54.253312)')::geometry);

          st_asmarc21
-----
<record xmlns="http://www.loc.gov/MARC21/slim">
  <datafield tag="034" ind1="1" ind2=" " >
    <subfield code="a">a</subfield>
    <subfield code="d">W0043015</subfield>
    <subfield code="e">W0043015</subfield>
    <subfield code="f">N0541512</subfield>
    <subfield code="g">N0541512</subfield>
  </datafield>
</record>
```

Converting a POLYGON to MARC21/XML formatted in decimal degrees

```
SELECT ST_AsMARC21 ('SRID=4326;POLYGON((-4.5792388916015625 54.18172660239091,
54.18172660239091,-4.56756591796875 54.196993557130355,-4.546623229980469 54.18313300502024,
-4.5792388916015625 54.18172660239091))')::geometry, 'hddd.dddd');

          st_asmarc21
-----
<record xmlns="http://www.loc.gov/MARC21/slim">
  <datafield tag="034" ind1="1" ind2=" " >
```

```

    <subfield code="a">a</subfield>
    <subfield code="d">W004.5792</subfield>
    <subfield code="e">W004.5466</subfield>
    <subfield code="f">N054.1970</subfield>
    <subfield code="g">N054.1817</subfield>
  </datafield>
</record>

```

Converting a `GEOMETRYCOLLECTION` to MARC21/XML formatted in decimal minutes. The geometries order in the MARC21/XML output correspond to their order in the collection.

```

SELECT ST_AsMARC21 ('SRID=4326;GEOMETRYCOLLECTION (POLYGON ((13.1 52.65,
13.516666666666667 52.65,13.516666666666667 52.38333333333333,13.1 52.38333333333333,
13.1 52.65)), POINT (-4.5 54.25))'::geometry, 'hdddmm.mmm');

          st_asmarc21
-----
<record xmlns="http://www.loc.gov/MARC21/slim">
  <datafield tag="034" ind1="1" ind2=" " >
    <subfield code="a">a</subfield>
    <subfield code="d">E01307.0000</subfield>
    <subfield code="e">E01331.0000</subfield>
    <subfield code="f">N05240.0000</subfield>
    <subfield code="g">N05224.0000</subfield>
  </datafield>
  <datafield tag="034" ind1="1" ind2=" " >
    <subfield code="a">a</subfield>
    <subfield code="d">W00430.0000</subfield>
    <subfield code="e">W00430.0000</subfield>
    <subfield code="f">N05415.0000</subfield>
    <subfield code="g">N05415.0000</subfield>
  </datafield>
</record>

```

**ST\_GeomFromMARC21**

**ST\_GeomFromMARC21**

### 8.9.3.9 ST\_AsMVTGeom

`ST_AsMVTGeom` — Transforms a geometry into the coordinate space of a MVT tile.

#### Synopsis

geometry `ST_AsMVTGeom`(geometry geom, box2d bounds, integer extent=4096, integer buffer=256, boolean clip\_geom=true);

**ST\_AsMVTGeom**

Transforms a geometry into the coordinate space of a MVT ([Mapbox Vector Tile](#)) tile, clipping it to the tile bounds if required. The geometry must be in the coordinate system of the target map (using [ST\\_Transform](#) if needed). Commonly this is [Web Mercator](#) (SRID:3857).

The function attempts to preserve geometry validity, and corrects it if needed. This may cause the result geometry to collapse to a lower dimension.

The rectangular bounds of the tile in the target map coordinate space must be provided, so the geometry can be transformed, and clipped if required. The bounds can be generated using [ST\\_MakeEnvelope](#).

This function is used to convert geometry into the tile coordinate space required by [ST\\_AsMVT](#).

`geom` is the geometry to transform, in the coordinate system of the target map.

`bounds` is the rectangular bounds of the tile in map coordinate space, with no buffer.

`extent` is the tile extent size in tile coordinate space as defined by the [MVT specification](#). Defaults to 4096.

`buffer` is the buffer size in tile coordinate space for geometry clipping. Defaults to 256.

`clip_geom` is a boolean to control if geometries are clipped or encoded as-is. Defaults to true.

Availability: 2.4.0



#### Note

From 3.0, Wagyu can be chosen at configure time to clip and validate MVT polygons. This library is faster and produces more correct results than the GEOS default, but it might drop small polygons.

#### Canonical example

```
SELECT ST_AsText(ST_AsMVTGeom(
  ST_GeomFromText('POLYGON ((0 0, 10 0, 10 5, 0 -5, 0 0))'),
  ST_MakeBox2D(ST_Point(0, 0), ST_Point(4096, 4096)),
  4096, 0, false));
           st_astext
-----
MULTIPOLYGON(((5 4096,10 4091,10 4096,5 4096)),((5 4096,0 4101,0 4096,5 4096)))
```

Canonical example for a Web Mercator tile using a computed tile bounds to query and clip geometry.

```
SELECT ST_AsMVTGeom(
  ST_Transform( geom, 3857 ),
  ST_TileEnvelope(12, 513, 412), extent => 4096, buffer => 64) AS geom
FROM data
WHERE geom && ST_TileEnvelope(12, 513, 412, margin => (64.0 / 4096))
```

#### See also

[ST\\_AsMVT](#), [ST\\_MakeEnvelope](#), [PostGIS\\_Wagyu\\_Version](#)

### 8.9.3.10 ST\_AsMVT

`ST_AsMVT` — Aggregate function returning a MVT representation of a set of rows.

#### Synopsis

bytea `ST_AsMVT`(anyelement set row);

bytea `ST_AsMVT`(anyelement row, text name);

bytea `ST_AsMVT`(anyelement row, text name, integer extent);

bytea `ST_AsMVT`(anyelement row, text name, integer extent, text geom\_name);

bytea `ST_AsMVT`(anyelement row, text name, integer extent, text geom\_name, text feature\_id\_name);

## ST\_AsMVTGeom

An aggregate function which returns a binary [Mapbox Vector Tile](#) representation of a set of rows corresponding to a tile layer. The rows must contain a geometry column which will be encoded as a feature geometry. The geometry must be in tile coordinate space and valid as per the [MVT specification](#). [ST\\_AsMVTGeom](#) can be used to transform geometry into tile coordinate space. Other row columns are encoded as feature attributes.

The [Mapbox Vector Tile](#) format can store features with varying sets of attributes. To use this capability supply a JSONB column in the row data containing Json objects one level deep. The keys and values in the JSONB values will be encoded as feature attributes.

Tiles with multiple layers can be created by concatenating multiple calls to this function using `||` or `STRING_AGG`.



### Important

Do not call with a `GEOMETRYCOLLECTION` as an element in the row. However you can use [ST\\_AsMVTGeom](#) to prepare a geometry collection for inclusion.

`row` row data with at least a geometry column.

`name` is the name of the layer. Default is the string "default".

`extent` is the tile extent in screen space as defined by the specification. Default is 4096.

`geom_name` is the name of the geometry column in the row data. Default is the first geometry column. Note that PostgreSQL by default automatically [folds unquoted identifiers to lower case](#), which means that unless the geometry column is quoted, e.g. "MyMVTGeom", this parameter must be provided as lowercase.

`feature_id_name` is the name of the Feature ID column in the row data. If NULL or negative the Feature ID is not set. The first column matching name and valid type (smallint, integer, bigint) will be used as Feature ID, and any subsequent column will be added as a property. JSON properties are not supported.

Enhanced: 3.0 - added support for Feature ID.

Enhanced: 2.5.0 - added support parallel query.

Availability: 2.4.0

## Example

```
WITH mvtgeom AS
(
  SELECT ST_AsMVTGeom(geom, ST_TileEnvelope(12, 513, 412), extent => 4096, buffer => 64) AS ←
    geom, name, description
  FROM points_of_interest
  WHERE geom && ST_TileEnvelope(12, 513, 412, margin => (64.0 / 4096))
)
SELECT ST_AsMVT(mvtgeom.*)
FROM mvtgeom;
```

## ST\_AsMVT

[ST\\_AsMVTGeom](#), [ST\\_MakeEnvelope](#)

### 8.9.3.11 ST\_AsSVG

`ST_AsSVG` — Returns SVG path data for a geometry.



**array\_agg**

array\_agg(  
TWKB(Tiny Well-Known Binary)  
TWKB  
&#xc0ac;&#xc6a9;&#xc790;&#xc9c0;&#xce68;&#xc11c;  
&#xc124;&#xba85;  
&#xb3c4;&#xd615;&#xc744; TWKB(Tiny Well-Known Binary) &#xd615;&#xc2dd;&#xc73c;&#xb85c; &#xbc18;&#xd658;&#xd569  
TWKB&#xb294; &#xcd9c;&#xb825;&#xbb3c;&#xc758; &#xc6a9;&#xb7c9;&#xc744; &#xcd5c;&#xc18c;&#xd654;&#xd558;&#xb  
&#xb370; &#xc9d1;&#xc911;&#xd55c; &#xc555;&#xcd95; &#bc14;&#xc774;&#xb108;&#xb9ac; &#xd615;&#xc2dd; &#xc785;&#  
&#xc18c;&#xc218;&#xc810; &#xc774;&#xd558; &#xc790;&#xb9bf;&#xc218; &#xd30c;&#xb77c;&#xbbf8;&#xd130;&#xac00;  
&#xcd9c;&#xb825;&#xbb3c;&#xc5d0; &#xc5b4;&#xb290; &#xc815;&#xb3c4;&#xc758; &#xc815;&#xbc00;&#xb3c4;&#xb97c;  
&#xc800;&#xc7a5;&#xd560;&#xc9c0; &#xacb0;&#xc815;&#xd569;&#xb2c8;&#xb2e4;. &#xae30;&#xbcf8;&#xc801;&#xc73c;&#  
&#xaa8;&#xb4e0; &#xac12;&#xc740; &#xc778;&#xcf54;&#xb529;&#xd558;&#xae30; &#xc804;&#xc5d0; &#xac00;&#xc7a5;  
&#xac00;&#xae4c;&#xc6b4; &#xb2e8;&#xc704;&#xb85c; &#bc18;&#xc62c;&#xb9bc;&#xb429;&#xb2c8;&#xb2e4;. &#xb354;  
&#xb192;&#xc740; &#xc815;&#xbc00;&#xb3c4;&#xb97c; &#bcf5;&#xc0ac;&#xd558;&#xae0; &#xc2f6;&#xb2e4;&#xba74;,  
&#xc790;&#xb9bf;&#xc218;&#xb97c; &#xc62c;&#xb9ac;&#xc2ed;&#xc2dc;&#xc624;. &#xc608;&#xb97c; &#xb4e4;&#xc5b4;,  
&#xac12;&#xc774; 1&#xc774;&#xb77c;&#xba74; &#xc18c;&#xc218;&#xc810; &#xc624;&#xb978;&#xcabd;&#xc758; &#xc-  
cab; &#bc88;&#xc9f8; &#xc22b;&#xc790;&#xae4c;&#xc9c0; &#bcf4;&#xc804;&#xb420; &#xac83;&#xc785;&#xb2c8;&#xb2e4;  
&#xd06c;&#xae30; &#bc0f; &#xabd; &#acc4; &#xc0c1;&#xc790; &#xd30c;&#xb77c;&#xbbf8;&#xd130;&#xb294; &#xac1d;&#  
&#xc778;&#xcf54;&#xb529;&#xb41c; &#xae38;&#xc774; &#bc0f; &#xabd; &#acc4; &#xc5d0; &#xb300;&#xd55c; &#xc120;&#  
&#xc815;&#xbcf4;&#xb97c; &#xcd9c;&#xb825;&#xbb3c;&#xc5d0; &#xd3ec;&#xd568;&#xc2dc;&#xd0ac;&#xc9c0; &#xb9d0;&#  
&#xacb0;&#xc815;&#xd569;&#xb2c8;&#xb2e4;. &#xae30;&#xbcf8;&#xc801;&#xc73c;&#xb85c;&#xb294; &#xd3ec;&#xd568;&#  
&#xc54a;&#xc2b5;&#xb2c8;&#xb2e4;. &#xc0ac;&#xc6a9;&#xc790;&#xc758; &#xd074;&#xb77c;&#xc774;&#xc5b8;&#xd2b8;  
&#xc18c;&#xd504;&#xd2b8;&#xc6e8;&#xc5b4;&#xac00; &#xd544;&#xc694;&#xb85c;&#xd558;&#xc9c0; &#xc54a;&#xb294;&#  
&#xd65c;&#xc131;&#xd654;&#xc2dc;&#xd0a4;&#xc9c0; &#xb9c8;&#xc2ed;&#xc2dc;&#xc624;. &#xb514;&#xc2a4;&#xd06c;  
&#xacf5;&#xac04;&#xc744; &#xc18c;&#xbe44;&#xd560; &#bfd0;&#xc785;&#xb2c8;&#xb2e4;(&#xb514;&#xc2a4;&#xd06c;  
&#xacf5;&#xac04;&#xc744; &#xc808;&#xc57d;&#xd558;&#xb294; &#xac83;&#xc774; TWKB&#xc758; &#xaa9;&#xc801;&#xc7  
&#xc774; &#xd568;&#xc218;&#xc758; &#bc30;&#xc5f4; &#xc785;&#xb825; &#xd615;&#xc2dd;&#xc740; &#xb3c4;&#xd615;  
&#xc9d1;&#xd569; &#bc0f; &#xc720;&#xc77c; &#xc2dd;&#xbcc4;&#xc790;&#xb97c; &#xc2dd;&#xbcc4;&#xc790;&#xb97c;  
&#bcf4;&#xc804;&#xd558;&#xb294; TWKB&#xc9d1;&#xd569;&#xc73c;&#xb85c; &#bcc0;&#xd658;&#xd558;&#xb294;  
&#xb370; &#xc4f0;&#xc785;&#xb2c8;&#xb2e4;. &#xc9d1;&#xd569;&#xc758; &#xc555;&#xcd95;&#xc744; &#xd480;&#xc5b4;  
&#xadf8; &#xc548;&#xc5d0; &#xc788;&#xb294; &#xac1d;&#xccb4;&#xb4e4;&#xc5d0; &#xb300;&#xd55c; &#xc0c1;&#xc138;  
&#xc815;&#xbcf4;&#xc5d0; &#xc811;&#xadfc;&#xd558;&#xb294; &#xae30;&#xb2a5;&#xc744; &#xac00;&#xc9c4; &#xd074;&#  
&#xc720;&#xc6a9;&#xd569;&#xb2c8;&#xb2e4;. array\_agg &#xd568;&#xc218;&#xb97c; &#xc774;&#xc6a9;&#xd574;&#xc11c;  
&#bc30;&#xc5f4;&#xc744; &#xc0dd;&#xc131;&#xd560; &#xc218; &#xc788;&#xc2b5;&#xb2c8;&#xb2e4;. &#xb2e4;&#xb978;  
&#xd30c;&#xb77c;&#xbbf8;&#xd130;&#xb4e4;&#xc740; &#xc774; &#xd568;&#xc218;&#xc758; &#xb2e8;&#xc21c; &#xd615;&#  
&#xabd;&#xc6b0;&#xc640; &#xb3d9;&#xc77c;&#xd558;&#xac8c; &#xc2e4;&#xd589;&#xb429;&#xb2c8;&#xb2e4;.



**Note**

<https://github.com/TWKB/Specification> &#xc5d0;&#xc11c; &#xd615;&#xc2dd; &#xc0ac;&#xc591;&#xc11c;&#xb97c;  
&#xcc3e;&#xc544;&#xbcf; &#xc218; &#xc788;&#xc73c;&#xba70;,, [https://github.com/  
TWKB/twkb.js](https://github.com/TWKB/twkb.js) &#xc5d0;&#xc11c; &#xc790;&#xb3c4;&#xc2a4;&#xd06c;&#xb9bd;&#xd2b8;  
&#xd074;&#xb77c;&#xc774;&#xc5b8;&#xd2b8;&#xb97c; &#xbe4c;&#xb4dc;&#xd558;&#xae30; &#xc704;&#xd55c;  
&#xcf54;&#xb4dc;&#xb97c; &#xcc3e;&#xc744; &#xc218; &#xc788;&#xc2b5;&#xb2c8;&#xb2e4;.

Enhanced: 2.4.0 memory and speed improvements.

2.2.0 &#xbc84;&#xc804;&#xbd80;&#xd130; &#xc0ac;&#xc6a9;&#xd560; &#xc218; &#xc788;&#xc2b5;&#xb2c8;&#xb2e4;.

**array\_agg**

```
SELECT ST_AsTWKB('LINESTRING(1 1,5 5)')::geometry);
-----
st_astwkb
-----
\x02000202020808
```

&#xc2dd;&#xbcc4;&#xc790;&#xb97c; &#xd3ec;&#xd568;&#xd558;&#xb294; &#xc885;&#xd569; TWKB &#xac1d;&#xcbb4;&#xb  
&#xc0dd;&#xc131;&#xd558;&#xb824;&#xba74;,, &#xba3c;&#xc800; "array\_agg()"&#xb97c; &#xd1b5;&#xd574; &#xc6d0;&#xd55







```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE X3D PUBLIC "ISO//Web3D//DTD X3D 3.0//EN" "http://www.web3d.org/specifications/x3d ←
-3.0.dtd">
<X3D>
  <Scene>
    <Transform>
      <Shape>
        <Appearance>
          <Material emissiveColor='0 0 1' />
        </Appearance>
        <IndexedFaceSet coordIndex='0 1 2 3 -1 4 5 6 7 -1 8 9 10 11 -1 12 13 14 15 -1 16 17 ←
18 19 -1 20 21 22 23'>
          <Coordinate point='0 0 0 0 0 1 0 1 1 0 1 0 0 0 0 0 1 0 1 1 0 1 0 0 0 0 0 1 0 0 ←
1 0 1 0 0 1 1 1 0 1 1 1 0 1 1 0 0 0 1 0 0 1 1 1 1 1 1 1 0 0 0 1 1 0 1 1 1 ←
1 0 1 1' />
        </IndexedFaceSet>
      </Shape>
    </Transform>
  </Scene>
</X3D>
>
```

## PostGIS buildings

Copy and paste the output of this query to [x3d scene viewer](#) and click Show

```
SELECT string_agg('<Shape>' || ST_AsX3D(ST_Extrude(geom, 0,0, i*0.5)) ||
  '<Appearance>' ||
    '<Material diffuseColor="' || (0.01*i)::text || ' 0.8 0.2" specularColor="' || ←
    (0.05*i)::text || ' 0 0.5"/>' ||
  '</Appearance>' ||
  '</Shape>', '')
FROM ST_Subdivide(ST_Letters('PostGIS'),20) WITH ORDINALITY AS f(geom,i);
```



*Buildings formed by subdividing PostGIS and extrusion*

**PostGIS**

```
SELECT ST_AsX3D(
  ST_Translate(
    ST_Force_3d(
      ST_Buffer(ST_Point(10,10),5, 'quad_segs=2')), 0,0,
    3)
  ,6) As x3dfrag;
```

```
x3dfrag
-----
<IndexedFaceSet coordIndex="0 1 2 3 4 5 6 7">
  <Coordinate point="15 10 3 13.535534 6.464466 3 10 5 3 6.464466 6.464466 3 5 10 3  ←
    6.464466 13.535534 3 10 15 3 13.535534 13.535534 3 " />
</IndexedFaceSet
>
```

### TIN

```
SELECT ST_AsX3D(ST_GeomFromEWKT('TIN (((
    0 0 0,
    0 0 1,
    0 1 0,
    0 0 0
  )), ((
    0 0 0,
    0 1 0,
    1 1 0,
    0 0 0
  ))
)')) As x3dfrag;

x3dfrag
-----
<IndexedTriangleSet index='0 1 2 3 4 5'
><Coordinate point='0 0 0 0 1 0 1 0 0 0 0 1 0 1 1 0' /></IndexedTriangleSet
>
```

### MULTILINESTRING((20 0 10,16 -12 10,0 -16 10,-12 -12 10,-20 0 10,-12 16 10,0 24 10,16 16 10,20 0 10), (12 0 10,8 8 10,0 12 10,-8 8 10,-8 0 10,-8 -4 10,0 -8 10,8 -4 10,12 0 10))

```
SELECT ST_AsX3D(
  ST_GeomFromEWKT('MULTILINESTRING((20 0 10,16 -12 10,0 -16 10,-12 -12  ←
    10,-20 0 10,-12 16 10,0 24 10,16 16 10,20 0 10),
    (12 0 10,8 8 10,0 12 10,-8 8 10,-8 0 10,-8 -4 10,0 -8 10,8 -4 10,12 0 10))')
) As x3dfrag;

x3dfrag
-----
<IndexedLineSet coordIndex='0 1 2 3 4 5 6 7 0 -1 8 9 10 11 12 13 14 15 8'>
  <Coordinate point='20 0 10 16 -12 10 0 -16 10 -12 -12 10 -20 0 10 -12 16 10 0 24 10 16  ←
    16 10 12 0 10 8 8 10 0 12 10 -8 8 10 -8 0 10 -8 -4 10 0 -8 10 8 -4 10 ' />
</IndexedLineSet
>
```

#### 8.9.3.14 ST\_GeoHash

ST\_GeoHash — GeoHash

#### Synopsis

text ST\_GeoHash(geometry geom, integer maxchars=full\_precision\_of\_point);





### 8.10.1.2 `&&(geometry,box2df)`

`&&(geometry,box2df)` — Returns TRUE if a geometry's (cached) 2D bounding box intersects a 2D float precision bounding box (BOX2DF).

#### Synopsis

boolean `&&( geometry A , box2df B );`

#### Notes

The `&&` operator returns TRUE if the cached 2D bounding box of geometry A intersects the 2D bounding box B, using float precision. This means that if B is a (double precision) box2d, it will be internally converted to a float precision 2D bounding box (BOX2DF)



#### Note

This operand is intended to be used internally by BRIN indexes, more than by users.

Availability: 2.3.0 support for Block Range INdexes (BRIN) was introduced. Requires PostgreSQL 9.5+.



This method supports Circular Strings and Curves



This function supports Polyhedral surfaces.

#### Example

```
SELECT ST_Point(1,1) && ST_MakeBox2D(ST_Point(0,0), ST_Point(2,2)) AS overlaps;

 overlaps
-----
 t
(1 row)
```

#### Operator Precedence

`&&(box2df,geometry)`, `&&(box2df,box2df)`, `~(geometry,box2df)`, `~(box2df,geometry)`, `~(box2df,box2df)`, `@(geometry,box2df)`, `@(box2df,geometry)`, `@(box2df,box2df)`

### 8.10.1.3 `&&(box2df,geometry)`

`&&(box2df,geometry)` — Returns TRUE if a 2D float precision bounding box (BOX2DF) intersects a geometry's (cached) 2D bounding box.

#### Synopsis

boolean `&&( box2df A , geometry B );`

**8.10.1.3**

The `&&` operator returns `TRUE` if the 2D bounding box A intersects the cached 2D bounding box of geometry B, using float precision. This means that if A is a (double precision) `box2d`, it will be internally converted to a float precision 2D bounding box (`BOX2DF`)

**Note**

This operand is intended to be used internally by BRIN indexes, more than by users.

Availability: 2.3.0 support for Block Range INdexes (BRIN) was introduced. Requires PostgreSQL 9.5+.



This method supports Circular Strings and Curves



This function supports Polyhedral surfaces.

**8.10.1.4**

```
SELECT ST_MakeBox2D(ST_Point(0,0), ST_Point(2,2)) && ST_Point(1,1) AS overlaps;

overlaps
-----
t
(1 row)
```

**8.10.1.5**

`&&(geometry,box2df)`, `&&(box2df,box2df)`, `~(geometry,box2df)`, `~(box2df,geometry)`, `~(box2df,box2df)`, `@(geometry,box2df)`, `@(box2df,geometry)`, `@(box2df,box2df)`

**8.10.1.4** `&&(box2df,box2df)`

`&&(box2df,box2df)` — Returns `TRUE` if two 2D float precision bounding boxes (`BOX2DF`) intersect each other.

**Synopsis**

boolean `&&( box2df A , box2df B )`;

**8.10.1.5**

The `&&` operator returns `TRUE` if two 2D bounding boxes A and B intersect each other, using float precision. This means that if A (or B) is a (double precision) `box2d`, it will be internally converted to a float precision 2D bounding box (`BOX2DF`)

**Note**

This operator is intended to be used internally by BRIN indexes, more than by users.

Availability: 2.3.0 support for Block Range INdexes (BRIN) was introduced. Requires PostgreSQL 9.5+.



This method supports Circular Strings and Curves



This function supports Polyhedral surfaces.

**overlaps;**

```
SELECT ST_MakeBox2D(ST_Point(0,0), ST_Point(2,2)) && ST_MakeBox2D(ST_Point(1,1), ST_Point(3,3)) AS overlaps;

overlaps
-----
t
(1 row)
```

**overlaps\_3d;**

**overlaps\_3d(geometry, box2df), overlaps\_3d(box2df, geometry), ~overlaps\_3d(geometry, box2df), ~overlaps\_3d(box2df, geometry), @overlaps\_3d(geometry, box2df), @overlaps\_3d(box2df, geometry), @overlaps\_3d(box2df, box2df)**

**8.10.1.5 overlaps\_3d**

**overlaps\_3d** — Returns true if the two 3D geometries overlap. The geometries must be of the same SRID. The function returns true if the two geometries intersect, but do not touch. The function returns false if the two geometries are disjoint, or if they touch at a point, line, or surface.

**Synopsis**

boolean **overlaps\_3d**( geometry A , geometry B );

**overlaps\_3d(geometry, box2df);**

**overlaps\_3d(geometry, box2df)** Returns true if the 3D geometry overlaps the 2D box2df. The function returns true if the geometry overlaps the box2df, but does not touch it. The function returns false if the geometry is disjoint from the box2df, or if they touch at a point, line, or surface.

**Note**

**overlaps\_3d(geometry, box2df)** does not support 3D geometries. The function returns false for 3D geometries. The function returns true for 2D geometries that overlap the box2df.

**overlaps\_3d(box2df, geometry);** Returns true if the 2D box2df overlaps the 3D geometry. The function returns true if the box2df overlaps the geometry, but does not touch it. The function returns false if the box2df is disjoint from the geometry, or if they touch at a point, line, or surface.



This method supports Circular Strings and Curves



This function supports Polyhedral surfaces.



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



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

**overlaps\_3d(geometry, box2df); 3d overlaps\_3d(box2df, geometry);**

```
SELECT tbl1.column1, tbl2.column1, tbl1.column2 && tbl2.column2 AS overlaps_3d,
       tbl1.column2 && tbl2.column2 AS overlaps_2d
FROM ( VALUES
      (1, 'LINESTRING Z(0 0 1, 3 3 2)::geometry),
      (2, 'LINESTRING Z(1 2 0, 0 5 -1)::geometry)) AS tbl1,
```

```
( VALUES
  (3, 'LINESTRING Z(1 2 1, 4 6 1)::geometry) AS tbl2;
```

| column1 | column1 | overlaps_3d | overlaps_2d |
|---------|---------|-------------|-------------|
| 1       | 3       | t           | t           |
| 2       | 3       | f           | t           |

### 3DM &#x26a9; &#x26a9; &#x26a9;

```
SELECT tbl1.column1, tbl2.column1, tbl1.column2 && tbl2.column2 AS overlaps_3zm,
       tbl1.column2 && tbl2.column2 AS overlaps_2d
```

```
FROM ( VALUES
  (1, 'LINESTRING M(0 0 1, 3 3 2)::geometry),
  (2, 'LINESTRING M(1 2 0, 0 5 -1)::geometry)) AS tbl1,
( VALUES
  (3, 'LINESTRING M(1 2 1, 4 6 1)::geometry) AS tbl2;
```

| column1 | column1 | overlaps_3zm | overlaps_2d |
|---------|---------|--------------|-------------|
| 1       | 3       | t            | t           |
| 2       | 3       | f            | t           |

### &#x26a9;

&&

#### 8.10.1.6 &&&(geometry,gidx)

&&&(geometry,gidx) — Returns TRUE if a geometry's (cached) n-D bounding box intersects a n-D float precision bounding box (GIDX).

#### Synopsis

boolean &&&( geometry A , gidx B );

### &#x26a9;

The &&& operator returns TRUE if the cached n-D bounding box of geometry A intersects the n-D bounding box B, using float precision. This means that if B is a (double precision) box3d, it will be internally converted to a float precision 3D bounding box (GIDX)



#### Note

This operator is intended to be used internally by BRIN indexes, more than by users.

Availability: 2.3.0 support for Block Range INdexes (BRIN) was introduced. Requires PostgreSQL 9.5+.



This method supports Circular Strings and Curves



This function supports Polyhedral surfaces.



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



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

**Example 1**

```
SELECT ST_MakePoint(1,1,1) &&& ST_3DMakeBox(ST_MakePoint(0,0,0), ST_MakePoint(2,2,2)) AS overlaps;

overlaps
-----
t
(1 row)
```

**Example 2**

```
&&&(gidx,geometry), &&&(gidx,gidx)
```

**8.10.1.7 &&&(gidx,geometry)**

**&&&(gidx,geometry)** — Returns TRUE if a n-D float precision bounding box (GIDX) intersects a geometry's (cached) n-D bounding box.

**Synopsis**

boolean **&&&**( gidx A , geometry B );

**Notes**

The **&&&** operator returns TRUE if the n-D bounding box A intersects the cached n-D bounding box of geometry B, using float precision. This means that if A is a (double precision) box3d, it will be internally converted to a float precision 3D bounding box (GIDX)

**Note**

This operator is intended to be used internally by BRIN indexes, more than by users.

Availability: 2.3.0 support for Block Range INdexes (BRIN) was introduced. Requires PostgreSQL 9.5+.



This method supports Circular Strings and Curves



This function supports Polyhedral surfaces.



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



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

**Example 3**

```
SELECT ST_3DMakeBox(ST_MakePoint(0,0,0), ST_MakePoint(2,2,2)) &&& ST_MakePoint(1,1,1) AS overlaps;

overlaps
-----
t
(1 row)
```

**Geometry**

`ST_3DMakeBox(geometry, gidx, gidx)`

### 8.10.1.8 ST\_3DMakeBox

`ST_3DMakeBox` — Returns TRUE if two n-D float precision bounding boxes (GIDX) intersect each other.

#### Synopsis

boolean `ST_3DMakeBox`( gidx A , gidx B );

**Notes**

The `ST_3DMakeBox` operator returns TRUE if two n-D bounding boxes A and B intersect each other, using float precision. This means that if A (or B) is a (double precision) box3d, it will be internally converted to a float precision 3D bounding box (GIDX)



#### Note

This operator is intended to be used internally by BRIN indexes, more than by users.

Availability: 2.3.0 support for Block Range INdexes (BRIN) was introduced. Requires PostgreSQL 9.5+.



This method supports Circular Strings and Curves



This function supports Polyhedral surfaces.



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



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

**Example**

```
SELECT ST_3DMakeBox(ST_MakePoint(0,0,0), ST_MakePoint(2,2,2)) &&& ST_3DMakeBox(ST_MakePoint(1,1,1), ST_MakePoint(3,3,3)) AS overlaps;
```

```
overlaps
-----
t
(1 row)
```

**Geometry**

`ST_3DMakeBox(geometry, gidx, geometry)`

### 8.10.1.9 ST\_3DMakeBox

`ST_3DMakeBox` — A 3D bounding box (GIDX) for the geometry. B is a 3D bounding box (GIDX). TRUE if A and B intersect each other, using float precision. TRUE if A and B intersect each other, using float precision. TRUE if A and B intersect each other, using float precision.

## Synopsis

boolean `&<( geometry A , geometry B );`

### Notes

`&<` returns true if the two geometries intersect. The intersection of two geometries is the set of points that are common to both geometries. The intersection of two lines is a point, a line segment, or a line. The intersection of two polygons is a polygon, a line segment, or a point. The intersection of two surfaces is a surface, a line segment, or a point.



#### Note

The `&<` function is implemented as a wrapper around the `ST_Intersection` function. The `ST_Intersection` function returns the intersection of two geometries as a geometry. The `&<` function returns a boolean value indicating whether the intersection is non-empty.

### Examples

```
SELECT tbl1.column1, tbl2.column1, tbl1.column2 &< tbl2.column2 AS overleft
FROM
  ( VALUES
    (1, 'LINESTRING(1 2, 4 6)::geometry) AS tbl1,
  ( VALUES
    (2, 'LINESTRING(0 0, 3 3)::geometry),
    (3, 'LINESTRING(0 1, 0 5)::geometry),
    (4, 'LINESTRING(6 0, 6 1)::geometry) AS tbl2;
```

| column1 | column1 | overleft |
|---------|---------|----------|
| 1       | 2       | f        |
| 1       | 3       | f        |
| 1       | 4       | t        |

(3 rows)

### See also

`&&`, `!&>`, `&>`, `&<`

#### 8.10.1.10 `&<`

`&<` — A `&<` B returns true if the two geometries intersect. The intersection of two geometries is the set of points that are common to both geometries. The intersection of two lines is a point, a line segment, or a line. The intersection of two polygons is a polygon, a line segment, or a point. The intersection of two surfaces is a surface, a line segment, or a point.

## Synopsis

boolean `&<( geometry A , geometry B );`







```

column1 | column1 | below
-----+-----+-----
         |         |
         1 |         2 | t
         1 |         3 | f
         1 |         4 | f
(3 rows)

```

## 8.10.1.13

<<, >>, |>>

### 8.10.1.14 =

= — Returns TRUE if the coordinates and coordinate order geometry/geography A are the same as the coordinates and coordinate order of geometry/geography B.

#### Synopsis

```
boolean =( geometry A , geometry B );
boolean =( geography A , geography B );
```

## 8.10.1.15

The = operator returns TRUE if the coordinates and coordinate order geometry/geography A are the same as the coordinates and coordinate order of geometry/geography B. PostgreSQL uses the =, <, and > operators defined for geometries to perform internal orderings and comparison of geometries (ie. in a GROUP BY or ORDER BY clause).



#### Note

Only geometry/geography that are exactly equal in all respects, with the same coordinates, in the same order, are considered equal by this operator. For "spatial equality", that ignores things like coordinate order, and can detect features that cover the same spatial area with different representations, use [ST\\_OrderingEquals](#) or [ST\\_Equals](#)



#### Caution

This operand will NOT make use of any indexes that may be available on the geometries. For an index assisted exact equality test, combine = with &&.

Changed: 2.4.0, in prior versions this was bounding box equality not a geometric equality. If you need bounding box equality, use ~= instead.



This method supports Circular Strings and Curves



This function supports Polyhedral surfaces.

## 8.10.1.16

```

SELECT 'LINESTRING(0 0, 0 1, 1 0)::geometry = 'LINESTRING(1 1, 0 0)::geometry;
?column?
-----
f
(1 row)

```

```

SELECT ST_AsText(column1)
FROM ( VALUES
      ('LINESTRING(0 0, 1 1)::geometry),
      ('LINESTRING(1 1, 0 0)::geometry)) AS foo;
      st_astext
-----
LINESTRING(0 0,1 1)
LINESTRING(1 1,0 0)
(2 rows)

-- Note: the GROUP BY uses the "=" to compare for geometry equivalency.
SELECT ST_AsText(column1)
FROM ( VALUES
      ('LINESTRING(0 0, 1 1)::geometry),
      ('LINESTRING(1 1, 0 0)::geometry)) AS foo
GROUP BY column1;
      st_astext
-----
LINESTRING(0 0,1 1)
LINESTRING(1 1,0 0)
(2 rows)

-- In versions prior to 2.0, this used to return true --
SELECT ST_GeomFromText('POINT(1707296.37 4820536.77)') =
      ST_GeomFromText('POINT(1707296.27 4820536.87)') As pt_intersect;

--pt_intersect --
f

```

## ST\_Equals

**ST\_Equals, ST\_OrderingEquals, ~=**

### 8.10.1.15 >>

>> — A &#x758; &#xacbd; &#xacc4; &#xc0c1; &#xc790; &#xac00; &#xc624; &#xb85c; &#xc9c0; B &#x758; &#xacbd; &#xacc4; &#xc0c1; &#xc790; &#xc624; &#xb978; &#xcabd; &#xc5d0; &#xc788; &#xc744; &#xacbd; &#xc6b0; &#xc5d0; &#xb9cc; TRUE &#xb97c; &#xbc18; &#xd658; &#xd569; &#xb2c8; &#xb2e4;.

## Synopsis

boolean >>( geometry A , geometry B );

## &#xc124; &#xba85;

>> &#xc5f0; &#xc0b0; &#xc790; &#xb294; &#xb3c4; &#xd615; A &#x758; &#xacbd; &#xacc4; &#xc0c1; &#xc790; &#xac00; &#xc624; &#xb85c; &#xc9c0; &#xb3c4; &#xd615; B &#x758; &#xacbd; &#xacc4; &#xc0c1; &#xc790; &#xc624; &#xb978; &#xcabd; &#xc788; &#xc744; &#xacbd; &#xc6b0; &#xc5d0; &#xb9cc; TRUE &#xb97c; &#xbc18; &#xd658; &#xd569; &#xb2c8; &#xb2e4;.



### Note

&#xc774; &#xd53c; &#xc5f0; &#xc0b0; &#xc790; (operand) &#xb294; &#xb3c4; &#xd615; &#xc5d0; &#xc11c;  
 &#xc774; &#xc6a9; &#xd560; &#xc218; &#xb3c4; &#xc788; &#xb294; &#xbaa8; &#xb4e0;  
 &#xc778; &#xb371; &#xc2a4; &#xb97c; &#xd65c; &#xc6a9; &#xd560; &#xac83; &#xc785; &#xb2c8; &#xb2e4;.

**8.10.15**

```
SELECT tbl1.column1, tbl2.column1, tbl1.column2 >> tbl2.column2 AS right
FROM
  ( VALUES
    (1, 'LINESTRING (2 3, 5 6)::geometry) AS tbl1,
  ( VALUES
    (2, 'LINESTRING (1 4, 1 7)::geometry),
    (3, 'LINESTRING (6 1, 6 5)::geometry),
    (4, 'LINESTRING (0 0, 4 3)::geometry) AS tbl2;

column1 | column1 | right
-----+-----+-----
        1 |         2 | t
        1 |         3 | f
        1 |         4 | f
(3 rows)
```

**8.10.16**

&lt;&lt;, &gt;&gt;, &lt;&lt;

**8.10.16 @**

@ — B &#x2264; A; A &#x2264; B; TRUE; FALSE

**Synopsis**

boolean @( geometry A , geometry B );

**8.10.16.1**

@ ( geometry A , geometry B ) — TRUE if A is contained within B, FALSE otherwise.

**Note**

When comparing two geometry objects, the result is TRUE if the first geometry is contained within the second geometry. If the first geometry is not contained within the second geometry, the result is FALSE.

**8.10.17**

```
SELECT tbl1.column1, tbl2.column1, tbl1.column2 @ tbl2.column2 AS contained
FROM
  ( VALUES
    (1, 'LINESTRING (1 1, 3 3)::geometry) AS tbl1,
  ( VALUES
    (2, 'LINESTRING (0 0, 4 4)::geometry),
    (3, 'LINESTRING (2 2, 4 4)::geometry),
    (4, 'LINESTRING (1 1, 3 3)::geometry) AS tbl2;

column1 | column1 | contained
```

```

-----+-----+-----
      1 |      2 | t
      1 |      3 | f
      1 |      4 | t
(3 rows)

```

**Notes:**

~, &&

### 8.10.1.17 @ (geometry, box2df)

@ (geometry, box2df) — Returns TRUE if a geometry's 2D bounding box is contained into a 2D float precision bounding box (BOX2DF).

#### Synopsis

```
boolean @( geometry A , box2df B );
```

**Notes:**

The @ operator returns TRUE if the A geometry's 2D bounding box is contained the 2D bounding box B, using float precision. This means that if B is a (double precision) box2d, it will be internally converted to a float precision 2D bounding box (BOX2DF).



#### Note

This operand is intended to be used internally by BRIN indexes, more than by users.

Availability: 2.3.0 support for Block Range INdexes (BRIN) was introduced. Requires PostgreSQL 9.5+.



This method supports Circular Strings and Curves



This function supports Polyhedral surfaces.

**Notes:**

```

SELECT ST_Buffer(ST_GeomFromText('POINT(2 2)'), 1) @ ST_MakeBox2D(ST_Point(0,0), ST_Point(
      (5,5)) AS is_contained;

 is_contained
-----
      t
(1 row)

```

**Notes:**

&&(geometry, box2df), &&(box2df, geometry), &&(box2df, box2df), ~(geometry, box2df), ~(box2df, geometry), ~(box2df, box2df), @ (box2df, geometry), @ (box2df, box2df)

### 8.10.1.18 @(**box2df,geometry**)

@(**box2df,geometry**) — Returns `TRUE` if a 2D float precision bounding box (BOX2DF) is contained into a geometry's 2D bounding box.

#### Synopsis

```
boolean @( box2df A , geometry B );
```

#### Notes

The @ operator returns `TRUE` if the 2D bounding box A is contained into the B geometry's 2D bounding box, using float precision. This means that if B is a (double precision) box2d, it will be internally converted to a float precision 2D bounding box (BOX2DF)



#### Note

This operand is intended to be used internally by BRIN indexes, more than by users.

Availability: 2.3.0 support for Block Range INdices (BRIN) was introduced. Requires PostgreSQL 9.5+.



This method supports Circular Strings and Curves



This function supports Polyhedral surfaces.

#### Examples

```
SELECT ST_MakeBox2D(ST_Point(2,2), ST_Point(3,3)) @ ST_Buffer(ST_GeomFromText('POINT(1 1)') ←
, 10) AS is_contained;
```

```
is_contained
-----
t
(1 row)
```

#### See also

[ST\\_Contains](#), [ST\\_ContainsProperly](#), [ST\\_ContainsWithin](#), [ST\\_ContainsStrictly](#), [ST\\_ContainsProperlyStrictly](#), [ST\\_ContainsWithinStrictly](#), [ST\\_ContainsProperlyWithin](#), [ST\\_ContainsProperlyWithinStrictly](#), [ST\\_ContainsWithinProperly](#), [ST\\_ContainsWithinProperlyStrictly](#)

### 8.10.1.19 @(**box2df,box2df**)

@(**box2df,box2df**) — Returns `TRUE` if a 2D float precision bounding box (BOX2DF) is contained into another 2D float precision bounding box.

#### Synopsis

```
boolean @( box2df A , box2df B );
```





```

      1 |      3 | f
      1 |      4 | f
(3 rows)

```

**Navigation:**

<<, >>, <|

### 8.10.1.22 ~

~ — A **contains** B; TRUE if the geometry A contains the geometry B.

#### Synopsis

boolean `~( geometry A , geometry B );`

**Notes:**

~ — A **contains** B; TRUE if the geometry A contains the geometry B.



#### Note

~ — A **contains** B; TRUE if the geometry A contains the geometry B.

**Example:**

```

SELECT tbl1.column1, tbl2.column1, tbl1.column2 ~ tbl2.column2 AS contains
FROM
  ( VALUES
    (1, 'LINESTRING (0 0, 3 3)::geometry) AS tbl1,
  ( VALUES
    (2, 'LINESTRING (0 0, 4 4)::geometry),
    (3, 'LINESTRING (1 1, 2 2)::geometry),
    (4, 'LINESTRING (0 0, 3 3)::geometry) AS tbl2;

```

```

column1 | column1 | contains
-----+-----+-----
      1 |      2 | f
      1 |      3 | t
      1 |      4 | t
(3 rows)

```

**Navigation:**

@, &&

### 8.10.1.23 ~(geometry,box2df)

~(geometry,box2df) — Returns TRUE if a geometry's 2D bounding box contains a 2D float precision bounding box (GIDX).

## Synopsis

boolean `~( geometry A , box2df B );`

### Notes

The `~` operator returns `TRUE` if the 2D bounding box of a geometry A contains the 2D bounding box B, using float precision. This means that if B is a (double precision) box2d, it will be internally converted to a float precision 2D bounding box (BOX2DF)



#### Note

This operand is intended to be used internally by BRIN indexes, more than by users.

Availability: 2.3.0 support for Block Range INdexes (BRIN) was introduced. Requires PostgreSQL 9.5+.



This method supports Circular Strings and Curves



This function supports Polyhedral surfaces.

### Examples

```
SELECT ST_Buffer(ST_GeomFromText('POINT(1 1)'), 10) ~ ST_MakeBox2D(ST_Point(0,0), ST_Point(2,2)) AS contains;
```

```
contains
-----
t
(1 row)
```

### Operator Syntax

`~(geometry,box2df)`, `~(box2df,geometry)`, `~(box2df,box2df)`, `~(box2df,geometry)`, `~(box2df,box2df)`, `@(geometry,box2df)`, `@(box2df,geometry)`, `@(box2df,box2df)`

#### 8.10.1.24 `~(box2df,geometry)`

`~(box2df,geometry)` — Returns `TRUE` if a 2D float precision bounding box (BOX2DF) contains a geometry's 2D bounding box.

## Synopsis

boolean `~( box2df A , geometry B );`

### Notes

The `~` operator returns `TRUE` if the 2D bounding box A contains the B geometry's bounding box, using float precision. This means that if A is a (double precision) box2d, it will be internally converted to a float precision 2D bounding box (BOX2DF)



#### Note

This operand is intended to be used internally by BRIN indexes, more than by users.

Availability: 2.3.0 support for Block Range INdices (BRIN) was introduced. Requires PostgreSQL 9.5+.



This method supports Circular Strings and Curves



This function supports Polyhedral surfaces.

### Examples

```
SELECT ST_MakeBox2D(ST_Point(0,0), ST_Point(5,5)) ~ ST_Buffer(ST_GeomFromText('POINT(2 2)')
, 1) AS contains;
```

```
contains
-----
t
(1 row)
```

### Operator

`~(geometry,box2df), ~(box2df,geometry), ~(box2df,box2df), ~(geometry,box2df), ~(box2df,box2df), @(geometry,box2df), @(box2df,geometry), @(box2df,box2df)`

#### 8.10.1.25 `~(box2df,box2df)`

`~(box2df,box2df)` — Returns `TRUE` if a 2D float precision bounding box (BOX2DF) contains another 2D float precision bounding box (BOX2DF).

### Synopsis

boolean `~( box2df A , box2df B );`

### Notes

The `~` operator returns `TRUE` if the 2D bounding box A contains the 2D bounding box B, using float precision. This means that if A is a (double precision) `box2d`, it will be internally converted to a float precision 2D bounding box (BOX2DF)



#### Note

This operand is intended to be used internally by BRIN indexes, more than by users.

Availability: 2.3.0 support for Block Range INdices (BRIN) was introduced. Requires PostgreSQL 9.5+.



This method supports Circular Strings and Curves



This function supports Polyhedral surfaces.

### Examples

```
SELECT ST_MakeBox2D(ST_Point(0,0), ST_Point(5,5)) ~ ST_MakeBox2D(ST_Point(2,2), ST_Point
(3,3)) AS contains;
```

```
contains
-----
t
(1 row)
```

**Usage**

`ST_Equals(geometry, geometry)`, `ST_OrderingEquals(geometry, geometry)`, `ST_Contains(geometry, geometry)`, `ST_ContainsProperly(geometry, geometry)`, `ST_Within(geometry, geometry)`, `ST_WithinProperly(geometry, geometry)`, `ST_ContainsProperly(geometry, geometry)`, `ST_Contains(geometry, geometry)`, `ST_Within(geometry, geometry)`, `ST_WithinProperly(geometry, geometry)`

**8.10.1.26 ~ =**

`~ =` — A `LINESTRING`; B `LINESTRING`; TRUE if A is a `LINESTRING` that contains B, FALSE otherwise.

**Synopsis**

boolean `~ =`( geometry A , geometry B );

**Notes**

`~ =` is a `LINESTRING` containment test. It returns TRUE if A is a `LINESTRING` that contains B, FALSE otherwise.



**Note**

`ST_Equals` and `ST_Contains` are `LINESTRING` containment tests. They return TRUE if A is a `LINESTRING` that contains B, FALSE otherwise.

1.5.0 `ST_Equals` and `ST_Contains` are `LINESTRING` containment tests. They return TRUE if A is a `LINESTRING` that contains B, FALSE otherwise.



This function supports Polyhedral surfaces.



**Warning**

This operator has changed behavior in PostGIS 1.5 from testing for actual geometric equality to only checking for bounding box equality. To complicate things it also depends on if you have done a hard or soft upgrade which behavior your database has. To find out which behavior your database has you can run the query below. To check for true equality use `ST_OrderingEquals` or `ST_Equals`.

**Example**

```
select 'LINESTRING(0 0, 1 1)::geometry ~ = 'LINESTRING(0 1, 1 0)::geometry as equality;
equality
-----+
t      |
```

**Usage**

`ST_Equals`, `ST_OrderingEquals`, `=`

**8.10.2 ST\_Equals(geometry, geometry)**

**8.10.2.1 <->**

`<->` — A `LINESTRING`; B `LINESTRING`; TRUE if A is a `LINESTRING` that contains B, FALSE otherwise.

Synopsis

double precision <->( geometry A , geometry B );
double precision <->( geography A , geography B );

&#xc124;&#xba85;

<-> &#xc5f0;&#xc0b0;&#xc790;&#xb294; &#xb450; &#xb3c4;&#xd615; &#xc0ac;&#xc774;&#xc758; 2&#xcc28;&#xc6d0;
&#xac70;&#xb9ac;&#xb97c; &#xbc18;&#xd658;&#xd569;&#xb2c8;&#xb2e4;. "ORDER BY" &#xc808; &#xc548;&#xc5d0;&#xc11c;
&#xc774;&#xc6a9;&#xd558;&#xba74; &#xc778;&#xb371;&#xc2a4; &#xc9c0;&#xc6d0;&#xd615;(index-assisted) &#xcd5c;&#xadf8;
neighbor) &#xacb0;&#xacfc; &#xc9d1;&#xd569;&#xc744; &#xcd9c;&#xb825;&#xd569;&#xb2c8;&#xb2e4;. PostgreSQL 9.5
&#xbbf8;&#xb9cc; &#xbc84;&#xc804;&#xc740; &#xacbd;&#xac4; &#xc0c1;&#xc790;&#xc758; &#xc911;&#xc2ec;&#xc810;
&#xc0ac;&#xc774;&#xc758; &#xac70;&#xb9ac;&#xb9cc; &#xcd9c;&#xb825;&#xd558;&#xace0;, 9.5 &#xc774;&#xc0c1; &#xbc84;
&#xb3c4;&#xd615; &#xc0ac;&#xc774;&#xc758; &#xc9c4;&#xc815;&#xd55c; &#xac70;&#xb9ac;, &#xadf8;&#xb9ac;&#xace0;
&#xc9c0;&#xb9ac;&#xd615; &#xc0ac;&#xc774;&#xc758; &#xad6c;&#xba74; &#xac70;&#xb9ac;(distance sphere)&#xb97c;
&#xc0b0;&#xcd9c;&#xd558;&#xb294; &#xc9c4;&#xc815;&#xd55c; KNN &#xac70;&#xb9ac; &#xd0d0;&#xc0c9;&#xc744;
&#xc2e4;&#xd589;&#xd569;&#xb2c8;&#xb2e4;.

Note



&#xc774; &#xd53c;&#xc5f0;&#xc0b0;&#xc790;(operand)&#xb294; &#xb3c4;&#xd615;&#xc5d0; &#xb300;&#xd574;
&#xc774;&#xc6a9;&#xd560; &#xc218; &#xc788;&#xc744;&#xc9c0;&#xb3c4; &#xaa8;&#xb974;&#xb294;
2&#xcc28;&#xc6d0; GiST &#xc778;&#xb371;&#xc2a4;&#xb97c; &#xd65c;&#xc6a9;&#xd560;
&#xac83;&#xc785;&#xb2c8;&#xb2e4;. &#xc5f0;&#xc0b0;&#xc790;&#xc00; ORDER BY &#xc808;
&#xc548;&#xc5d0; &#xc788;&#xc744; &#xb54c;&#xb9cc; &#xacf5;&#xac04; &#xc778;&#xb371;&#xc2a4;&#xb97c;
&#xc4f4;&#xb2e4;&#xb294; &#xc810;&#xc5d0;&#xc11c; &#xacf5;&#xac04; &#xc778;&#xb371;&#xc2a4;&#xb97c;
&#xc774;&#xc6a9;&#xd558;&#xb294; &#xb2e4;&#xb978; &#xc5f0;&#xc0b0;&#xc790;&#xb4e4;&#xacfc;&#xb294;
&#xb2e4;&#xb985;&#xb2c8;&#xb2e4;.

Note



&#xb3c4;&#xd615; &#xc00;&#xc6b4;&#xb370; &#xd558;&#xb098;&#xc00;, &#xc608;&#xb97c; &#xb4e4;&#xc5b4;
a.geom&#xc774; &#xc544;&#xb2c8;&#xb77c; 'SRID=3005;POINT(1011102 450541)::geometry&#xc640;
&#xac19;&#xc774;, (&#xd558;&#xc704; &#xcffc;&#xb9ac;/CTE(common table expression) &#xc548;&#xc5d0;
&#xc788;&#xc9c0; &#xc54a;&#xc740;) &#xc0c1;&#xc218;&#xc77c; &#xacbd;&#xc6b0;&#xc5d0;&#xb9cc;
&#xc778;&#xb371;&#xc2a4;&#xc758; &#xd6a8;&#xacfc;&#xc00; &#xb098;&#xd0c0;&#xb0a9;&#xb2c8;&#xb2e4;.

&#xc2e4;&#xc81c; &#xc0ac;&#xc6a9;&#xb840;&#xb97c; &#xc0b4;&#xd3b4;&#xbcf4;&#xb824;&#xba74; OpenGeo work-
shop: Nearest-Neighbour Searching &#xc744; &#xcc38;&#xc870;&#xd558;&#xc2ed;&#xc2dc;&#xc624;.

&#xac1c;&#xc120; &#xc0ac;&#xd56d;: 2.2.0 &#xbc84;&#xc804; -- PostgreSQL 9.5 &#xc774;&#xc0c1; &#xbc84;&#xc804;&#xc5d0;
&#xb3c4;&#xd615; &#xbc0f;&#xc9c0;&#xb9ac;&#xd615;&#xc5d0; &#xb300;&#xd574; &#xc9c4;&#xc815;&#xd55c; KNN("K
nearest neighbor") &#xc2b5;&#xc131;&#xc744; &#xb3c4;&#xc785;&#xd588;&#xc2b5;&#xb2c8;&#xb2e4;. &#xc9c0;&#xb9ac;&#xc744;
KNN&#xc774; &#xd0c0;&#xc6d0;&#xc6b4;&#xc00; &#xc544;&#xb2c8;&#xb77c; &#xad6c;&#xc6b4;&#xb97c; &#xae30;&#xb9cc;
&#xd55c;&#xb2e4;&#xb294; &#xc810;&#xc5d0; &#xc8fc;&#xc758;&#xd558;&#xc2ed;&#xc2dc;&#xc624;. PostgreSQL 9.4
&#xc774;&#xd558; &#xbc84;&#xc804;&#xc5d0;&#xc11c; &#xc9c0;&#xb9ac;&#xd615;&#xc744; &#xc4f8; &#xc218; &#xc788;&#xc744;
&#xd558;&#xc9c0;&#xb9cc;, &#xc911;&#xc2ec;&#xc810; &#xc0c1;&#xc790;&#xb9cc; &#xc9c0;&#xc6d0;&#xd569;&#xb2c8;&#xb9cc;
&#xacbd; &#xc0ac;&#xd56d;: 2.2.0 &#xbc84;&#xc804; -- PostgreSQL 9.5 &#xbc84;&#xc804; &#xc0ac;&#xc6a9;&#xc790;
&#xacbd;&#xc6b0; &#xc608;&#xc804;&#xc758; &#xd558;&#xc774;&#xbe0c;&#xb9ac;&#xb4dc; &#xbb38;&#xbc95;(Hybrid
syntax)&#xc774; &#xb290;&#xb9b4; &#xc218;&#xb3c4; &#xc788;&#xae30; &#xb54c;&#xbb38;&#xc5d0; PostGIS 2.2 &#xc774;&#xc744;
PostgreSQL 9.5 &#xc774;&#xc0c1; &#xbc84;&#xc804;&#xc5d0;&#xc11c;&#xb9cc; &#xc0ac;&#xc6a9;&#xc790; &#xcf54;&#xb4dc;
&#xc2e4;&#xd589;&#xd558;&#xb294; &#xacbd;&#xc6b0; &#xc774; &#xaf3c;&#xc218;&#xb97c; &#xc81c;&#xc70;&#xd558;&#xc744;
&#xd3b8;&#xc774; &#xc88b;&#xc744; &#xc218;&#xb3c4; &#xc788;&#xc2b5;&#xb2c8;&#xb2e4;. &#xb2e4;&#xc74c; &#xc608;&#xc744;
&#xcc38;&#xc870;&#xd558;&#xc138;&#xc694;.

&#xbcc0;&#xacbd; &#xc0ac;&#xd56d;: 2.2.0 &#xbc84;&#xc804; -- PostgreSQL 9.5 &#xbc84;&#xc804; &#xc0ac;&#xc6a9;&#xc790;
&#xacbd;&#xc6b0; &#xc608;&#xc804;&#xc758; &#xd558;&#xc774;&#xbe0c;&#xb9ac;&#xb4dc; &#xbb38;&#xbc95;(Hybrid
syntax)&#xc774; &#xb290;&#xb9b4; &#xc218;&#xb3c4; &#xc788;&#xae30; &#xb54c;&#xbb38;&#xc5d0; PostGIS 2.2 &#xc774;&#xc744;
PostgreSQL 9.5 &#xc774;&#xc0c1; &#xbc84;&#xc804;&#xc5d0;&#xc11c;&#xb9cc; &#xc0ac;&#xc6a9;&#xc790; &#xcf54;&#xb4dc;
&#xc2e4;&#xd589;&#xd558;&#xb294; &#xacbd;&#xc6b0; &#xc774; &#xaf3c;&#xc218;&#xb97c; &#xc81c;&#xc70;&#xd558;&#xc744;
&#xd3b8;&#xc774; &#xc88b;&#xc744; &#xc218;&#xb3c4; &#xc788;&#xc2b5;&#xb2c8;&#xb2e4;. &#xb2e4;&#xc74c; &#xc608;&#xc744;
&#xcc38;&#xc870;&#xd558;&#xc138;&#xc694;.

2.0.0 &#xbc84;&#xc804;&#xbd80;&#xd130; &#xc0ac;&#xc6a9;&#xd560; &#xc218; &#xc788;&#xc2b5;&#xb2c8;&#xb2e4;.
&#xc57d;&#xd55c; KNN&#xc774; &#xc2e4;&#xc81c; &#xc70;&#xb9ac; &#xb300;&#xc2e0; &#xb3c4;&#xd615; &#xc911;&#xc2ec;&#xc810;

```

SELECT ST_Distance(geom, 'SRID=3005;POINT(1011102 450541)::geometry') as d,edabbr, vaabbr
FROM va2005
ORDER BY d limit 10;

```

| d                | edabbr | vaabbr |
|------------------|--------|--------|
| 0                | ALQ    | 128    |
| 5541.57712511724 | ALQ    | 129A   |
| 5579.67450712005 | ALQ    | 001    |
| 6083.4207708641  | ALQ    | 131    |
| 7691.2205404848  | ALQ    | 003    |
| 7900.75451037313 | ALQ    | 122    |
| 8694.20710669982 | ALQ    | 129B   |
| 9564.24289057111 | ALQ    | 130    |
| 12089.665931705  | ALQ    | 127    |
| 18472.5531479404 | ALQ    | 002    |

(10 rows)

## Distance with KNN

```

SELECT ST_Distance(geom, 'SRID=3005;POINT(1011102 450541)::geometry') as d,edabbr, vaabbr
FROM va2005
ORDER BY d limit 10;

```

| d                | edabbr | vaabbr |
|------------------|--------|--------|
| 0                | ALQ    | 128    |
| 5541.57712511724 | ALQ    | 129A   |
| 5579.67450712005 | ALQ    | 001    |
| 6083.4207708641  | ALQ    | 131    |
| 7691.2205404848  | ALQ    | 003    |
| 7900.75451037313 | ALQ    | 122    |
| 8694.20710669982 | ALQ    | 129B   |
| 9564.24289057111 | ALQ    | 130    |
| 12089.665931705  | ALQ    | 127    |
| 18472.5531479404 | ALQ    | 002    |

(10 rows)

## Distance with KNN and CTE

```

SELECT st_distance(geom, 'SRID=3005;POINT(1011102 450541)::geometry') as d,edabbr, vaabbr
FROM va2005
ORDER BY geom <-> 'SRID=3005;POINT(1011102 450541)::geometry' limit 10;

```

| d                | edabbr | vaabbr |
|------------------|--------|--------|
| 0                | ALQ    | 128    |
| 5541.57712511724 | ALQ    | 129A   |
| 5579.67450712005 | ALQ    | 001    |
| 6083.4207708641  | ALQ    | 131    |
| 7691.2205404848  | ALQ    | 003    |
| 7900.75451037313 | ALQ    | 122    |
| 8694.20710669982 | ALQ    | 129B   |
| 9564.24289057111 | ALQ    | 130    |
| 12089.665931705  | ALQ    | 127    |
| 18472.5531479404 | ALQ    | 002    |

(10 rows)

```

EXPLAIN ANALYZE
SELECT ST_Distance(geom, 'SRID=3005;POINT(1011102 450541)::geometry') as d,edabbr, vaabbr
FROM va2005
ORDER BY geom <-> 'SRID=3005;POINT(1011102 450541)::geometry' limit 10;

```

```

PostgreSQL 9.5
WITH index_query AS (
  SELECT ST_Distance(geom, 'SRID=3005;POINT(1011102 450541)::geometry') as d,edabbr, vaabbr

```

```

WITH index_query AS (
  SELECT ST_Distance(geom, 'SRID=3005;POINT(1011102 450541)::geometry') as d,edabbr, vaabbr

```

```

FROM va2005
ORDER BY geom <-> 'SRID=3005;POINT(1011102 450541)':geometry LIMIT 100)
SELECT *
FROM index_query
ORDER BY d limit 10;

```

| d                | edabbr | vaabbr |
|------------------|--------|--------|
| 0                | ALQ    | 128    |
| 5541.57712511724 | ALQ    | 129A   |
| 5579.67450712005 | ALQ    | 001    |
| 6083.4207708641  | ALQ    | 131    |
| 7691.2205404848  | ALQ    | 003    |
| 7900.75451037313 | ALQ    | 122    |
| 8694.20710669982 | ALQ    | 129B   |
| 9564.24289057111 | ALQ    | 130    |
| 12089.665931705  | ALQ    | 127    |
| 18472.5531479404 | ALQ    | 002    |

(10 rows)

## ST\_DWithin, ST\_Distance, <#>

**ST\_DWithin, ST\_Distance, <#>**

### 8.10.2.2 |=

**|=** — A &#xc640; B &#xc0ac;&#xc774;&#xc758; &#xcd5c;&#xadfc;&#xc811;&#xc810;(closest point of approach)&#xc744;&#xc787;&#xb294; &#xada4;&#xb3c4;(trajectory)&#xc758; &#xac70;&#xb9ac;&#xb97c; &#xbc18;&#xd658;&#xd569;&#xb2c8;&

## Synopsis

double precision |= ( geometry A , geometry B );

## &#xc124;&#xba85;

**|=** &#xc5f0;&#xc0b0;&#xc790;&#xb294; &#xb450; &#xada4;&#xb3c4;(&#xc774;&#xc5f0;&#xc0b0;&#xc790;&#xb294; &#xb3c4;&#xd615;&#xc5d0; &#xb300;&#xd574; &#xc774;&#xc6a9;&#xd560; &#xc218; &#xc788;&#xc744;&#xc9c0;&#xb3c4; &#xbaa8;&#xb974;&#xb294; N&#xc28;&#xc6d0; GiST &#xc778;&#xb371;&#xc2a4;&#xb97c; &#xd65c;&#xc6a9;&#xd560; &#xac83;&#xc785;&#xb2c8;&#xb2e4;. &#xc5f0;&#xc0b0;&#xc790;&#xc00; ORDER BY &#xc808; &#xc548;&#xc5d0; &#xc788;&#xc744; &#xb54c;&#xb9cc; &#xacf5;&#xac04; &#xc778;&#xb371;&#xc2a4;&#xb97c; &#xc4f4;&#xb2e4;&#xb294; &#xc810;&#xc5d0;&#xc11c; &#xacf5;&#xac04; &#xc778;&#xb371;&#xc2a4;&#xb97c; &#xc774;&#xc6a9;&#xd558;&#xb294; &#xb2e4;&#xb978; &#xc5f0;&#xc0b0;&#xc790;&#xb4e4;&#xacfc;&#xb294; &#xb2e4;&#xb985;&#xb2c8;&#xb2e4;.

## Note

&#xc774; &#xd53c;&#xc5f0;&#xc0b0;&#xc790;(operand)&#xb294; &#xb3c4;&#xd615;&#xc5d0; &#xb300;&#xd574; &#xc774;&#xc6a9;&#xd560; &#xc218; &#xc788;&#xc744;&#xc9c0;&#xb3c4; &#xbaa8;&#xb974;&#xb294; N&#xc28;&#xc6d0; GiST &#xc778;&#xb371;&#xc2a4;&#xb97c; &#xd65c;&#xc6a9;&#xd560; &#xac83;&#xc785;&#xb2c8;&#xb2e4;. &#xc5f0;&#xc0b0;&#xc790;&#xc00; ORDER BY &#xc808; &#xc548;&#xc5d0; &#xc788;&#xc744; &#xb54c;&#xb9cc; &#xacf5;&#xac04; &#xc778;&#xb371;&#xc2a4;&#xb97c; &#xc4f4;&#xb2e4;&#xb294; &#xc810;&#xc5d0;&#xc11c; &#xacf5;&#xac04; &#xc778;&#xb371;&#xc2a4;&#xb97c; &#xc774;&#xc6a9;&#xd558;&#xb294; &#xb2e4;&#xb978; &#xc5f0;&#xc0b0;&#xc790;&#xb4e4;&#xacfc;&#xb294; &#xb2e4;&#xb985;&#xb2c8;&#xb2e4;.



**Note**

&#x09; &#xc0ac;&#xc6a9;&#xc790; &#xc9c0;&#xce68;&#xc11c; a.geom&#xc774; &#xc544;&#xb2c8;&#xb77c; 'SRID=3005;LINESTRING(0 0 0,0 0 1)';geometry&#xc640; &#xc19;&#xc774;,, (&#xd558;&#xc704; &#xcffc;&#xb9ac;/CTE(common table expression) &#xc548;&#xc5d0; &#xc788;&#xc9c0; &#xc54a;&#xc740;) &#xc0c1;&#xc218;&#xc77c; &#xacbd;&#xc6b0;&#xc5d0;&#xb9cc; &#xc778;&#xb371;&#xc2a4;&#xc758; &#xd6a8;&#xacfc;&#xc0ac; &#xb098;&#xd0c0;&#xb0a9;&#xb2c8;&#xb2e4;.

2.2.0 &#xbc84;&#xc804;&#xbd80;&#xd130; &#xc0ac;&#xc6a9;&#xd560; &#xc218; &#xc788;&#xc2b5;&#xb2c8;&#xb2e4;. PostgreSQL 9.5 &#xc774;&#xc0c1; &#xbc84;&#xc804;&#xc5d0;&#xc11c;&#xb9cc; &#xc778;&#xb371;&#xc2a4; &#xc9c0;&#xc6d0; supported)&#xc744; &#xc0ac;&#xc6a9;&#xd560; &#xc218; &#xc788;&#xc2b5;&#xb2c8;&#xb2e4;.

**&#xc608;&#xc2dc;**

```
-- &#xae00;&#xc790; &#xadf8;&#xb300;&#xb85c;&#xc758; &#xcffc;&#xb9ac; <-
&#xada4;&#xb3c4;&#xb97c; psql &#xbcc0;&#xc218;&#xb85c; <-
&#xc800;&#xc7a5;&#xd569;&#xb2c8;&#xb2e4;...
\set qt 'ST_AddMeasure(ST_MakeLine(ST_MakePointM(-350,300,0),ST_MakePointM(-410,490,0)) <-
,10,20)'
-- &#xcffc;&#xb9ac;&#xb97c; &#xc2e4;&#xd589;&#xd569;&#xb2c8;&#xb2e4;!
SELECT track_id, dist FROM (
  SELECT track_id, ST_DistanceCPA(tr,:qt) dist
  FROM trajectories
  ORDER BY tr || :qt
  LIMIT 5
) foo;
track_id      dist
-----+-----
    395 | 0.576496831518066
    380 | 5.06797130410151
    390 | 7.72262293958322
    385 | 9.8004461358071
    405 | 10.9534397988433
(5 rows)
```

**&#xcc38;&#xace0;**

**ST\_DistanceCPA, ST\_ClosestPointOfApproach, ST\_IsValidTrajectory**

**8.10.2.3 <#>**

<#> — A&#xc640; B&#xc758; &#xacbd;&#xacc4; &#xc0c1;&#xc790; &#xc0ac;&#xc774;&#xc758; 2&#xcc28;&#xc6d0; &#xc70;&#xb9ac;&#xb97c; &#xbc18;&#xd658;&#xd569;&#xb2c8;&#xb2e4;.

**Synopsis**

double precision <#>( geometry A , geometry B );

**&#xc124;&#xba85;**

<#> &#xc5f0;&#xc0b0;&#xc790;&#xb294; &#xb450; &#xbd80;&#xb3d9;&#xc18c;&#xc218;&#xc810;(floating point) &#xacbd;&#xc0c1;&#xc790; &#xc0ac;&#xc774;&#xc758; &#xc70;&#xb9ac;&#xb97c; &#xbc18;&#xd658;&#xd569;&#xb2c8;&#xb2e4;. &#xacf5;&#xc04; &#xc778;&#xb371;&#xc2a4;(PostgreSQL 9.1 &#xc774;&#xc0c1; &#xbc84;&#xc804; &#xd544;&#xc694;)&#xb8 &#xacbd;&#xacc4; &#xc0c1;&#xc790;&#xb4e4;&#xc744; &#xc77d;&#xc5b4; &#xc62c; &#xc218;&#xb3c4; &#xc788;&#xc2b5;&#xc5d0; &#xadfc;&#xc811; **&#xadfc;&#xc0ac;&#xce58;** &#xc70;&#xb9ac;&#xc758; &#xc21c;&#xc11c;&#xb97c; &#xc815;&#xc790; &#xb370; &#xc720;&#xc6a9;&#xd569;&#xb2c8;&#xb2e4;.

**Note**



```

ORDER BY
  ST_DWithin(geom, ST_GeomFromText('POINT(1 2)')
  geom)
  ST_Distance(b.geom, ST_GeomFromText('LINESTRING(746149 2948672,745954 2948576,
  745787 2948499,745740 2948468,745712 2948438,
  745690 2948384,745677 2948319)',2249)) As b_dist,
  ST_Distance(b.geom, ST_GeomFromText('LINESTRING(746149 2948672,745954
  2948576,
  745787 2948499,745740 2948468,745712 2948438,
  745690 2948384,745677 2948319)',2249)) As act_dist
FROM bos_roads As b
ORDER BY b_dist, b.tlid
LIMIT 100) As foo
ORDER BY act_dist, tlid LIMIT 10;

```

**Note**



```

ORDER BY
  ST_DWithin(geom, ST_GeomFromText('POINT(1 2)')
  geom)
  ST_Distance(b.geom, ST_GeomFromText('LINESTRING(746149 2948672,745954 2948576,
  745787 2948499,745740 2948468,745712 2948438,
  745690 2948384,745677 2948319)',2249)) As b_dist,
  ST_Distance(b.geom, ST_GeomFromText('LINESTRING(746149 2948672,745954
  2948576,
  745787 2948499,745740 2948468,745712 2948438,
  745690 2948384,745677 2948319)',2249)) As act_dist
FROM bos_roads As b
ORDER BY b_dist, b.tlid
LIMIT 100) As foo
ORDER BY act_dist, tlid LIMIT 10;

```

2.0.0  
 PostgreSQL 9.1

**ST\_DWithin, ST\_Distance, <->**

```

SELECT *
FROM (
SELECT b.tlid, b.mtfcc,
      b.geom <#
> ST_GeomFromText('LINESTRING(746149 2948672,745954 2948576,
      745787 2948499,745740 2948468,745712 2948438,
      745690 2948384,745677 2948319)',2249) As b_dist,
      ST_Distance(b.geom, ST_GeomFromText('LINESTRING(746149 2948672,745954
      2948576,
      745787 2948499,745740 2948468,745712 2948438,
      745690 2948384,745677 2948319)',2249)) As act_dist
FROM bos_roads As b
ORDER BY b_dist, b.tlid
LIMIT 100) As foo
ORDER BY act_dist, tlid LIMIT 10;

```

| tlid      | mtfcc | b_dist           | act_dist         |
|-----------|-------|------------------|------------------|
| 85732027  | S1400 | 0                | 0                |
| 85732029  | S1400 | 0                | 0                |
| 85732031  | S1400 | 0                | 0                |
| 85734335  | S1400 | 0                | 0                |
| 85736037  | S1400 | 0                | 0                |
| 624683742 | S1400 | 0                | 128.528874268666 |
| 85719343  | S1400 | 260.839270432962 | 260.839270432962 |
| 85741826  | S1400 | 164.759294123275 | 260.839270432962 |
| 85732032  | S1400 | 277.75           | 311.830282365264 |
| 85735592  | S1400 | 222.25           | 311.830282365264 |

(10 rows)

**ST\_DWithin, ST\_Distance, <->**

ST\_DWithin, ST\_Distance, <->

### 8.10.2.4 <->

<-> — A &#xacbd;&#xacc4; &#xc0c1;&#xc790;&#xc640; B&#xc758; &#xacbd;&#xacc4; &#xc0c1;&#xc790;&#xc758; &#xc911;&#xc2ec;&#xc810; &#xc0ac;&#xc774;&#xc758; N&#xcc28;&#xc6d0; &#xac70;&#xb9ac;&#xb97c; &#xbc18;&#xd658;&

#### Synopsis

double precision <->( geometry A , geometry B );

#### &#xc124;&#xba85;

<-> &#xc5f0;&#xc0b0;&#xc790;&#xb294; &#xb450; &#xb3c4;&#xd615;&#xc758; &#xacbd;&#xacc4; &#xc0c1;&#xc790;&#xc758; &#xc911;&#xc2ec;&#xc810; &#xc0ac;&#xc774;&#xc758; N&#xcc28; (&#xc720;&#xd074;&#xb9ac;&#xb4dc;) &#xac70;&#xb9ac;&#xb97c; &#xbc18;&#xd658;&#xd569;&#xb2c8;&#xb2e4;. &#xcd5c;&#xadfc;&#xc811; &#xadfc;&#xc0ac;&#xc58; &#xac70;&#xb9ac;&#xb97c; &#xc21c;&#xc11c;&#xb97c; &#xc815;&#xb82c;&#xd558;&#xb294; &#xb370; &#xc720;&#xc6a9;&#xd569;&#xb2c8;&#xb2e4;.

#### Note



&#xc774; &#xd53c;&#xc5f0;&#xc0b0;&#xc790;(operand)&#xb294; &#xb3c4;&#xd615;&#xc5d0; &#xb300;&#xd574; &#xc774;&#xc6a9;&#xd560; &#xc218; &#xc788;&#xc744;&#xc9c0;&#xb3c4; &#xbaa8;&#xb974;&#xb294; N&#xcc28;&#xc6d0; GiST &#xc778;&#xb371;&#xc2a4;&#xb97c; &#xd65c;&#xc6a9;&#xd560; &#xac83;&#xc785;&#xb2c8;&#xb2e4;. &#xc5f0;&#xc0b0;&#xc790;&#xac00; ORDER BY &#xc808; &#xc548;&#xc5d0; &#xc788;&#xc744; &#xb54c;&#xb9cc; &#xacf5;&#xac04; &#xc778;&#xb371;&#xc2a4;&#xb97c; &#xc4f4;&#xb2e4;&#xb294; &#xc810;&#xc5d0;&#xc11c; &#xacf5;&#xac04; &#xc778;&#xb371;&#xc2a4;&#xb97c; &#xc774;&#xc6a9;&#xd558;&#xb294; &#xb2e4;&#xb978; &#xc5f0;&#xc0b0;&#xc790;&#xb4e4;&#xacfc;&#xb294; &#xb2e4;&#xb985;&#xb2c8;&#xb2e4;.

#### Note



&#xb3c4;&#xd615; &#xac00;&#xc6b4;&#xb370; &#xd558;&#xb098;&#xac00; &#xc608;&#xb97c; &#xb4e4;&#xc5b4; a.geom&#xc774; &#xc544;&#xb2c8;&#xb77c; 'SRID=3005;POINT(1011102 450541)':geometry&#xc640; &#xac19;&#xc774;; (&#xd558;&#xc704; &#xcffc;&#xb9ac;/CTE(common table expression) &#xc548;&#xc5d0; &#xc788;&#xc9c0; &#xc54a;&#xc740;) &#xc0c1;&#xc218;&#xc77c; &#xacbd;&#xc6b0;&#xc5d0;&#xb9cc; &#xc778;&#xb371;&#xc2a4;&#xc758; &#xd6a8;&#xacfc;&#xac00; &#xb098;&#xd0c0;&#xb0a9;&#xb2c8;&#xb2e4;.

2.2.0 &#xbc84;&#xc804;&#xbd80;&#xd130; &#xc0ac;&#xc6a9;&#xd560; &#xc218; &#xc788;&#xc2b5;&#xb2c8;&#xb2e4; PostgreSQL 9.1 &#xc774;&#xc0c1; &#xbc84;&#xc804;&#xc5d0;&#xc11c;&#xb9cc; KNN&#xc744; &#xc4f8; &#xc218; &#xc788;&

#### &#xcc38;&#xace0;

<<#>>, <->

### 8.10.2.5 <#>

<#> — A&#xc640; B&#xc758; &#xacbd;&#xacc4; &#xc0c1;&#xc790; &#xc0ac;&#xc774;&#xc758; N&#xcc28;&#xc6d0; &#xac70;&#xb9ac;&#xb97c; &#xbc18;&#xd658;&#xd569;&#xb2c8;&#xb2e4;.

#### Synopsis

double precision <<#>>( geometry A , geometry B );

**ST\_3DIntersects**

`<#>` (floating point) `&#xacbd;&#xc0c1;&#xc790;&#xc0ac;&#xc774;&#xc758;&#xac70;&#xb9ac;&#xb97c;&#xbc18;&#xd658;&#xd569;&#xb2c8;&#xb2e4;&#xacf5;&#xac04;&#xc778;&#xb371;&#xc2a4;`(PostgreSQL 9.1 `&#xc774;&#xc0c1;&#xbc84;&#xc804;&#xd544;&#xc694;`)`&#xb2c8;&#xb2e4;&#xacbd;&#xacc4;&#xc0c1;&#xc790;&#xb4e4;&#xc744;&#xc77d;&#xc5b4;&#xc62c;&#xc218;&#xb3c4;&#xc788;&#xc2b5;&#xcd5c;&#xadfc;&#xc811;&#xadfc;&#xc0ac;&#xce58;&#xac70;&#xb9ac;&#xc758;&#xc21c;&#xc11c;&#xb97c;&#xc815;&#xb370;&#xc720;&#xc6a9;&#xd569;&#xb2c8;&#xb2e4;`.

**Note**

`&#xc774;&#xd53c;&#xc5f0;&#xc0b0;&#xc790;`(operand)`&#xb294;&#xb3c4;&#xd615;&#xc5d0;&#xb300;&#xd574;&#xc774;&#xc6a9;&#xd560;&#xc218;&#xc788;&#xc744;&#xc9c0;&#xb3c4;&#xaa8;&#xb974;&#xb294;&#xc5b4;&#xb5a4;&#xc778;&#xb371;&#xc2a4;&#xb77c;&#xb3c4;&#xd65c;&#xc6a9;&#xd560;&#xac83;&#xc785;&#xb2c8;&#xb2e4;`. `&#xc5f0;&#xc0b0;&#xc790;&#xac00;&#xc808;&#xc548;&#xc5d0;&#xc788;&#xc744;&#xb54c;&#xb9cc;&#xacf5;&#xac04;&#xc778;&#xb371;&#xc2a4;&#xb97c;&#xc4f4;&#xb2e4;&#xb294;&#xc810;&#xc5d0;&#xc11c;&#xacf5;&#xac04;&#xc778;&#xb371;&#xc2a4;&#xb97c;&#xc774;&#xc6a9;&#xd558;&#xb294;&#xb2e4;&#xb978;&#xc5f0;&#xc0b0;&#xc790;&#xb4e4;&#xacfc;&#xb294;&#xb2e4;&#xb985;&#xb2c8;&#xb2e4;`.

**Note**

`&#xb3c4;&#xd615;&#xac00;&#xc6b4;&#xb370;&#xd558;&#xb098;&#xac00;&#xc608;&#xb97c;&#xb4e4;&#xc5b4;``g1.geom <<#>&#xac00;&#xc544;&#xb2c8;&#xb77c;&#xc774;&#xc0c1;&#xc218;&#xc77c;&#xacbd;&#xc6b0;&#xc5d0;&#xb9cc;&#xc778;&#xb371;&#xc2a4;&#xc758;&#xd6a8;&#xacfc;&#xac00;&#xb098;&#xd0c0;&#xb0a9;&#xb2c8;&#xb2e4;`.

2.2.0 `&#bc84;&#xc804;&#bd80;&#xd130;&#xc0ac;&#xc6a9;&#xd560;&#xc218;&#xc788;&#xc2b5;&#xb2c8;&#xb2e4;`.

PostgreSQL 9.1 `&#xc774;&#xc0c1;&#bc84;&#xc804;&#xc5d0;&#xc11c;&#xb9cc;&#xc744;&#xc4f8;&#xc218;&#xc788;&`

**ST\_3DIntersects**

`<<->`, `<#>`

## 8.11 Spatial Relationships

### 8.11.1 Topological Relationships

#### 8.11.1.1 ST\_3DIntersects

**ST\_3DIntersects** — Tests if two geometries spatially intersect in 3D - only for points, linestrings, polygons, polyhedral surface (area).

**Synopsis**

boolean **ST\_3DIntersects**( geometry geomA , geometry geomB );

**Description**

Overlaps, Touches, Within all imply spatial intersection. If any of the aforementioned returns true, then the geometries also spatially intersect. Disjoint implies false for spatial intersection.

**Note**

This function automatically includes a bounding box comparison that makes use of any spatial indexes that are available on the geometries.

Changed: 3.0.0 SFCGAL backend removed, GEOS backend supports TINs.

Availability: 2.0.0



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



This function supports Polyhedral surfaces.



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



This method implements the SQL/MM specification. SQL-MM IEC 13249-3: 5.1

**Geometry Examples**

```
SELECT ST_3DIntersects(pt, line), ST_Intersects(pt, line)
   FROM (SELECT 'POINT(0 0 2)::geometry As pt, 'LINESTRING (0 0 1, 0 2 3)::geometry As ←
         line) As foo;
st_3dintersects | st_intersects
-----+-----
f                | t
(1 row)
```

**TIN Examples**

```
SELECT ST_3DIntersects('TIN(((0 0 0,1 0 0,0 1 0,0 0 0)))::geometry, 'POINT(.1 .1 0):: ←
         geometry);
st_3dintersects
-----
t
```

**See Also**

[ST\\_Intersects](#)

**8.11.1.2 ST\_Contains**

**ST\_Contains** — Tests if no points of B lie in the exterior of A, and A and B have at least one interior point in common.

**Synopsis**

boolean **ST\_Contains**(geometry geomA, geometry geomB);

## Description

Returns TRUE if geometry B is completely inside geometry A. A contains B if and only if no points of B lie in the exterior of A, and at least one point of the interior of B lies in the interior of A.

A subtlety of the definition is that a geometry does not contain things in its boundary. Thus polygons and lines do *not* contain lines and points lying in their boundary. For further details see [Subtleties of OGC Covers, Contains, Within](#). (The [ST\\_Covers](#) predicate provides a more inclusive relationship.) However, a geometry does contain itself. (In contrast, in the [ST\\_ContainsProperly](#) predicate a geometry does *not* properly contain itself.)

ST\_Contains is the inverse of [ST\\_Within](#). So,  $ST\_Contains(A, B) = ST\_Within(B, A)$ .



### Note

This function automatically includes a bounding box comparison that makes use of any spatial indexes that are available on the geometries. To avoid index use, use the function `_ST_Contains`.

Performed by the GEOS module

Enhanced: 2.3.0 Enhancement to PIP short-circuit extended to support MultiPoints with few points. Prior versions only supported point in polygon.



### Important

Enhanced: 3.0.0 enabled support for GEOMETRYCOLLECTION



### Important

Do not use this function with invalid geometries. You will get unexpected results.

NOTE: this is the "allowable" version that returns a boolean, not an integer.



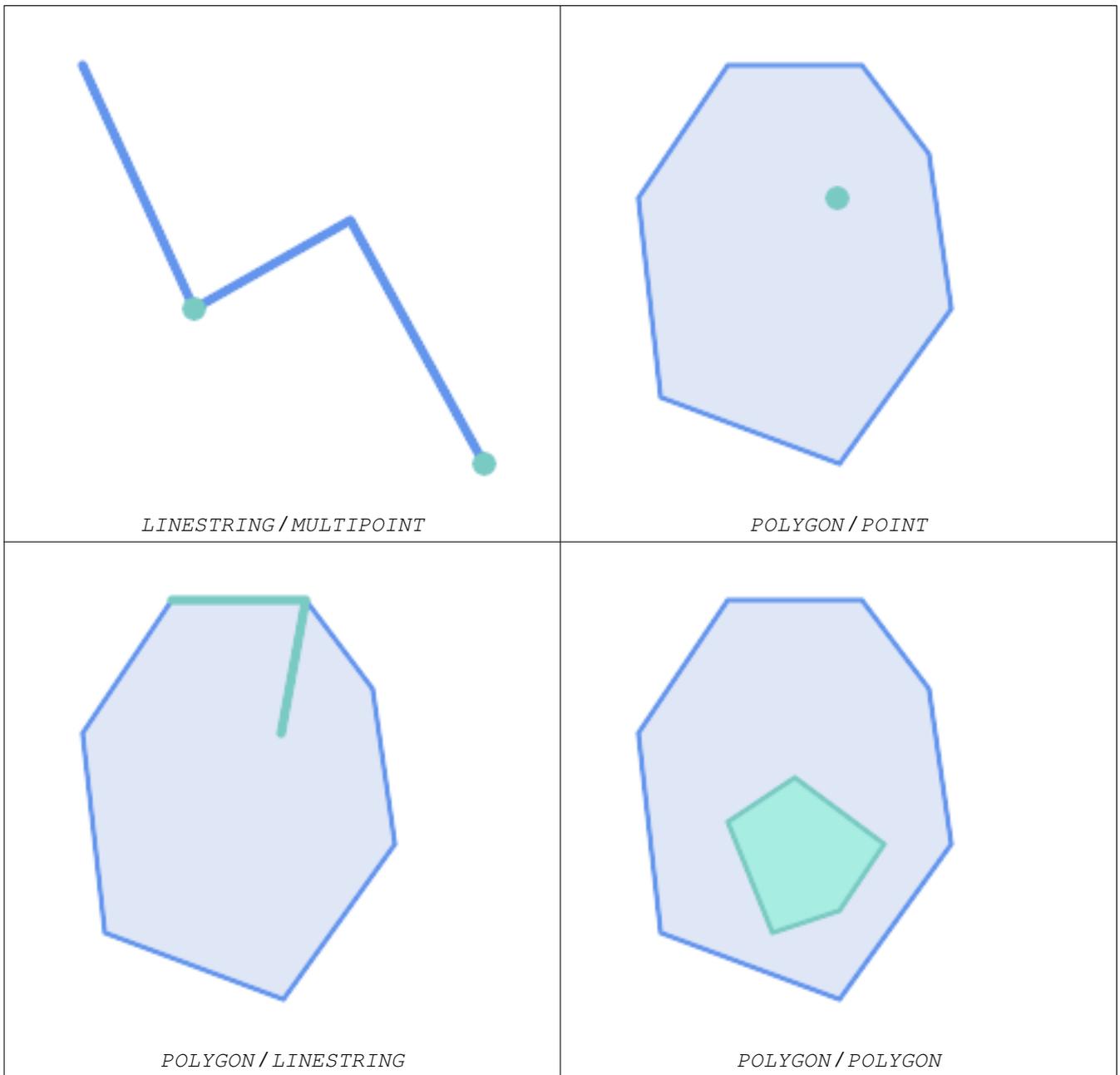
This method implements the [OGC Simple Features Implementation Specification for SQL 1.1](#). s2.1.1.2 // s2.1.13.3 - same as `within(geometry B, geometry A)`



This method implements the SQL/MM specification. SQL-MM 3: 5.1.31

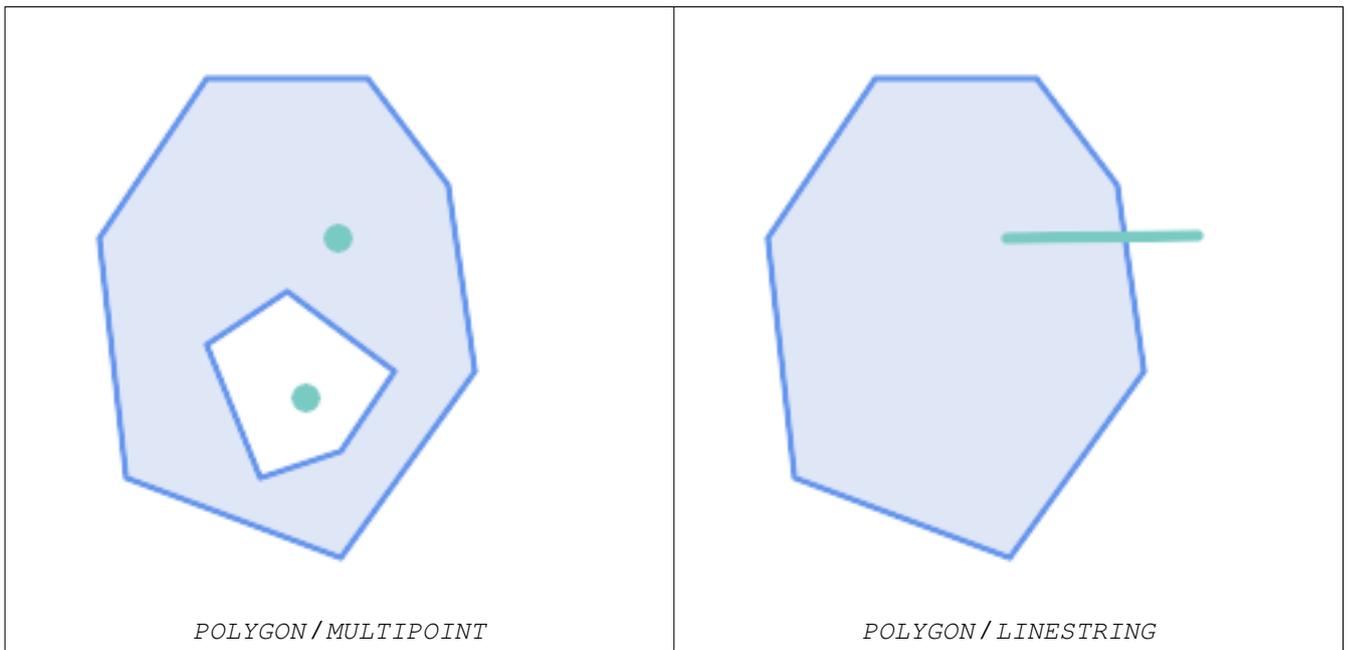
## Examples

ST\_Contains returns TRUE in the following situations:



The ST\_Contains predicate returns FALSE in the following situations:

---



```
-- A circle within a circle
SELECT ST_Contains(smallc, bigc) As smallcontainsbig,
       ST_Contains(bigc,smallc) As bigcontainssmall,
       ST_Contains(bigc, ST_Union(smallc, bigc)) as bigcontainsunion,
       ST_Equals(bigc, ST_Union(smallc, bigc)) as bigisunion,
       ST_Covers(bigc, ST_ExteriorRing(bigc)) As bigcoversexterior,
       ST_Contains(bigc, ST_ExteriorRing(bigc)) As bigcontainsexterior
FROM (SELECT ST_Buffer(ST_GeomFromText('POINT(1 2)'), 10) As smallc,
        ST_Buffer(ST_GeomFromText('POINT(1 2)'), 20) As bigc) As foo;

-- Result
smallcontainsbig | bigcontainssmall | bigcontainsunion | bigisunion | bigcoversexterior | bigcontainsexterior
-----+-----+-----+-----+-----+-----
f                | t                | t                | t          | t                | f

-- Example demonstrating difference between contains and contains properly
SELECT ST_GeometryType(geomA) As geomtype, ST_Contains(geomA,geomA) AS acontainsa,
       ST_ContainsProperly(geomA, geomA) AS acontainspropa,
       ST_Contains(geomA, ST_Boundary(geomA)) As acontainsba, ST_ContainsProperly(geomA,
       ST_Boundary(geomA)) As acontainspropba
FROM (VALUES ( ST_Buffer(ST_Point(1,1), 5,1) ),
            ( ST_MakeLine(ST_Point(1,1), ST_Point(-1,-1) ) ),
            ( ST_Point(1,1) )
        ) As foo(geomA);

geomtype      | acontainsa | acontainspropa | acontainsba | acontainspropba
-----+-----+-----+-----+-----
ST_Polygon    | t          | f              | f           | f
ST_LineString | t          | f              | f           | f
ST_Point      | t          | t              | f           | f
```

## See Also

[ST\\_Boundary](#), [ST\\_ContainsProperly](#), [ST\\_Covers](#), [ST\\_CoveredBy](#), [ST\\_Equals](#), [ST\\_Within](#)

### 8.11.1.3 ST\_ContainsProperly

`ST_ContainsProperly` — Tests if B intersects the interior of A but not the boundary or exterior.

## Synopsis

```
boolean ST_ContainsProperly(geometry geomA, geometry geomB);
```

## Description

Returns true if B intersects the interior of A but not the boundary or exterior.

A does not properly contain itself, but does contain itself.

Every point of the other geometry is a point of this geometry's interior. The DE-9IM Intersection Matrix for the two geometries matches `[T**FF*FF*]` used in [ST\\_Relate](#)

An example use case for this predicate is computing the intersections of a set of geometries with a large polygonal geometry. Since intersection is a fairly slow operation, it can be more efficient to use `containsProperly` to filter out test geometries which lie wholly inside the area. In these cases the intersection is known a priori to be exactly the original test geometry.



#### Note

This function automatically includes a bounding box comparison that makes use of any spatial indexes that are available on the geometries. To avoid index use, use the function `_ST_ContainsProperly`.



#### Note

The advantage of this predicate over [ST\\_Contains](#) and [ST\\_Intersects](#) is that it can be computed more efficiently, with no need to compute topology at individual points.

---

Performed by the GEOS module.

Availability: 1.4.0



#### Important

Enhanced: 3.0.0 enabled support for `GEOMETRYCOLLECTION`



#### Important

Do not use this function with invalid geometries. You will get unexpected results.

---

## Examples

```
--a circle within a circle
SELECT ST_ContainsProperly(smallc, bigc) As smallcontainspropbig,
       ST_ContainsProperly(bigc,smallc) As bigcontainspropsmall,
       ST_ContainsProperly(bigc, ST_Union(smallc, bigc)) as bigcontainspropunion,
       ST_Equals(bigc, ST_Union(smallc, bigc)) as bigisunion,
       ST_Covers(bigc, ST_ExteriorRing(bigc)) As bigcoversexterior,
       ST_ContainsProperly(bigc, ST_ExteriorRing(bigc)) As bigcontainsexterior
FROM (SELECT ST_Buffer(ST_GeomFromText('POINT(1 2)'), 10) As smallc,
       ST_Buffer(ST_GeomFromText('POINT(1 2)'), 20) As bigc) As foo;
--Result
smallcontainspropbig | bigcontainspropsmall | bigcontainspropunion | bigisunion | ↔
bigcoversexterior | bigcontainsexterior
-----+-----+-----+-----+-----
f                | t                | f                | t                | ↔
                | f                |                  |                  |
--example demonstrating difference between contains and contains properly
SELECT ST_GeometryType(geomA) As geomtype, ST_Contains(geomA,geomA) AS acontainsa, ↔
       ST_ContainsProperly(geomA, geomA) AS acontainspropa,
       ST_Contains(geomA, ST_Boundary(geomA)) As acontainsba, ST_ContainsProperly(geomA, ↔
       ST_Boundary(geomA)) As acontainspropba
FROM (VALUES ( ST_Buffer(ST_Point(1,1), 5,1) ),
           ( ST_MakeLine(ST_Point(1,1), ST_Point(-1,-1) ) ),
           ( ST_Point(1,1) )
       ) As foo(geomA);

geomtype      | acontainsa | acontainspropa | acontainsba | acontainspropba
-----+-----+-----+-----+-----
ST_Polygon    | t          | f              | f           | f
ST_LineString | t          | f              | f           | f
ST_Point      | t          | t              | f           | f
```

## See Also

[ST\\_GeometryType](#), [ST\\_Boundary](#), [ST\\_Contains](#), [ST\\_Covers](#), [ST\\_CoveredBy](#), [ST\\_Equals](#), [ST\\_Relate](#), [ST\\_Within](#)

### 8.11.1.4 ST\_CoveredBy

`ST_CoveredBy` — Tests if no point in A is outside B

#### Synopsis

```
boolean ST_CoveredBy(geometry geomA, geometry geomB);
boolean ST_CoveredBy(geography geogA, geography geogB);
```

#### Description

Returns `true` if no point in Geometry/Geography A lies outside Geometry/Geography B. Equivalently, tests if every point of geometry A is inside (i.e. intersects the interior or boundary of) geometry B.



#### Note

This function automatically includes a bounding box comparison that makes use of any spatial indexes that are available on the geometries. To avoid index use, use the function `_ST_CoveredBy`.

**Important**

Enhanced: 3.0.0 enabled support for GEOMETRYCOLLECTION

**Important**

Do not use this function with invalid geometries. You will get unexpected results.

Performed by the GEOS module

Availability: 1.2.2

NOTE: this is the "allowable" version that returns a boolean, not an integer.

Not an OGC standard, but Oracle has it too.

**Examples**

```
--a circle coveredby a circle
SELECT ST_CoveredBy(smallc,smallc) As smallinsmall,
       ST_CoveredBy(smallc, bigc) As smallcoveredbybig,
       ST_CoveredBy(ST_ExteriorRing(bigc), bigc) As exteriorcoveredbybig,
       ST_Within(ST_ExteriorRing(bigc),bigc) As exeriorwithinbig
FROM (SELECT ST_Buffer(ST_GeomFromText('POINT(1 2)'), 10) As smallc,
       ST_Buffer(ST_GeomFromText('POINT(1 2)'), 20) As bigc) As foo;
--Result
smallinsmall | smallcoveredbybig | exteriorcoveredbybig | exeriorwithinbig
-----+-----+-----+-----
t           | t                 | t                 | f
(1 row)
```

**See Also**[ST\\_Contains](#), [ST\\_Covers](#), [ST\\_ExteriorRing](#), [ST\\_Within](#)**8.11.1.5 ST\_Covers**

ST\_Covers — Tests if no point in B is outside A

**Synopsis**

```
boolean ST_Covers(geometry geomA, geometry geomB);
boolean ST_Covers(geography geogpolyA, geography geogpointB);
```

**Description**Returns `true` if no point in Geometry/Geography B is outside Geometry/Geography A. Equivalently, tests if every point of geometry B is inside (i.e. intersects the interior or boundary of) geometry A.**Note**This function automatically includes a bounding box comparison that makes use of any spatial indexes that are available on the geometries. To avoid index use, use the function `_ST_Covers`.

**Important**

Enhanced: 3.0.0 enabled support for GEOMETRYCOLLECTION

**Important**

Do not use this function with invalid geometries. You will get unexpected results.

Performed by the GEOS module

Enhanced: 2.4.0 Support for polygon in polygon and line in polygon added for geography type

Enhanced: 2.3.0 Enhancement to PIP short-circuit for geometry extended to support MultiPoints with few points. Prior versions only supported point in polygon.

Availability: 1.5 - support for geography was introduced.

Availability: 1.2.2

NOTE: this is the "allowable" version that returns a boolean, not an integer.

Not an OGC standard, but Oracle has it too.

**Examples****Geometry example**

```
--a circle covering a circle
SELECT ST_Covers(smallc,smallc) As smallinsmall,
       ST_Covers(smallc, bigc) As smallcoversbig,
       ST_Covers(bigc, ST_ExteriorRing(bigc)) As bigcoversexterior,
       ST_Contains(bigc, ST_ExteriorRing(bigc)) As bigcontainsexterior
FROM (SELECT ST_Buffer(ST_GeomFromText('POINT(1 2)'), 10) As smallc,
       ST_Buffer(ST_GeomFromText('POINT(1 2)'), 20) As bigc) As foo;
--Result
smallinsmall | smallcoversbig | bigcoversexterior | bigcontainsexterior
-----+-----+-----+-----
t            | f              | t                 | f
(1 row)
```

**Geography Example**

```
-- a point with a 300 meter buffer compared to a point, a point and its 10 meter buffer
SELECT ST_Covers(geog_poly, geog_pt) As poly_covers_pt,
       ST_Covers(ST_Buffer(geog_pt,10), geog_pt) As buff_10m_covers_cent
FROM (SELECT ST_Buffer(ST_GeogFromText('SRID=4326;POINT(-99.327 31.4821)'), 300) As ←
       geog_poly,
       ST_GeogFromText('SRID=4326;POINT(-99.33 31.483)') As geog_pt ) As foo;

poly_covers_pt | buff_10m_covers_cent
-----+-----
f              | t
```

**See Also**

[ST\\_Contains](#), [ST\\_CoveredBy](#), [ST\\_Within](#)

### 8.11.1.6 ST\_Crosses

ST\_Crosses — Tests if two geometries have some, but not all, interior points in common.

#### Synopsis

```
boolean ST_Crosses(geometry g1, geometry g2);
```

#### Description

Compares two geometry objects and returns `true` if their intersection "spatially cross", that is, the geometries have some, but not all interior points in common. The intersection of the interiors of the geometries must be non-empty and must have dimension less than the maximum dimension of the two input geometries. Additionally, the intersection of the two geometries must not equal either of the source geometries. Otherwise, it returns `false`.

In mathematical terms, this is:

$$a.Crosses(b) \Leftrightarrow (dim(I(a) \cap I(b)) < max(dim(I(a)), dim(I(b)))) \wedge (a \cap b \neq a) \wedge (a \cap b \neq b)$$

Geometries cross if their DE-9IM Intersection Matrix matches:

- T\*T\*\*\*\*\* for Point/Line, Point/Area, and Line/Area situations
- T\*\*\*\*\*T\*\* for Line/Point, Area/Point, and Area/Line situations
- 0\*\*\*\*\* for Line/Line situations

For Point/Point and Area/Area situations this predicate returns `false`.

The OpenGIS Simple Features Specification defines this predicate only for Point/Line, Point/Area, Line/Line, and Line/Area situations. JTS / GEOS extends the definition to apply to Line/Point, Area/Point and Area/Line situations as well. This makes the relation symmetric.



#### Note

This function automatically includes a bounding box comparison that makes use of any spatial indexes that are available on the geometries.

---



#### Important

Enhanced: 3.0.0 enabled support for GEOMETRYCOLLECTION

---



This method implements the [OGC Simple Features Implementation Specification for SQL 1.1](#). s2.1.13.3

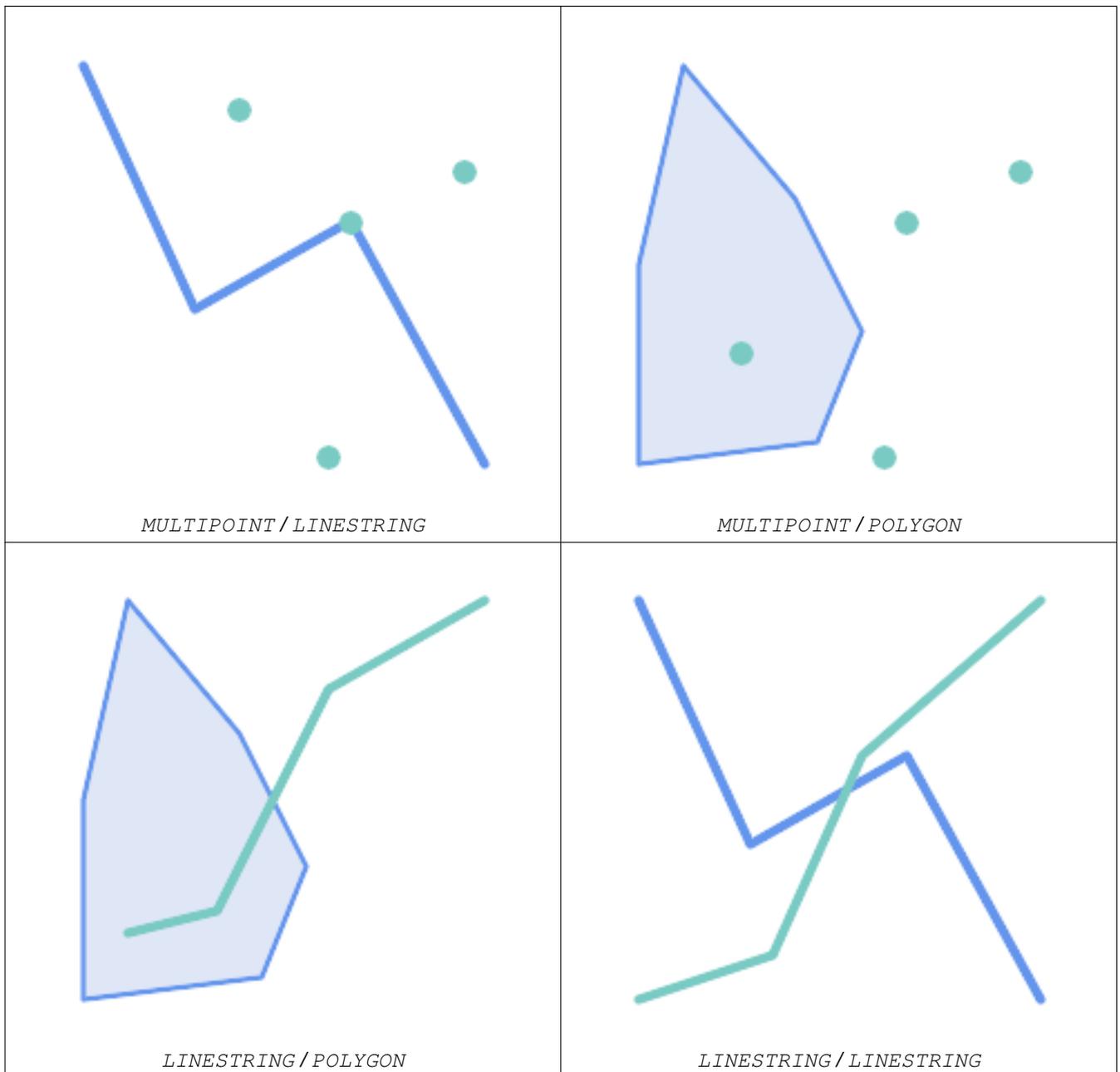


This method implements the SQL/MM specification. SQL-MM 3: 5.1.29

---

## Examples

The following situations all return `true`.



Consider a situation where a user has two tables: a table of roads and a table of highways.

```
CREATE TABLE roads (
  id serial NOT NULL,
  geom geometry,
  CONSTRAINT roads_pkey PRIMARY KEY (
    road_id)
);
```

```
CREATE TABLE highways (
  id serial NOT NULL,
  the_geom geometry,
  CONSTRAINT roads_pkey PRIMARY KEY (
    road_id)
);
```

To determine a list of roads that cross a highway, use a query similar to:

```
SELECT roads.id
FROM roads, highways
WHERE ST_Crosses(roads.geom, highways.geom);
```

### See Also

[ST\\_Contains](#), [ST\\_Overlaps](#)

#### 8.11.1.7 ST\_Disjoint

**ST\_Disjoint** — Tests if two geometries are disjoint (they have no point in common).

### Synopsis

boolean **ST\_Disjoint**( geometry A , geometry B );

### Description

Overlaps, Touches, Within all imply geometries are not spatially disjoint. If any of the aforementioned returns true, then the geometries are not spatially disjoint. Disjoint implies false for spatial intersection.



#### Important

Enhanced: 3.0.0 enabled support for GEOMETRYCOLLECTION

---

Performed by the GEOS module

---



#### Note

This function call does not use indexes

---



#### Note

NOTE: this is the "allowable" version that returns a boolean, not an integer.

---



This method implements the [OGC Simple Features Implementation Specification for SQL 1.1](#). [s2.1.1.2](#) // [s2.1.13.3](#) - [a.Relate\(b, 'FF\\*FF\\*\\*\\*\\*'\)](#)



This method implements the SQL/MM specification. SQL-MM 3: 5.1.26

---

## Examples

```
SELECT ST_Disjoint('POINT(0 0)::geometry, 'LINESTRING ( 2 0, 0 2 ) '::geometry);
st_disjoint
-----
t
(1 row)
SELECT ST_Disjoint('POINT(0 0)::geometry, 'LINESTRING ( 0 0, 0 2 ) '::geometry);
st_disjoint
-----
f
(1 row)
```

## See Also

[ST\\_Intersects](#)

### 8.11.1.8 ST\_Equals

**ST\_Equals** — Tests if two geometries include the same set of points.

## Synopsis

boolean **ST\_Equals**(geometry A, geometry B);

## Description

Returns `true` if the given geometries are "spatially equal". Use this for a 'better' answer than `'='`. Note by spatially equal we mean `ST_Within(A,B) = true` and `ST_Within(B,A) = true` and also mean ordering of points can be different but represent the same geometry structure. To verify the order of points is consistent, use `ST_OrderingEquals` (it must be noted `ST_OrderingEquals` is a little more stringent than simply verifying order of points are the same).



### Important

Enhanced: 3.0.0 enabled support for GEOMETRYCOLLECTION



This method implements the [OGC Simple Features Implementation Specification for SQL 1.1](#). s2.1.1.2



This method implements the SQL/MM specification. SQL-MM 3: 5.1.24

Changed: 2.2.0 Returns true even for invalid geometries if they are binary equal

## Examples

```
SELECT ST_Equals(ST_GeomFromText('LINESTRING(0 0, 10 10)'),
  ST_GeomFromText('LINESTRING(0 0, 5 5, 10 10)'));
st_equals
-----
t
(1 row)

SELECT ST_Equals(ST_Reverse(ST_GeomFromText('LINESTRING(0 0, 10 10)')),
  ST_GeomFromText('LINESTRING(0 0, 5 5, 10 10)'));
```

```

st_equals
-----
t
(1 row)

```

## See Also

[ST\\_IsValid](#), [ST\\_OrderingEquals](#), [ST\\_Reverse](#), [ST\\_Within](#)

### 8.11.1.9 ST\_Intersects

`ST_Intersects` — Tests if two geometries intersect (they have at least one point in common).

## Synopsis

```

boolean ST_Intersects( geometry geomA , geometry geomB );
boolean ST_Intersects( geography geogA , geography geogB );

```

## Description

Compares two geometries and returns `true` if they intersect. Geometries intersect if they have any point in common. For geography, a distance tolerance of 0.00001 meters is used (so points that are very close are considered to intersect). Geometries intersect if their DE-9IM Intersection Matrix matches one of:

- T\*\*\*\*\*
- \*T\*\*\*\*\*
- \*\*\*T\*\*\*\*\*
- \*\*\*\*T\*\*\*\*\*

Spatial intersection is implied by all the other spatial relationship tests, except [ST\\_Disjoint](#), which tests that geometries do NOT intersect.



### Note

This function automatically includes a bounding box comparison that makes use of any spatial indexes that are available on the geometries.

Changed: 3.0.0 SFCGAL version removed and native support for 2D TINs added.

Enhanced: 2.5.0 Supports GEOMETRYCOLLECTION.

Enhanced: 2.3.0 Enhancement to PIP short-circuit extended to support MultiPoints with few points. Prior versions only supported point in polygon.

Performed by the GEOS module (for geometry), geography is native

Availability: 1.5 support for geography was introduced.



### Note

For geography, this function has a distance tolerance of about 0.00001 meters and uses the sphere rather than spheroid calculation.

**Note**

NOTE: this is the "allowable" version that returns a boolean, not an integer.

- ✔ This method implements the [OGC Simple Features Implementation Specification for SQL 1.1](#). s2.1.1.2 //s2.1.13.3 - ST\_Intersects(g1, g2) --> Not (ST\_Disjoint(g1, g2))
- ✔ This method implements the SQL/MM specification. SQL-MM 3: 5.1.27
- ✔ This method supports Circular Strings and Curves
- ✔ This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).

**Geometry Examples**

```
SELECT ST_Intersects('POINT(0 0)::geometry, 'LINESTRING ( 2 0, 0 2 ) '::geometry);
st_intersects
-----
f
(1 row)
SELECT ST_Intersects('POINT(0 0)::geometry, 'LINESTRING ( 0 0, 0 2 ) '::geometry);
st_intersects
-----
t
(1 row)

-- Look up in table. Make sure table has a GiST index on geometry column for faster lookup.
SELECT id, name FROM cities WHERE ST_Intersects(geom, 'SRID=4326;POLYGON((28 53,27.707 ↵
52.293,27 52,26.293 52.293,26 53,26.293 53.707,27 54,27.707 53.707,28 53))');
id | name
---+-----
 2 | Minsk
(1 row)
```

**Geography Examples**

```
SELECT ST_Intersects(
  'SRID=4326;LINESTRING(-43.23456 72.4567,-43.23456 72.4568) '::geography,
  'SRID=4326;POINT(-43.23456 72.4567772) '::geography
);

st_intersects
-----
t
```

**See Also**

[ST\\_3DIntersects](#), [ST\\_Disjoint](#)

**8.11.1.10 ST\_LineCrossingDirection**

ST\_LineCrossingDirection — Returns a number indicating the crossing behavior of two LineStrings.

## Synopsis

integer **ST\_LineCrossingDirection**(geometry linestringA, geometry linestringB);

## Description

Given two linestrings returns an integer between -3 and 3 indicating what kind of crossing behavior exists between them. 0 indicates no crossing. This is only supported for LINESTRINGs.

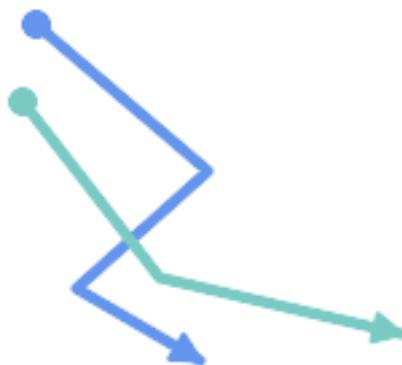
The crossing number has the following meaning:

- 0: LINE NO CROSS
- -1: LINE CROSS LEFT
- 1: LINE CROSS RIGHT
- -2: LINE MULTICROSS END LEFT
- 2: LINE MULTICROSS END RIGHT
- -3: LINE MULTICROSS END SAME FIRST LEFT
- 3: LINE MULTICROSS END SAME FIRST RIGHT

Availability: 1.4

## Examples

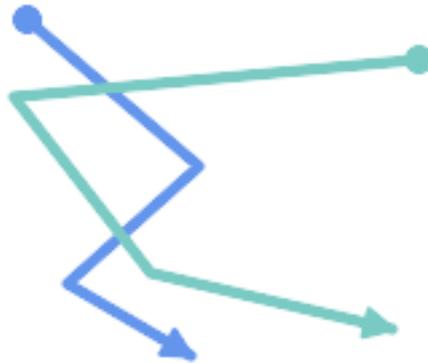
**Example:** LINE CROSS LEFT and LINE CROSS RIGHT



*Blue: Line A; Green: Line B*

```
SELECT ST_LineCrossingDirection(lineA, lineB) As A_cross_B,  
       ST_LineCrossingDirection(lineB, lineA) As B_cross_A  
FROM (SELECT  
  ST_GeomFromText('LINESTRING(25 169,89 114,40 70,86 43)') As lineA,  
  ST_GeomFromText('LINESTRING (20 140, 71 74, 161 53)') As lineB  
  ) As foo;
```

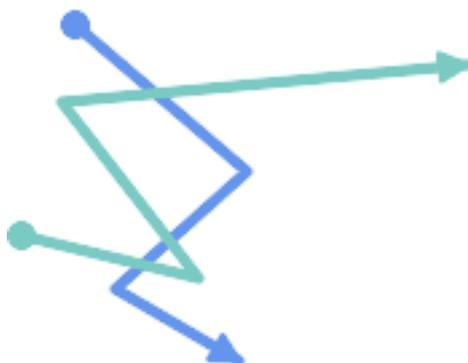
| A_cross_B | B_cross_A |
|-----------|-----------|
| -1        | 1         |

**Example:** LINE MULTICROSS END SAME FIRST LEFT and LINE MULTICROSS END SAME FIRST RIGHT

*Blue: Line A; Green: Line B*

```
SELECT ST_LineCrossingDirection(lineA, lineB) As A_cross_B,
       ST_LineCrossingDirection(lineB, lineA) As B_cross_A
FROM (SELECT
      ST_GeomFromText('LINESTRING(25 169,89 114,40 70,86 43)') As lineA,
      ST_GeomFromText('LINESTRING(171 154,20 140,71 74,161 53)') As lineB
    ) As foo;
```

| A_cross_B | B_cross_A |
|-----------|-----------|
| 3         | -3        |

**Example:** LINE MULTICROSS END LEFT and LINE MULTICROSS END RIGHT

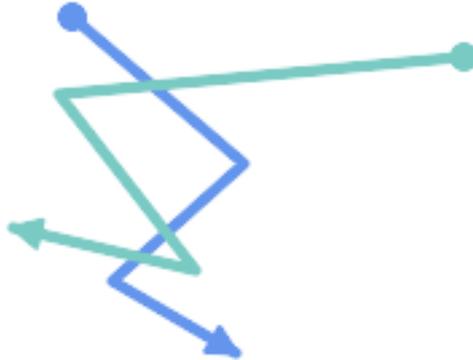
*Blue: Line A; Green: Line B*

```
SELECT ST_LineCrossingDirection(lineA, lineB) As A_cross_B,
       ST_LineCrossingDirection(lineB, lineA) As B_cross_A
FROM (SELECT
```

```
ST_GeomFromText('LINESTRING(25 169,89 114,40 70,86 43)') As lineA,
ST_GeomFromText('LINESTRING(5 90, 71 74, 20 140, 171 154)') As lineB
) As foo;
```

| A_cross_B | B_cross_A |
|-----------|-----------|
| -2        | 2         |

### Example: LINE\_MULTICROSS\_END\_LEFT and LINE\_MULTICROSS\_END\_RIGHT



*Blue: Line A; Green: Line B*

```
SELECT ST_LineCrossingDirection(lineA, lineB) As A_cross_B,
       ST_LineCrossingDirection(lineB, lineA) As B_cross_A
FROM (SELECT
       ST_GeomFromText('LINESTRING(25 169,89 114,40 70,86 43)') As lineA,
       ST_GeomFromText('LINESTRING (171 154, 20 140, 71 74, 2.99 90.16)') As lineB
) As foo;
```

| A_cross_B | B_cross_A |
|-----------|-----------|
| 2         | -2        |

```
SELECT s1.gid, s2.gid, ST_LineCrossingDirection(s1.geom, s2.geom)
FROM streets s1 CROSS JOIN streets s2
ON (s1.gid != s2.gid AND s1.geom && s2.geom )
WHERE ST_LineCrossingDirection(s1.geom, s2.geom) > 0;
```

### See Also

[ST\\_Crosses](#)

#### 8.11.1.11 ST\_OrderingEquals

**ST\_OrderingEquals** — Tests if two geometries represent the same geometry and have points in the same directional order.

### Synopsis

boolean **ST\_OrderingEquals**(geometry A, geometry B);

## Description

`ST_OrderingEquals` compares two geometries and returns t (TRUE) if the geometries are equal and the coordinates are in the same order; otherwise it returns f (FALSE).



### Note

This function is implemented as per the ArcSDE SQL specification rather than SQL-MM. [http://edndoc.esri.com/arcsde/9.1/sql\\_api/sqlapi3.htm#ST\\_OrderingEquals](http://edndoc.esri.com/arcsde/9.1/sql_api/sqlapi3.htm#ST_OrderingEquals)



This method implements the SQL/MM specification. SQL-MM 3: 5.1.43

## Examples

```
SELECT ST_OrderingEquals(ST_GeomFromText('LINESTRING(0 0, 10 10)'),
  ST_GeomFromText('LINESTRING(0 0, 5 5, 10 10)'));
 st_orderingequals
-----
 f
(1 row)

SELECT ST_OrderingEquals(ST_GeomFromText('LINESTRING(0 0, 10 10)'),
  ST_GeomFromText('LINESTRING(0 0, 0 0, 10 10)'));
 st_orderingequals
-----
 t
(1 row)

SELECT ST_OrderingEquals(ST_Reverse(ST_GeomFromText('LINESTRING(0 0, 10 10)'),
  ST_GeomFromText('LINESTRING(0 0, 0 0, 10 10)'));
 st_orderingequals
-----
 f
(1 row)
```

## See Also

[&&](#), [ST\\_Equals](#), [ST\\_Reverse](#)

### 8.11.1.12 ST\_Overlaps

`ST_Overlaps` — Tests if two geometries intersect and have the same dimension, but are not completely contained by each other.

## Synopsis

boolean `ST_Overlaps`(geometry A, geometry B);

## Description

Returns TRUE if geometry A and B "spatially overlap". Two geometries overlap if they have the same dimension, each has at least one point not shared by the other (or equivalently neither covers the other), and the intersection of their interiors has the same dimension. The overlaps relationship is symmetrical.

**Note**

This function automatically includes a bounding box comparison that makes use of any spatial indexes that are available on the geometries. To avoid index use, use the function `_ST_Overlaps`.

Performed by the GEOS module

**Important**

Enhanced: 3.0.0 enabled support for `GEOMETRYCOLLECTION`

NOTE: this is the "allowable" version that returns a boolean, not an integer.



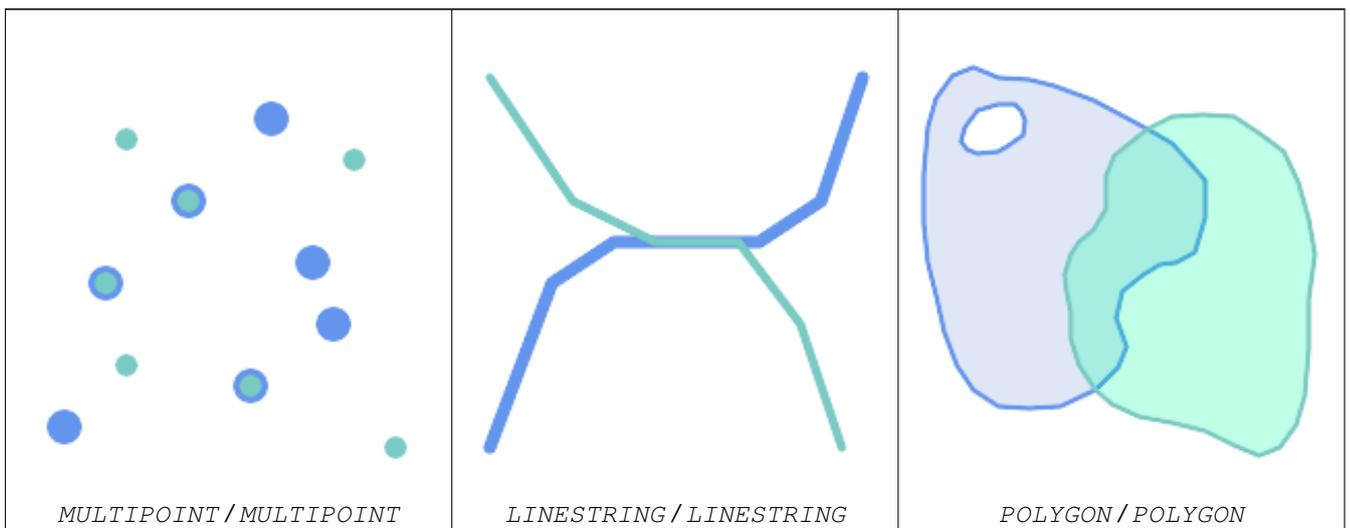
This method implements the [OGC Simple Features Implementation Specification for SQL 1.1](#). s2.1.1.2 // s2.1.13.3

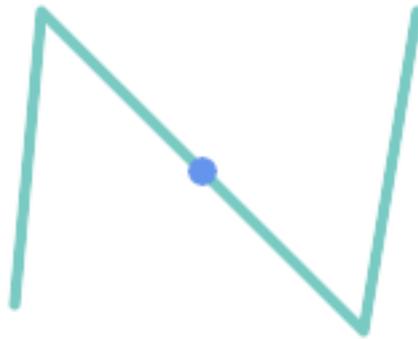


This method implements the SQL/MM specification. SQL-MM 3: 5.1.32

**Examples**

`ST_Overlaps` returns TRUE in the following situations:

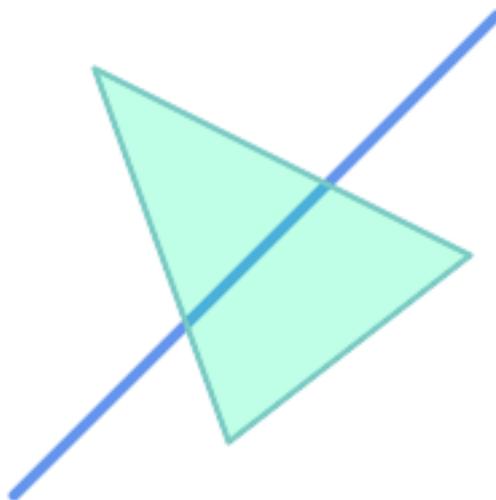




A Point on a LineString is contained, but since it has lower dimension it does not overlap or cross.

```
SELECT ST_Overlaps(a,b) AS overlaps,          ST_Crosses(a,b) AS crosses,
       ST_Intersects(a, b) AS intersects,    ST_Contains(b,a) AS b_contains_a
FROM (SELECT ST_GeomFromText('POINT (100 100)') As a,
       ST_GeomFromText('LINESTRING (30 50, 40 160, 160 40, 180 160)') AS b) AS t
```

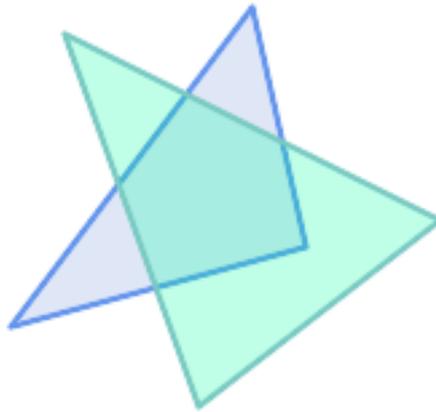
| overlaps | crosses | intersects | b_contains_a |
|----------|---------|------------|--------------|
| f        | f       | t          | t            |



A LineString that partly covers a Polygon intersects and crosses, but does not overlap since it has different dimension.

```
SELECT ST_Overlaps(a,b) AS overlaps,          ST_Crosses(a,b) AS crosses,
       ST_Intersects(a, b) AS intersects,    ST_Contains(a,b) AS contains
FROM (SELECT ST_GeomFromText('POLYGON ((40 170, 90 30, 180 100, 40 170))') AS a,
       ST_GeomFromText('LINESTRING(10 10, 190 190)') AS b) AS t;
```

| overlap | crosses | intersects | contains |
|---------|---------|------------|----------|
| f       | t       | t          | f        |



Two Polygons that intersect but with neither contained by the other overlap, but do not cross because their intersection has the same dimension.

```
SELECT ST_Overlaps(a,b) AS overlaps,      ST_Crosses(a,b) AS crosses,
       ST_Intersects(a, b) AS intersects,  ST_Contains(b, a) AS b_contains_a,
       ST_Dimension(a) AS dim_a, ST_Dimension(b) AS dim_b,
       ST_Dimension(ST_Intersection(a,b)) AS dim_int
FROM (SELECT ST_GeomFromText('POLYGON ((40 170, 90 30, 180 100, 40 170))') AS a,
       ST_GeomFromText('POLYGON ((110 180, 20 60, 130 90, 110 180))') AS b) AS t;
```

| overlaps | crosses | intersects | b_contains_a | dim_a | dim_b | dim_int |
|----------|---------|------------|--------------|-------|-------|---------|
| t        | f       | t          | f            | 2     | 2     | 2       |

## See Also

[ST\\_Contains](#), [ST\\_Crosses](#), [ST\\_Dimension](#), [ST\\_Intersects](#)

### 8.11.1.13 ST\_Relate

**ST\_Relate** — Tests if two geometries have a topological relationship matching an Intersection Matrix pattern, or computes their Intersection Matrix

#### Synopsis

```
boolean ST_Relate(geometry geomA, geometry geomB, text intersectionMatrixPattern);
text ST_Relate(geometry geomA, geometry geomB);
text ST_Relate(geometry geomA, geometry geomB, integer boundaryNodeRule);
```

#### Description

These functions allow testing and evaluating the spatial (topological) relationship between two geometries, as defined by the [Dimensionally Extended 9-Intersection Model](#) (DE-9IM).

The DE-9IM is specified as a 9-element matrix indicating the dimension of the intersections between the Interior, Boundary and Exterior of two geometries. It is represented by a 9-character text string using the symbols 'F', '0', '1', '2' (e.g. 'FF1FF0102').

A specific kind of spatial relationships is evaluated by comparing the intersection matrix to an *intersection matrix pattern*. A pattern can include the additional symbols 'T' and '\*'. Common spatial relationships are provided by the named functions [ST\\_Contains](#), [ST\\_ContainsProperly](#), [ST\\_Covers](#), [ST\\_CoveredBy](#), [ST\\_Crosses](#), [ST\\_Disjoint](#), [ST\\_Equals](#), [ST\\_Intersects](#), [ST\\_Overlaps](#), [ST\\_Touches](#), and [ST\\_Within](#). Using an explicit pattern allows testing multiple conditions of intersects, crosses, etc in one step. It also allows testing spatial relationships which do not have a named spatial relationship function. For example, the relationship "Interior-Intersects" has the DE-9IM pattern T\*\*\*\*\*, which is not evaluated by any named predicate.

For more information refer to Section 5.1.

**Variante 1:** Tests if two geometries are spatially related according to the given `intersectionMatrixPattern`.



#### Note

Unlike most of the named spatial relationship predicates, this does NOT automatically include an index call. The reason is that some relationships are true for geometries which do NOT intersect (e.g. Disjoint). If you are using a relationship pattern that requires intersection, then include the `&&` index call.



#### Note

It is better to use a named relationship function if available, since they automatically use a spatial index where one exists. Also, they may implement performance optimizations which are not available with full relate evaluation.

**Variante 2:** Returns the DE-9IM matrix string for the spatial relationship between the two input geometries. The matrix string can be tested for matching a DE-9IM pattern using [ST\\_RelateMatch](#).

**Variante 3:** Like variant 2, but allows specifying a **Boundary Node Rule**. A boundary node rule allows finer control over whether geometry boundary points are considered to lie in the DE-9IM Interior or Boundary. The `boundaryNodeRule` code is: 1: OGC/MOD2, 2: Endpoint, 3: MultivalentEndpoint, 4: MonovalentEndpoint.

This function is not in the OGC spec, but is implied. see s2.1.13.2



This method implements the [OGC Simple Features Implementation Specification for SQL 1.1](#). s2.1.1.2 // s2.1.13.3



This method implements the SQL/MM specification. SQL-MM 3: 5.1.25

Performed by the GEOS module

Enhanced: 2.0.0 - added support for specifying boundary node rule.



#### Important

Enhanced: 3.0.0 enabled support for GEOMETRYCOLLECTION

## Examples

Using the boolean-valued function to test spatial relationships.

```
SELECT ST_Relate('POINT(1 2)', ST_Buffer( 'POINT(1 2)', 2), '0FFFFF212');
st_relate
-----
t

SELECT ST_Relate(POINT(1 2)', ST_Buffer( 'POINT(1 2)', 2), '*FF*FF212');
st_relate
-----
t
```

Testing a custom spatial relationship pattern as a query condition, with `&&` to enable using a spatial index.

```
-- Find compounds that properly intersect (not just touch) a poly (Interior Intersects)

SELECT c.* , p.name As poly_name
  FROM polys AS p
  INNER JOIN compounds As c
    ON c.geom && p.geom
     AND ST_Relate(p.geom, c.geom, 'T*****');
```

Computing the intersection matrix for spatial relationships.

```
SELECT ST_Relate( 'POINT(1 2)',
                 ST_Buffer( 'POINT(1 2)', 2));

st_relate
-----
0FFFFFF212

SELECT ST_Relate( 'LINESTRING(1 2, 3 4)',
                 'LINESTRING(5 6, 7 8)' );

st_relate
-----
FF1FF0102
```

## See Also

Section 5.1, [ST\\_RelateMatch](#), [ST\\_Contains](#), [ST\\_ContainsProperly](#), [ST\\_Covers](#), [ST\\_CoveredBy](#), [ST\\_Crosses](#), [ST\\_Disjoint](#), [ST\\_Equals](#), [ST\\_Intersects](#), [ST\\_Overlaps](#), [ST\\_Touches](#), [ST\\_Within](#)

### 8.11.1.14 ST\_RelateMatch

`ST_RelateMatch` — Tests if a DE-9IM Intersection Matrix matches an Intersection Matrix pattern

## Synopsis

boolean **ST\_RelateMatch**(text intersectionMatrix, text intersectionMatrixPattern);

## Description

Tests if a [Dimensionally Extended 9-Intersection Model](#) (DE-9IM) `intersectionMatrix` value satisfies an `intersectionMatrixPattern`. Intersection matrix values can be computed by [ST\\_Relate](#).

For more information refer to Section 5.1.

Performed by the GEOS module

Availability: 2.0.0

## Examples

```
SELECT ST_RelateMatch('101202FFF', 'TTTTTFFF') ;
-- result --
t
```

Patterns for common spatial relationships matched against intersection matrix values, for a line in various positions relative to a polygon

```

SELECT pat.name AS relationship, pat.val AS pattern,
       mat.name AS position, mat.val AS matrix,
       ST_RelateMatch(mat.val, pat.val) AS match
FROM (VALUES ( 'Equality', 'T1FF1FFF1' ),
            ( 'Overlaps', 'T*T***T**' ),
            ( 'Within', 'T*F**F***' ),
            ( 'Disjoint', 'FF*FF****' )) AS pat(name,val)
CROSS JOIN
  (VALUES ('non-intersecting', 'FF1FF0212'),
         ('overlapping', '1010F0212'),
         ('inside', '1FF0FF212')) AS mat(name,val);

```

| relationship | pattern   | position         | matrix    | match |
|--------------|-----------|------------------|-----------|-------|
| Equality     | T1FF1FFF1 | non-intersecting | FF1FF0212 | f     |
| Equality     | T1FF1FFF1 | overlapping      | 1010F0212 | f     |
| Equality     | T1FF1FFF1 | inside           | 1FF0FF212 | f     |
| Overlaps     | T*T***T** | non-intersecting | FF1FF0212 | f     |
| Overlaps     | T*T***T** | overlapping      | 1010F0212 | t     |
| Overlaps     | T*T***T** | inside           | 1FF0FF212 | f     |
| Within       | T*F**F*** | non-intersecting | FF1FF0212 | f     |
| Within       | T*F**F*** | overlapping      | 1010F0212 | f     |
| Within       | T*F**F*** | inside           | 1FF0FF212 | t     |
| Disjoint     | FF*FF**** | non-intersecting | FF1FF0212 | t     |
| Disjoint     | FF*FF**** | overlapping      | 1010F0212 | f     |
| Disjoint     | FF*FF**** | inside           | 1FF0FF212 | f     |

## See Also

Section [5.1, ST\\_Relate](#)

### 8.11.1.15 ST\_Touches

`ST_Touches` — Tests if two geometries have at least one point in common, but their interiors do not intersect.

## Synopsis

boolean `ST_Touches`(geometry A, geometry B);

## Description

Returns TRUE if A and B intersect, but their interiors do not intersect. Equivalently, A and B have at least one point in common, and the common points lie in at least one boundary. For Point/Point inputs the relationship is always FALSE, since points do not have a boundary.

In mathematical terms, this relationship is:

$$a.Touches(b) \Leftrightarrow (I(a) \cap I(b) = \emptyset) \wedge (a \cap b) \neq \emptyset$$

This relationship holds if the DE-9IM Intersection Matrix for the two geometries matches one of:

- FT\*\*\*\*\*
- F\*\*T\*\*\*\*\*
- F\*\*\*T\*\*\*\*\*



**Note**

This function automatically includes a bounding box comparison that makes use of any spatial indexes that are available on the geometries. To avoid using an index, use `_ST_Touches` instead.



**Important**

Enhanced: 3.0.0 enabled support for GEOMETRYCOLLECTION



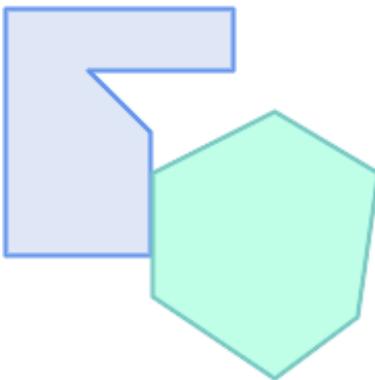
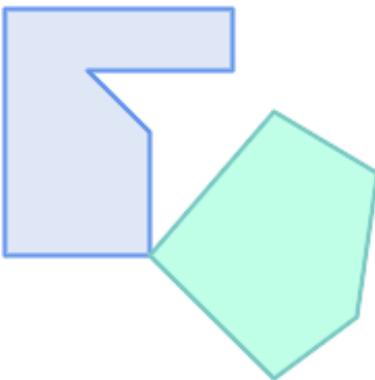
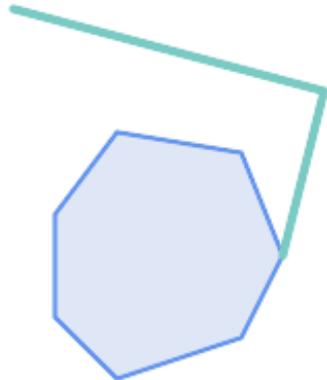
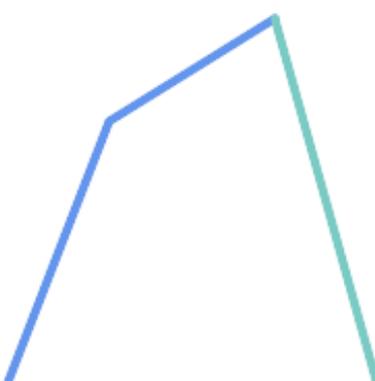
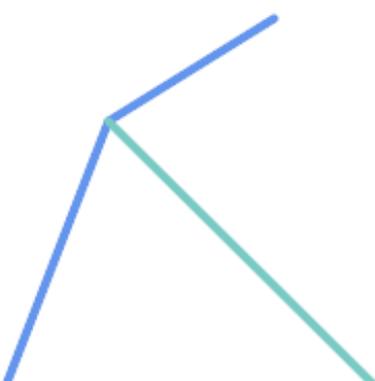
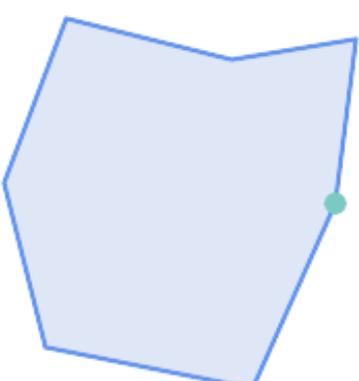
This method implements the [OGC Simple Features Implementation Specification for SQL 1.1](#). s2.1.1.2 // s2.1.13.3



This method implements the SQL/MM specification. SQL-MM 3: 5.1.28

**Examples**

The `ST_Touches` predicate returns `TRUE` in the following examples.

|   |  |   |
|---|--|---|
|  <p><i>POLYGON / POLYGON</i></p>        |  <p><i>POLYGON / POLYGON</i></p>        |  <p><i>POLYGON / LINESTRING</i></p> |
|  <p><i>LINESTRING / LINESTRING</i></p> |  <p><i>LINESTRING / LINESTRING</i></p> |  <p><i>POLYGON / POINT</i></p>     |

```
SELECT ST_Touches('LINESTRING(0 0, 1 1, 0 2)::geometry, 'POINT(1 1)::geometry');
```

```

st_touches
-----
f
(1 row)

SELECT ST_Touches('LINESTRING(0 0, 1 1, 0 2)::geometry, 'POINT(0 2)::geometry');
st_touches
-----
t
(1 row)

```

### 8.11.1.16 ST\_Within

**ST\_Within** — Tests if no points of A lie in the exterior of B, and A and B have at least one interior point in common.

#### Synopsis

boolean **ST\_Within**(geometry A, geometry B);

#### Description

Returns TRUE if geometry A is completely inside geometry B. For this function to make sense, the source geometries must both be of the same coordinate projection, having the same SRID. It is a given that if `ST_Within(A,B)` is true and `ST_Within(B,A)` is true, then the two geometries are considered spatially equal.

A subtlety of this definition is that the boundary of a geometry is not within the geometry. This means that lines and points lying in the boundary of a polygon or line are *not* within the geometry. For further details see [Subtleties of OGC Covers, Contains, Within](#). (The `ST_CoveredBy` predicate provides a more inclusive relationship).

`ST_Within` is the inverse of `ST_Contains`. So, `ST_Within(A, B) = ST_Contains(B, A)`.



#### Note

This function automatically includes a bounding box comparison that makes use of any spatial indexes that are available on the geometries. To avoid index use, use the function `_ST_Within`.

Performed by the GEOS module

Enhanced: 2.3.0 Enhancement to PIP short-circuit for geometry extended to support MultiPoints with few points. Prior versions only supported point in polygon.



#### Important

Enhanced: 3.0.0 enabled support for GEOMETRYCOLLECTION



#### Important

Do not use this function with invalid geometries. You will get unexpected results.

NOTE: this is the "allowable" version that returns a boolean, not an integer.



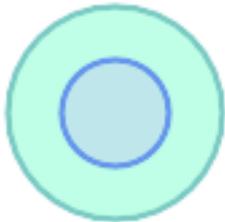
This method implements the [OGC Simple Features Implementation Specification for SQL 1.1](#). s2.1.1.2 // s2.1.13.3 - a.Relate(b, 'T\*\*F\*\*F\*\*\*')



This method implements the SQL/MM specification. SQL-MM 3: 5.1.30

## Examples

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



## See Also

[ST\\_Contains](#), [ST\\_CoveredBy](#), [ST\\_Equals](#), [ST\\_IsValid](#)

## 8.11.2 Distance Relationships

### 8.11.2.1 ST\_3DDWithin

**ST\_3DDWithin** — Tests if two 3D geometries are within a given 3D distance

#### Synopsis

boolean **ST\_3DDWithin**(geometry g1, geometry g2, double precision distance\_of\_srid);

#### Description

Returns true if the 3D distance between two geometry values is no larger than distance `distance_of_srid`. The distance is specified in units defined by the spatial reference system of the geometries. For this function to make sense the source geometries must be in the same coordinate system (have the same SRID).

**Note**

This function automatically includes a bounding box comparison that makes use of any spatial indexes that are available on the geometries.



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



This function supports Polyhedral surfaces.



This method implements the SQL/MM specification. SQL-MM ?

Availability: 2.0.0

**Examples**

```
-- Geometry example - units in meters (SRID: 2163 US National Atlas Equal area) (3D point ←
  and line compared 2D point and line)
-- Note: currently no vertical datum support so Z is not transformed and assumed to be same ←
  units as final.
SELECT ST_3DDWithin(
  ST_Transform(ST_GeomFromEWKT('SRID=4326;POINT(-72.1235 42.3521 4)'),2163),
  ST_Transform(ST_GeomFromEWKT('SRID=4326;LINESTRING(-72.1260 42.45 15, -72.123 42.1546 ←
    20)'),2163),
  126.8
) As within_dist_3d,
ST_DWithin(
  ST_Transform(ST_GeomFromEWKT('SRID=4326;POINT(-72.1235 42.3521 4)'),2163),
  ST_Transform(ST_GeomFromEWKT('SRID=4326;LINESTRING(-72.1260 42.45 15, -72.123 42.1546 ←
    20)'),2163),
  126.8
) As within_dist_2d;

within_dist_3d | within_dist_2d
-----+-----
f              | t
```

**See Also**

[ST\\_3DDFullyWithin](#), [ST\\_DWithin](#), [ST\\_DFullyWithin](#), [ST\\_3DDistance](#), [ST\\_Distance](#), [ST\\_3DMaxDistance](#), [ST\\_Transform](#)

**8.11.2.2 ST\_3DDFullyWithin**

**ST\_3DDFullyWithin** — Tests if two 3D geometries are entirely within a given 3D distance

**Synopsis**

boolean **ST\_3DDFullyWithin**(geometry g1, geometry g2, double precision distance);

**Description**

Returns true if the 3D geometries are fully within the specified distance of one another. The distance is specified in units defined by the spatial reference system of the geometries. For this function to make sense, the source geometries must both be of the same coordinate projection, having the same SRID.

**Note**

This function automatically includes a bounding box comparison that makes use of any spatial indexes that are available on the geometries.

Availability: 2.0.0



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



This function supports Polyhedral surfaces.

**Examples**

```
-- This compares the difference between fully within and distance within as well
-- as the distance fully within for the 2D footprint of the line/point vs. the 3d fully
  within
  SELECT ST_3DDFullyWithin(geom_a, geom_b, 10) as D3DFullyWithin10, ST_3DDWithin(geom_a,
  geom_b, 10) as D3DWithin10,
  ST_DFullyWithin(geom_a, geom_b, 20) as D2DFullyWithin20,
  ST_3DDFullyWithin(geom_a, geom_b, 20) as D3DFullyWithin20 from
  (select ST_GeomFromEWKT('POINT(1 1 2)') as geom_a,
  ST_GeomFromEWKT('LINESTRING(1 5 2, 2 7 20, 1 9 100, 14 12 3)') as geom_b) t1;
d3dfullywithin10 | d3dwithin10 | d2dfullywithin20 | d3dfullywithin20
-----+-----+-----+-----
f                | t          | t                | f
```

**See Also**

[ST\\_3DDWithin](#), [ST\\_DWithin](#), [ST\\_DFullyWithin](#), [ST\\_3DMaxDistance](#)

**8.11.2.3 ST\_DFullyWithin**

**ST\_DFullyWithin** — Tests if two geometries are entirely within a given distance

**Synopsis**

boolean **ST\_DFullyWithin**(geometry g1, geometry g2, double precision distance);

**Description**

Returns true if the geometries are entirely within the specified distance of one another. The distance is specified in units defined by the spatial reference system of the geometries. For this function to make sense, the source geometries must both be of the same coordinate projection, having the same SRID.

**Note**

This function automatically includes a bounding box comparison that makes use of any spatial indexes that are available on the geometries.

Availability: 1.5.0

## Examples

```
postgis=# SELECT ST_DFullyWithin(geom_a, geom_b, 10) as DFullyWithin10, ST_DWithin(geom_a, ←
geom_b, 10) as DWithin10, ST_DFullyWithin(geom_a, geom_b, 20) as DFullyWithin20 from
(select ST_GeomFromText('POINT(1 1)') as geom_a, ST_GeomFromText('LINESTRING(1 5, 2 7, 1 ←
9, 14 12)') as geom_b) t1;
```

```
-----
DFullyWithin10 | DWithin10 | DFullyWithin20 |
-----+-----+-----+
f              | t        | t              |
```

## See Also

[ST\\_MaxDistance](#), [ST\\_DWithin](#), [ST\\_3DDWithin](#), [ST\\_3DDFullyWithin](#)

### 8.11.2.4 ST\_DWithin

**ST\_DWithin** — Tests if two geometries are within a given distance

#### Synopsis

boolean **ST\_DWithin**(geometry g1, geometry g2, double precision distance\_of\_srid);

boolean **ST\_DWithin**(geography gg1, geography gg2, double precision distance\_meters, boolean use\_spheroid = true);

#### Description

Returns true if the geometries are within a given distance

For geometry: The distance is specified in units defined by the spatial reference system of the geometries. For this function to make sense, the source geometries must be in the same coordinate system (have the same SRID).

For geography: units are in meters and distance measurement defaults to `use_spheroid=true`. For faster evaluation use `use_spheroid=false` to measure on the sphere.



#### Note

Use [ST\\_3DDWithin](#) for 3D geometries.



#### Note

This function call includes a bounding box comparison that makes use of any indexes that are available on the geometries.



This method implements the [OGC Simple Features Implementation Specification for SQL 1.1](#).

Availability: 1.5.0 support for geography was introduced

Enhanced: 2.1.0 improved speed for geography. See [Making Geography faster](#) for details.

Enhanced: 2.1.0 support for curved geometries was introduced.

Prior to 1.3, [ST\\_Expand](#) was commonly used in conjunction with `&&` and `ST_Distance` to test for distance, and in pre-1.3.4 this function used that logic. From 1.3.4, `ST_DWithin` uses a faster short-circuit distance function.

## Examples

```
-- Find the nearest hospital to each school
-- that is within 3000 units of the school.
-- We do an ST_DWithin search to utilize indexes to limit our search list
-- that the non-indexable ST_Distance needs to process
-- If the units of the spatial reference is meters then units would be meters
SELECT DISTINCT ON (s.gid) s.gid, s.school_name, s.geom, h.hospital_name
  FROM schools s
  LEFT JOIN hospitals h ON ST_DWithin(s.geom, h.geom, 3000)
  ORDER BY s.gid, ST_Distance(s.geom, h.geom);

-- The schools with no close hospitals
-- Find all schools with no hospital within 3000 units
-- away from the school. Units is in units of spatial ref (e.g. meters, feet, degrees)
SELECT s.gid, s.school_name
  FROM schools s
  LEFT JOIN hospitals h ON ST_DWithin(s.geom, h.geom, 3000)
  WHERE h.gid IS NULL;

-- Find broadcasting towers that receiver with limited range can receive.
-- Data is geometry in Spherical Mercator (SRID=3857), ranges are approximate.

-- Create geometry index that will check proximity limit of user to tower
CREATE INDEX ON broadcasting_towers using gist (geom);

-- Create geometry index that will check proximity limit of tower to user
CREATE INDEX ON broadcasting_towers using gist (ST_Expand(geom, sending_range));

-- Query towers that 4-kilometer receiver in Minsk Hackerspace can get
-- Note: two conditions, because shorter LEAST(b.sending_range, 4000) will not use index.
SELECT b.tower_id, b.geom
  FROM broadcasting_towers b
  WHERE ST_DWithin(b.geom, 'SRID=3857;POINT(3072163.4 7159374.1)', 4000)
  AND ST_DWithin(b.geom, 'SRID=3857;POINT(3072163.4 7159374.1)', b.sending_range);
```

## See Also

[ST\\_Distance](#), [ST\\_3DDWithin](#)

### 8.11.2.5 ST\_PointInsideCircle

**ST\_PointInsideCircle** — Tests if a point geometry is inside a circle defined by a center and radius.

#### Synopsis

boolean **ST\_PointInsideCircle**(geometry a\_point, float center\_x, float center\_y, float radius);

#### Description

Returns true if the geometry is a point and is inside the circle with center `center_x`, `center_y` and radius `radius`.



#### Warning

Does not use spatial indexes. Use [ST\\_DWithin](#) instead.



**Note**



(2.5 \* ST\_Area(geom) / ST\_Area(geom) \* 0.3048 ^ 2) \* ST\_Area(geom) \* 0.3048 ^ 2  
 (non-zero) XY (EPSG:2249) EPSG:2249  
 (non-zero) XY (EPSG:2249) EPSG:2249  
 (non-zero) XY (EPSG:2249) EPSG:2249

**Area Conversion**

ST\_Area(geom) sqft, ST\_Area(geom) \* 0.3048 ^ 2 sqm  
 ST\_Area(geom) sqft, ST\_Area(geom) \* 0.3048 ^ 2 sqm  
 ST\_Area(geom) sqft, ST\_Area(geom) \* 0.3048 ^ 2 sqm

```

select ST_Area(geom) sqft,
       ST_Area(geom) * 0.3048 ^ 2 sqm
from (
    select 'SRID=2249;POLYGON((743238 2967416,743238 2967450,
                               743265 2967450,743265.625 2967416,743238 2967416))' :: geometry geom
) subquery;
sqft sqm
928.625 86.27208552
928.625 86.27208552
928.625 86.27208552
    
```

ST\_Area(ST\_Transform(geom, 26986)) As sqm  
 ST\_Area(ST\_Transform(geom, 26986)) As sqm  
 ST\_Area(ST\_Transform(geom, 26986)) As sqm

```

select ST_Area(geom) sqft,
       ST_Area(ST_Transform(geom, 26986)) As sqm
from (
    select
        'SRID=2249;POLYGON((743238 2967416,743238 2967450,
                             743265 2967450,743265.625 2967416,743238 2967416))' :: geometry geom
) subquery;
sqft sqm
928.625 86.272430607008
928.625 86.272430607008
928.625 86.272430607008
    
```

ST\_Area(geom) sqft, ST\_Area(geom) \* 0.3048 ^ 2 sqm  
 ST\_Area(geom) sqft, ST\_Area(geom) \* 0.3048 ^ 2 sqm  
 ST\_Area(geom) sqft, ST\_Area(geom) \* 0.3048 ^ 2 sqm

WGS84 4326 (SRID=2249;POLYGON((743238 2967416,743238 2967450,743265 2967450,743265.625 2967416,743238 2967416)))::geometry, 4326)::geography geog  
 ST\_Area(geog) / 0.3048 ^ 2 sqft\_spheroid,  
 ST\_Area(geog, false) / 0.3048 ^ 2 sqft\_sphere,  
 ST\_Area(geog) sqm\_spheroid  
 from (

```
select ST_Transform(
  'SRID=2249;POLYGON((743238 2967416,743238 2967450,743265 2967450,743265.625 2967416,743238 2967416)))::geometry,
  4326
) :: geography geog
) as subquery;
sqft_spheroid sqft_sphere sqm_spheroid
928.684405784452 927.049336105925 86.2776044979692
928.684405784452 927.049336105925 86.2776044979692
```

If your data is in geography already:

```
select ST_Area(geog) / 0.3048 ^ 2 sqft,
  ST_Area(the_geog) sqm
from somegeogtable;
```

**ST\_3DArea, ST\_GeomFromEWKT, ST\_LengthSpheroid, ST\_Perimeter, ST\_Transform**

## 8.12.2 ST\_Azimuth

**ST\_Azimuth** — Returns the azimuth in radians of the target point from the origin point, or NULL if the two points are coincident. The azimuth angle is a positive clockwise angle referenced from the positive Y axis (geometry) or the North meridian (geography): North = 0; Northeast =  $\pi/4$ ; East =  $\pi/2$ ; Southeast =  $3\pi/4$ ; South =  $\pi$ ; Southwest =  $5\pi/4$ ; West =  $3\pi/2$ ; Northwest =  $7\pi/4$ .

### Synopsis

```
float ST_Azimuth(geometry origin, geometry target);
float ST_Azimuth(geography origin, geography target);
```

**ST\_Azimuth**

Returns the azimuth in radians of the target point from the origin point, or NULL if the two points are coincident. The azimuth angle is a positive clockwise angle referenced from the positive Y axis (geometry) or the North meridian (geography): North = 0; Northeast =  $\pi/4$ ; East =  $\pi/2$ ; Southeast =  $3\pi/4$ ; South =  $\pi$ ; Southwest =  $5\pi/4$ ; West =  $3\pi/2$ ; Northwest =  $7\pi/4$ .

For the geography type, the azimuth solution is known as the [inverse geodesic problem](#).

The azimuth is a mathematical concept defined as the angle between a reference vector and a point, with angular units in radians. The result value in radians can be converted to degrees using the PostgreSQL function `degrees()`.

**ST\_Azimuth** (geometry origin, geometry target);  
**ST\_Azimuth** (geography origin, geography target);





## 8.12.4 ST\_ClosestPoint

`ST_ClosestPoint` — Returns the 2D point on `g1` that is closest to `g2`. This is the first point of the shortest line from one geometry to the other.

### Synopsis

geometry `ST_ClosestPoint`(geometry `geom1`, geometry `geom2`);

### Notes

Returns the 2-dimensional point on `geom1` that is closest to `geom2`. This is the first point of the shortest line between the geometries (as computed by `ST_ShortestLine`).

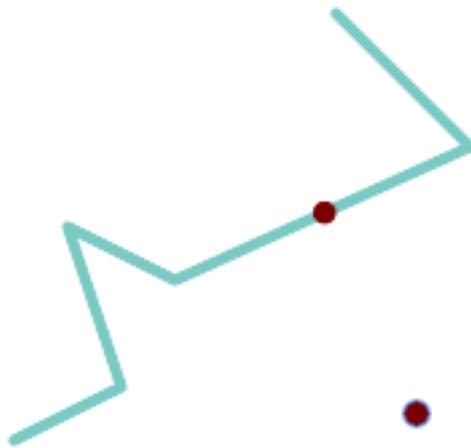


#### Note

`ST_ClosestPoint` is a 2D function. For 3D geometries, use `ST_3DClosestPoint`.

1.5.0 `ST_ClosestPoint` returns a 2D point. For 3D geometries, use `ST_3DClosestPoint`.

### Examples



*The closest point for a Point and a LineString is the point itself. The closest point for a LineString and a Point is a point on the line.*

```
SELECT ST_AsText( ST_ClosestPoint(pt,line)) AS cp_pt_line,
       ST_AsText( ST_ClosestPoint(line,pt)) AS cp_line_pt
FROM (SELECT 'POINT (160 40)::geometry AS pt,
            'LINESTRING (10 30, 50 50, 30 110, 70 90, 180 140, 130 190)::geometry AS line ) AS t;
```

| cp_pt_line    | cp_line_pt                                   |
|---------------|--|
| POINT(160 40) | POINT(125.75342465753425 115.34246575342466) |



```

0) 2D
&#xd3ec;&#xc778;&#xd2b8;&#xb97c; &#xbc18;&#xd658;&#xd569;&#xb2c8;&#b2e4;. 2D &#bc0f; 3D&#xc758; &#xacbd;&#xc
&#xb354; &#xc774;&#xc0c1; Z&#xac00; &#xc5c6;&#xc744; &#xb54c; Z&#xb97c; 0&#xc73c;&#xb85c; &#xac00;&#xc815;&#xd55
&#xc54a;&#xc2b5;&#xb2c8;&#xb2e4;.

```

### ST\_ClosestPoint

|  |
|--|
| <pre> &amp;#xb77c;&amp;#xc778;&amp;#xc2a4;&amp;#xd2b8;&amp;#xb9c1;&amp;#xacfc; &amp;#xd3ec;&amp;#xc778;&amp;#xd2b8; -- 3D, 2D &amp;#xaa8;&amp;#xb450;&amp;#xc758; &amp;#xcd5c;&amp;#xadfc;&amp;#xc811;&amp;#xc810;(closest point)  SELECT ST_AsEWKT(ST_3DClosestPoint(line,pt)) AS cp3d_line_pt,        ST_AsEWKT(ST_ClosestPoint(line,pt)) As cp2d_line_pt FROM (SELECT 'POINT(100 100 30)'::geometry As pt,             'LINESTRING (20 80 20, 98 190 1, 110 180 3, 50 75 1000)'):: geometry As line       ) As foo;  cp3d_line_pt                     cp2d_line_pt -----+----- POINT(54.6993798867619 128.935022917228 11.5475869506606)   POINT(73.0769230769231 115.384615384615) </pre>                                      |
| <pre> &amp;#xb77c;&amp;#xc778;&amp;#xc2a4;&amp;#xd2b8;&amp;#xb9c1;&amp;#xacfc; &amp;#xba40;&amp;#xd2f0;&amp;#xd3ec;&amp;#xc778;&amp;#xd2b8; -- 3D, 2D &amp;#xaa8;&amp;#xb450;&amp;#xc758; &amp;#xcd5c;&amp;#xadfc;&amp;#xc811;&amp;#xc810;(closest point)  SELECT ST_AsEWKT(ST_3DClosestPoint(line,pt)) AS cp3d_line_pt,        ST_AsEWKT(ST_ClosestPoint(line,pt)) As cp2d_line_pt FROM (SELECT 'MULTIPOINT(100 100 30, 50 74 1000)')::geometry As pt,             'LINESTRING (20 80 20, 98 190 1, 110 180 3, 50 75 900)'):: geometry As line       ) As foo;  cp3d_line_pt            cp2d_line_pt -----+----- POINT(54.6993798867619 128.935022917228 11.5475869506606)   POINT(50 75) </pre>                                  |
| <pre> &amp;#xba40;&amp;#xd2f0;&amp;#xb77c;&amp;#xc778;&amp;#xc2a4;&amp;#xd2b8;&amp;#xb9c1;&amp;#xacfc; &amp;#xd3f4;&amp;#xb9ac;&amp;#xace4; -- 3D, 2D &amp;#xaa8;&amp;#xb450;&amp;#xc758; &amp;#xcd5c;&amp;#xadfc;&amp;#xc811;&amp;#xc810;(closest point)  SELECT ST_AsEWKT(ST_3DClosestPoint(poly, mline)) As cp3d,        ST_AsEWKT(ST_ClosestPoint(poly, mline)) As cp2d FROM (SELECT ST_GeomFromEWKT('POLYGON((175 150 5, 20 40 5, 35 45 5, 50 60 5, 100 100 5, 175 150 5))') As poly,        ST_GeomFromEWKT('MULTILINESTRING((175 155 2, 20 40 20, 50 60 -2, 125 100 1, 175 155 1), (1 10 2, 5 20 1))') As mline ) As foo;  cp3d            cp2d -----+----- POINT(39.993580415989 54.1889925532825 5)   POINT(20 40) </pre> |

### ST\_3DShortestLine

ST\_AsEWKT, ST\_ClosestPoint, ST\_3DDistance, ST\_3DShortestLine

## 8.12.6 ST\_Distance

ST\_Distance — (longest)

### Synopsis

```
float ST_HausdorffDistance(geometry g1, geometry g2);
float ST_HausdorffDistance(geometry g1, geometry g2, float densifyFrac);
```

### Parameters

`g1`, `g2`: Geometry objects. `densifyFrac`: A float value representing the fraction of the distance between vertices that should be used to create new vertices. A value of 0.5 means that a new vertex will be created at the midpoint of each line segment. A value of 1.0 means that a new vertex will be created at every point along the line segment. A value of 0.0 means that no new vertices will be created.

For **geography** types defaults to return the minimum geodesic distance between two geographies in meters, compute on the spheroid determined by the SRID. If `use_spheroid` is false, a faster spherical calculation is used.



This method implements the [OGC Simple Features Implementation Specification for SQL 1.1](#).



This method implements the SQL/MM specification. SQL-MM 3: 5.1.23



This method supports Circular Strings and Curves

1.5.0 &#xc84;&#xc804;&#xbd80;&#xd130; &#xc9c0;&#xb9ac;&#xd615;&#xc744; &#xc9c0;&#xc6d0;&#xd569;&#xb2c8;&#xb2e4;&#xb300;&#xc6a9;&#xb7c9; &#xb610;&#xb294; &#xc218;&#xb9ce;&#xc740; &#xaf2d;&#xc9d3;&#xc810;&#xc744; &#xac00;&#xb3c4;&#xd615;&#xc744; &#xb354; &#xc798; &#xc98;&#xb9ac;&#xd558;&#xae30; &#xc704;&#xd574; &#xd3c9;&#xba74;&#xb300;&#xd55c; &#xc18d;&#xb3c4;&#xb97c; &#xd5a5;&#xc0c1;&#xc2dc;&#xcf30;&#xc2b5;&#xb2c8;&#xb2e4;.

&#xac1c;&#xc120; &#xc0ac;&#xd56d;: 2.1.0 &#xc84;&#xc804;&#xbd80;&#xd130; &#xc9c0;&#xb9ac;&#xd615;&#xc5d0; &#xb300;&#xd55c; &#xc18d;&#xb3c4;&#xac00; &#xd5a5;&#xc0c1;&#xb410;&#xc2b5;&#xb2c8;&#xb2e4;. &#xc790;&#xc138;&#xb0b4;&#xc6a9;&#xc740; **Making Geography faster** &#xb97c; &#xc38;&#xc870;&#xd558;&#xc2ed;&#xc2dc;&#xc624;.

&#xac1c;&#xc120; &#xc0ac;&#xd56d;: 2.1.0 &#xc84;&#xc804;&#xbd80;&#xd130; &#xb9cc;&#xace1; &#xb3c4;&#xd615;&#xc704; &#xc9c0;&#xc6d0;&#xd558;&#xae30; &#xc2dc;&#xc791;&#xd588;&#xc2b5;&#xb2c8;&#xb2e4;.

&#xac1c;&#xc120; &#xc0ac;&#xd56d;: 2.2.0 &#xc84;&#xc804;&#xbd80;&#xd130; &#xd68c;&#xc804;&#xd0c0;&#xc6d0;&#xc6c0;&#xc21;&#xc815;&#xc2dc; &#xc815;&#xd655;&#xb3c4;&#xc640; &#xac15;&#xb825;&#xd568;&#xc744; &#xd5a5;&#xc0c1;&#xc704;&#xd574; GeographicLib&#xc744; &#xc774;&#xc6a9;&#xd569;&#xb2c8;&#xb2e4;. &#xc774; &#xc0c8; &#xae30;&#xb2c8;&#xc7a5;&#xc810;&#xc744; &#xcde8;&#xd558;&#xb824;&#xba74; Proj 4.9.0 &#xc774;&#xc0c1; &#xc84;&#xc804;&#xc774; &#xd544;&#xc694;&#xd569;&#xb2c8;&#xb2e4;.

Changed: 3.0.0 - does not depend on SFCGAL anymore.

### Examples

Geometry example - units in planar degrees 4326 is WGS 84 long lat, units are degrees.

```
SELECT ST_Distance (
  'SRID=4326;POINT(-72.1235 42.3521) '::geometry,
  'SRID=4326;LINESTRING(-72.1260 42.45, -72.123 42.1546) '::geometry );
-----
0.00150567726382282
```

Geometry example - units in meters (SRID: 3857, proportional to pixels on popular web maps). Although the value is off, nearby ones can be compared correctly, which makes it a good choice for algorithms like KNN or KMeans.

```
SELECT ST_Distance(
  ST_Transform('SRID=4326;POINT(-72.1235 42.3521)::geometry, 3857),
  ST_Transform('SRID=4326;LINESTRING(-72.1260 42.45, -72.123 42.1546)::geometry, 3857) ) ←
  ;
-----
167.441410065196
```

Geometry example - units in meters (SRID: 3857 as above, but corrected by cos(lat) to account for distortion)

```
SELECT ST_Distance(
  ST_Transform('SRID=4326;POINT(-72.1235 42.3521)::geometry, 3857),
  ST_Transform('SRID=4326;LINESTRING(-72.1260 42.45, -72.123 42.1546)::geometry, 3857)
) * cosd(42.3521);
-----
123.742351254151
```

Geometry example - units in meters (SRID: 26986 Massachusetts state plane meters) (most accurate for Massachusetts)

```
SELECT ST_Distance(
  ST_Transform('SRID=4326;POINT(-72.1235 42.3521)::geometry, 26986),
  ST_Transform('SRID=4326;LINESTRING(-72.1260 42.45, -72.123 42.1546)::geometry, 26986) ←
  );
-----
123.797937878454
```

Geometry example - units in meters (SRID: 2163 US National Atlas Equal area) (least accurate)

```
SELECT ST_Distance(
  ST_Transform('SRID=4326;POINT(-72.1235 42.3521)::geometry, 2163),
  ST_Transform('SRID=4326;LINESTRING(-72.1260 42.45, -72.123 42.1546)::geometry, 2163) ) ←
  ;
-----
126.664256056812
```

**ST\_DistanceSphere**

Same as geometry example but note units in meters - use sphere for slightly faster and less accurate computation.

```
SELECT ST_Distance(gg1, gg2) As spheroid_dist, ST_Distance(gg1, gg2, false) As sphere_dist
FROM (SELECT
  'SRID=4326;POINT(-72.1235 42.3521)::geography as gg1,
  'SRID=4326;LINESTRING(-72.1260 42.45, -72.123 42.1546)::geography as gg2
  ) As foo ;

spheroid_dist | sphere_dist
-----+-----
123.802076746848 | 123.475736916397
```

**ST\_3DDistance**

[ST\\_3DDistance](#), [ST\\_DWithin](#), [ST\\_DistanceSphere](#), [ST\\_DistanceSpheroid](#), [ST\\_MaxDistance](#), [ST\\_HausdorffDistance](#), [ST\\_FrechetDistance](#), [ST\\_Transform](#)

## 8.12.7 ST\_3DDistance

**ST\_3DDistance** — Returns the 3D distance between two geometry objects. The distance is calculated using the 3D coordinates of the geometry objects. The distance is returned in the units of the geometry objects. The distance is calculated using the 3D coordinates of the geometry objects. The distance is returned in the units of the geometry objects.

## Synopsis

```
float ST_3DDistance(geometry g1, geometry g2);
```

### Notes

This function supports 3D and will not drop the z-index. This function supports Polyhedral surfaces. This method implements the SQL/MM specification. SQL-MM ISO/IEC 13249-3



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



This function supports Polyhedral surfaces.



This method implements the SQL/MM specification. SQL-MM ISO/IEC 13249-3

2.0.0 and 2.1.0: 2D and 3D. 2.2.0: 2D and 3D. 3.0.0: 2D and 3D.

2.0.0 and 2.1.0: 2D and 3D. 2.2.0: 2D and 3D. 3.0.0: 2D and 3D. 3.0.0: 2D and 3D.

Changed: 3.0.0 - SFCGAL version removed

### Examples

```
-- ST_3DDistance(geom1, geom2)
-- ST_3DDistance(ST_GeomFromEWKT('SRID=2163;POINT(-72.1235 42.3521 4)'),
--               ST_GeomFromEWKT('SRID=2163;LINESTRING(-72.1260 42.45 15, -72.123 42.1546 20)'),2163)
-- ST_Distance(ST_Transform(ST_GeomFromText('POINT(-72.1235 42.3521)',4326),2163),
--             ST_Transform(ST_GeomFromText('LINESTRING(-72.1260 42.45, -72.123 42.1546)',4326),2163))
--             As dist_3d,
--             ST_Distance(
--               ST_Transform(ST_GeomFromText('POINT(-72.1235 42.3521)',4326),2163),
--               ST_Transform(ST_GeomFromText('LINESTRING(-72.1260 42.45, -72.123 42.1546)',4326),2163))
--             As dist_2d;
--
-- dist_3d | dist_2d
-----+-----
127.295059324629 | 126.66425605671
```

```
-- ST_3DDistance(poly, mline)
-- ST_3DDistance(ST_GeomFromEWKT('POLYGON((175 150 5, 20 40 5, 35 45 5, 50 60 5, 100 100 5, 175 150 5))'),
--               ST_GeomFromEWKT('SRID=2163;LINESTRING(-72.1260 42.45 15, -72.123 42.1546 20)'),2163)
--             As dist3d,
--             ST_Distance(poly, mline) As dist2d
--             FROM (SELECT ST_GeomFromEWKT('POLYGON((175 150 5, 20 40 5, 35 45 5, 50 60 5, 100 100 5, 175 150 5))') As poly,
```

```

        ST_GeomFromEWKT('MULTILINESTRING((175 155 2, 20 40 20, 50 60 -2, 125 100 1, ←
            175 155 1),
            (1 10 2, 5 20 1))') As mline ) As foo;
    dist3d      | dist2d
-----+-----
0.716635696066337 |      0

```

**ST\_Distance**

[ST\\_Distance](#), [ST\\_3DClosestPoint](#), [ST\\_3DDWithin](#), [ST\\_3DMaxDistance](#), [ST\\_3DShortestLine](#), [ST\\_Transform](#)

## 8.12.8 ST\_DistanceSphere

**ST\_DistanceSphere** — Returns the spherical distance between two **geometry** objects. The distance is calculated using the Haversine formula. The radius of the Earth is used by default. The SRID of the geometry objects is used to determine the units of the distance. The distance is returned in the same units as the SRID. The SRID of the geometry objects is used to determine the units of the distance. The distance is returned in the same units as the SRID.

### Synopsis

```
float ST_DistanceSphere(geometry geom1lonlatA, geometry geom1lonlatB, float8 radius=6371008);
```

**ST\_DistanceSphere**

**ST\_DistanceSphere**(*geom1lonlatA*, *geom1lonlatB*, *radius*)

*geom1lonlatA* and *geom1lonlatB* are **geometry** objects. *radius* is a float8 value representing the radius of the Earth in meters. The SRID of the geometry objects is used to determine the units of the distance. The distance is returned in the same units as the SRID.

**ST\_DistanceSphere** returns the spherical distance between two **geometry** objects. The distance is calculated using the Haversine formula. The radius of the Earth is used by default. The SRID of the geometry objects is used to determine the units of the distance. The distance is returned in the same units as the SRID.

**ST\_DistanceSphere** returns the spherical distance between two **geometry** objects. The distance is calculated using the Haversine formula. The radius of the Earth is used by default. The SRID of the geometry objects is used to determine the units of the distance. The distance is returned in the same units as the SRID.

**ST\_DistanceSphere**

```

SELECT round(CAST(ST_DistanceSphere(ST_Centroid(the_geom), ST_GeomFromText('POINT(-118 38) ←
    ',4326)) As numeric),2) As dist_meters,
round(CAST(ST_Distance(ST_Transform(ST_Centroid(the_geom), 32611),
    ST_Transform(ST_GeomFromText('POINT(-118 38)', 4326), 32611)) As numeric),2) ←
    As dist_utm11_meters,
round(CAST(ST_Distance(ST_Centroid(the_geom), ST_GeomFromText('POINT(-118 38)', 4326)) As ←
    numeric),5) As dist_degrees,
round(CAST(ST_Distance(ST_Transform(the_geom, 32611),
    ST_Transform(ST_GeomFromText('POINT(-118 38)', 4326), 32611)) As numeric),2) ←
    As min_dist_line_point_meters
FROM
    (SELECT ST_GeomFromText('LINESTRING(-118.584 38.374,-118.583 38.5)', 4326) As ←
        the_geom) as foo;

```

| dist_meters | dist_utm11_meters | dist_degrees | min_dist_line_point_meters |
|-------------|-------------------|--------------|----------------------------|
| 70424.47    | 70438.00          | 0.72900      | 65871.18                   |

**ST\_Distance**

**ST\_DistanceSpheroid**

## 8.12.9 ST\_DistanceSpheroid

**ST\_DistanceSpheroid** — Returns the distance between two geometries in spherical Earth coordinates, assuming a spheroid model of the Earth. The distance is calculated using the Vincenty formula. The SRID of the input geometries is used to determine the spheroid model. The SRID of the output geometry is the same as the SRID of the input geometries. The SRID of the output geometry is the same as the SRID of the input geometries. The SRID of the output geometry is the same as the SRID of the input geometries.

### Synopsis

```
float ST_DistanceSpheroid(geometry geomlonlatA, geometry geomlonlatB, spheroid measurement_spheroid=WGS84);
```

**ST\_LengthSpheroid**

**ST\_LengthSpheroid** — Returns the length of a geometry in spherical Earth coordinates, assuming a spheroid model of the Earth. The length is calculated using the Vincenty formula. The SRID of the input geometry is used to determine the spheroid model. The SRID of the output geometry is the same as the SRID of the input geometry. The SRID of the output geometry is the same as the SRID of the input geometry. The SRID of the output geometry is the same as the SRID of the input geometry.

### Note



The SRID of the input geometries is used to determine the spheroid model. The SRID of the output geometry is the same as the SRID of the input geometries. The SRID of the output geometry is the same as the SRID of the input geometries. The SRID of the output geometry is the same as the SRID of the input geometries.

1.5 The SRID of the input geometries is used to determine the spheroid model. The SRID of the output geometry is the same as the SRID of the input geometries. The SRID of the output geometry is the same as the SRID of the input geometries. The SRID of the output geometry is the same as the SRID of the input geometries.

2.2.0 The SRID of the input geometries is used to determine the spheroid model. The SRID of the output geometry is the same as the SRID of the input geometries. The SRID of the output geometry is the same as the SRID of the input geometries. The SRID of the output geometry is the same as the SRID of the input geometries.

**ST\_DistanceSphere**

```
SELECT round(CAST(
  ST_DistanceSpheroid(ST_Centroid(the_geom), ST_GeomFromText('POINT(-118 38) ←
    ', 4326), 'SPHEROID["WGS 84", 6378137, 298.257223563]')
  As numeric), 2) As dist_meters_spheroid,
  round(CAST(ST_DistanceSphere(ST_Centroid(the_geom), ST_GeomFromText('POINT ←
    (-118 38)', 4326)) As numeric), 2) As dist_meters_sphere,
```

```

round(CAST(ST_Distance(ST_Transform(ST_Centroid(the_geom), 32611),
                        ST_Transform(ST_GeomFromText('POINT(-118 38)', 4326), 32611)) As numeric), 2) ←
      As dist_utm11_meters
FROM
  (SELECT ST_GeomFromText('LINESTRING(-118.584 38.374,-118.583 38.5)', 4326) As ←
    the_geom) as foo;
dist_meters_spheroid | dist_meters_sphere | dist_utm11_meters
-----+-----+-----
              70454.92 |              70424.47 |              70438.00

```

**ST\_Distance**

[ST\\_Distance](#), [ST\\_DistanceSphere](#)

### 8.12.10 ST\_FrechetDistance

**ST\_FrechetDistance** — Computes the Fréchet distance between two geometries. The Fréchet distance is a measure of similarity between curves that takes into account the location and ordering of the points along the curves. Therefore it is often better than the Hausdorff distance.

#### Synopsis

```
float ST_FrechetDistance(geometry g1, geometry g2, float densifyFrac = -1);
```

**ST\_FrechetDistance**

Implements algorithm for computing the Fréchet distance restricted to discrete points for both geometries, based on [Computing Discrete Fréchet Distance](#). The Fréchet distance is a measure of similarity between curves that takes into account the location and ordering of the points along the curves. Therefore it is often better than the Hausdorff distance.

When the optional densifyFrac is specified, this function performs a segment densification before computing the discrete Fréchet distance. The densifyFrac parameter sets the fraction by which to densify each segment. Each segment will be split into a number of equal-length subsegments, whose fraction of the total length is closest to the given fraction.

Units are in the units of the spatial reference system of the geometries.



#### Note

ST\_FrechetDistance(geometry g1, geometry g2, float densifyFrac = -1);



#### Note

The smaller densifyFrac we specify, the more accurate Fréchet distance we get. But, the computation time and the memory usage increase with the square of the number of subsegments.

GEOS [ST\\_FrechetDistance](#); requires GEOS >= 3.7.0

Availability: 2.4.0 - requires GEOS >= 3.7.0

**ST\_FrechetDistance**

```
postgres=# SELECT st_frechetdistance('LINESTRING (0 0, 100 0)::geometry, 'LINESTRING (0 0, ↵
          50 50, 100 0)::geometry');
 st_frechetdistance
-----
          70.7106781186548
(1 row)
```

```
SELECT st_frechetdistance('LINESTRING (0 0, 100 0)::geometry, 'LINESTRING (0 0, 50 50, 100 ↵
          0)::geometry, 0.5);
 st_frechetdistance
-----
                    50
(1 row)
```

**ST\_HausdorffDistance****ST\_HausdorffDistance****8.12.11 ST\_HausdorffDistance**

**ST\_HausdorffDistance** — Returns the Hausdorff distance between two geometries. The Hausdorff distance is a measure of how similar or dissimilar 2 geometries are.

**Synopsis**

```
float ST_HausdorffDistance(geometry g1, geometry g2);
float ST_HausdorffDistance(geometry g1, geometry g2, float densifyFrac);
```

**ST\_HausdorffDistance**

Returns the **Hausdorff distance** between two geometries. The Hausdorff distance is a measure of how similar or dissimilar 2 geometries are.

The function actually computes the "Discrete Hausdorff Distance". This is the Hausdorff distance computed at discrete points on the geometries. The *densifyFrac* parameter can be specified, to provide a more accurate answer by densifying segments before computing the discrete Hausdorff distance. Each segment is split into a number of equal-length subsegments whose fraction of the segment length is closest to the given fraction.

Units are in the units of the spatial reference system of the geometries.

**Note**

This algorithm is NOT equivalent to the standard Hausdorff distance. However, it computes an approximation that is correct for a large subset of useful cases. One important case is Linestrings that are roughly parallel to each other, and roughly equal in length. This is a useful metric for line matching.

1.5.0 Returns the Hausdorff distance between two geometries. The Hausdorff distance is a measure of how similar or dissimilar 2 geometries are.

**ST\_HausdorffDistance**



*Hausdorff distance (red) and distance (yellow) between two lines*

```
SELECT ST_HausdorffDistance(geomA, geomB),
       ST_Distance(geomA, geomB)
FROM (SELECT 'LINESTRING (20 70, 70 60, 110 70, 170 70)::geometry AS geomA,
            'LINESTRING (20 90, 130 90, 60 100, 190 100)::geometry AS geomB) AS t;
st_hausdorffdistance | st_distance
-----+-----
37.26206567625497 |          20
```

**Example:** Hausdorff distance with densification.

```
SELECT ST_HausdorffDistance(
    'LINESTRING (130 0, 0 0, 0 150)::geometry,
    'LINESTRING (10 10, 10 150, 130 10)::geometry,
    0.5);
-----
70
```

**Example:** For each building, find the parcel that best represents it. First we require that the parcel intersect with the building geometry. `DISTINCT ON` guarantees we get each building listed only once. `ORDER BY .. ST_HausdorffDistance` selects the parcel that is most similar to the building.

```
SELECT DISTINCT ON (buildings.gid) buildings.gid, parcels.parcel_id
FROM buildings
INNER JOIN parcels
ON ST_Intersects(buildings.geom, parcels.geom)
ORDER BY buildings.gid, ST_HausdorffDistance(buildings.geom, parcels.geom);
```

**ST\_FrechetDistance**

**ST\_FrechetDistance**

### 8.12.12 ST\_Length

`ST_Length` — Returns the length of a geometry in the SRID units.



**ST\_Length**

ST\_Length — Returns the length of a 2D line segment in the specified SRID. The length is calculated using the Pythagorean theorem. The length is returned in the same units as the input geometry. The length is calculated using the Pythagorean theorem. The length is returned in the same units as the input geometry.

```
SELECT ST_Length(ST_GeomFromText('LINESTRING(743238 2967416,743238 2967450,743265 2967450,743265.625 2967416,743238 2967416)',2249));
st_length
-----
122.630744000095

-- WGS84 ST_Length(ST_GeomFromText('LINESTRING(743238 2967416,743238 2967450,743265 2967450,743265.625 2967416,743238 2967416)',2249),4326);
st_length
-----
34309.4563576191
```

**ST\_LengthSpheroid**

ST\_LengthSpheroid — Returns the length of a 2D line segment in the specified SRID, using a spheroid model for the Earth. The length is calculated using the Pythagorean theorem. The length is returned in the same units as the input geometry.

```
-- ST_LengthSpheroid(ST_GeomFromText('LINESTRING(743238 2967416,743238 2967450,743265 2967450,743265.625 2967416,743238 2967416)',2249),4326);
st_length_spheroid
-----+-----
34310.5703627288 | 34346.2060960742
```

**ST\_Length2D**

[ST\\_GeographyFromText](#), [ST\\_GeomFromEWKT](#), [ST\\_LengthSpheroid](#), [ST\\_Perimeter](#), [ST\\_Transform](#)

**8.12.13 ST\_Length2D**

ST\_Length2D — Returns the length of a 2D line segment in the specified SRID, using a spheroid model for the Earth. The length is calculated using the Pythagorean theorem. The length is returned in the same units as the input geometry.

**Synopsis**

```
float ST_Length2D(geometry a_2dlinestring);
```

**ST\_3DLength**

ST\_3DLength — Returns the length of a 3D line string in units of the SRID. If the SRID is not specified, the length is returned in the units of the geometry's SRID. The length is calculated using the 3D distance between the start and end points of the line string.

**Parameters**

*geom*, *SRID*

**8.12.14 ST\_3DLength**

ST\_3DLength — Returns the length of a 3D line string in units of the SRID.

**Synopsis**

```
float ST_3DLength(geometry a_3dlinestring);
```

**Examples**

```
SELECT ST_3DLength(ST_GeomFromText('LINESTRING(743238 2967416 1,743238 2967450 1,743265
2967450 3,743265.625 2967416 3,743238 2967416 3)',2249));
ST_3DLength
-----
122.704716741457
```



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



This method implements the SQL/MM specification. SQL-MM IEC 13249-3: 7.1, 10.3

ST\_3DLength — Returns the length of a 3D line string in units of the SRID.

**ST\_LengthSpheroid**

ST\_LengthSpheroid — Returns the length of a 2D line string in units of the SRID, using a spheroid model for the Earth. The length is calculated using the 2D distance between the start and end points of the line string.

```
SELECT ST_LengthSpheroid(ST_GeomFromText('LINESTRING(743238 2967416 1,743238 2967450 1,743265
2967450 3,743265.625 2967416 3,743238 2967416 3)',2249));
ST_LengthSpheroid
-----
122.704716741457
```

**Parameters**

*geom*, *SRID*

**8.12.15 ST\_LengthSpheroid**

ST\_LengthSpheroid — Returns the length of a 2D line string in units of the SRID, using a spheroid model for the Earth.

## Synopsis

```
float ST_LengthSpheroid(geometry a_geometry, spheroid a_spheroid);
```

### Parameters

`a_geometry` is a geometry value.  
`a_spheroid` is a spheroid value, either a text name or a numeric pair of semi-major axis and inverse flattening.

```
SPHEROID [<NAME>, <SEMI-MAJOR AXIS>, <INVERSE FLATTENING>]
```

```
SPHEROID ["GRS_1980", 6378137, 298.257222101]
```

```
SPHEROID ["GRS_1980", 6378137, 298.257222101]
```

1.2.2 `ST_LengthSpheroid` is a 2D function. 2.2.0 `ST_LengthSpheroid` is a 3D function.

`ST_LengthSpheroid` is a 2D function. 2.2.0 `ST_LengthSpheroid` is a 3D function. `ST_LengthSpheroid` is a 2D function. 2.2.0 `ST_LengthSpheroid` is a 3D function.



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

### Examples

```
SELECT ST_LengthSpheroid( geometry_column,
                          'SPHEROID["GRS_1980",6378137,298.257222101]' )
FROM geometry_table;

SELECT ST_LengthSpheroid( the_geom, sph_m ) As tot_len,
ST_LengthSpheroid(ST_GeometryN(the_geom,1), sph_m) As len_line1,
ST_LengthSpheroid(ST_GeometryN(the_geom,2), sph_m) As len_line2
FROM (SELECT ST_GeomFromText('MULTILINESTRING((-118.584 38.374, -118.583 38.5),
(-71.05957 42.3589 , -71.061 43))') As the_geom,
CAST('SPHEROID["GRS_1980",6378137,298.257222101]' As spheroid) As sph_m) as foo;
  tot_len      | len_line1 | len_line2
-----+-----+-----
 85204.5207562955 | 13986.8725229309 | 71217.6482333646

-- 3D
SELECT ST_LengthSpheroid( the_geom, sph_m ) As tot_len,
ST_LengthSpheroid(ST_GeometryN(the_geom,1), sph_m) As len_line1,
ST_LengthSpheroid(ST_GeometryN(the_geom,2), sph_m) As len_line2
FROM (SELECT ST_GeomFromEWKT('MULTILINESTRING((-118.584 38.374 20, -118.583 38.5 30),
(-71.05957 42.3589 75, -71.061 43 90))') As the_geom,
CAST('SPHEROID["GRS_1980",6378137,298.257222101]' As spheroid) As sph_m) as foo;
  tot_len      | len_line1 | len_line2
-----+-----+-----
 85204.5259107402 | 13986.876097711 | 71217.6498130292
```

**ST\_LongestLine**

[ST\\_GeometryN](#), [ST\\_Length](#)

### 8.12.16 ST\_LongestLine

**ST\_LongestLine** — Returns the 2-dimensional longest line between the points of two geometries. The line returned starts on  $g_1$  and ends on  $g_2$ . The longest line always occurs between two vertices. The function returns the first longest line if more than one is found. The length of the line is equal to the distance returned by [ST\\_MaxDistance](#).

#### Synopsis

geometry **ST\_LongestLine**(geometry  $g_1$ , geometry  $g_2$ );

**Returns**

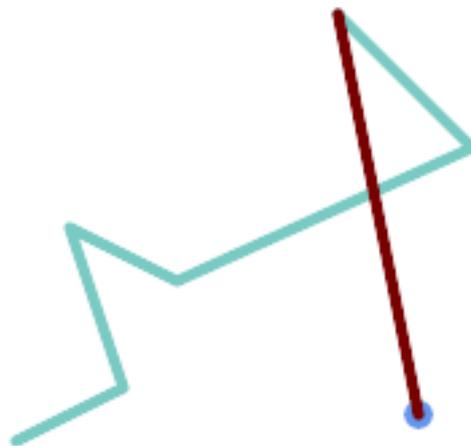
Returns the 2-dimensional longest line between the points of two geometries. The line returned starts on  $g_1$  and ends on  $g_2$ .

The longest line always occurs between two vertices. The function returns the first longest line if more than one is found. The length of the line is equal to the distance returned by [ST\\_MaxDistance](#).

If  $g_1$  and  $g_2$  are the same geometry, returns the line between the two vertices farthest apart in the geometry. This is a diameter of the circle computed by [ST\\_MinimumBoundingCircle](#)

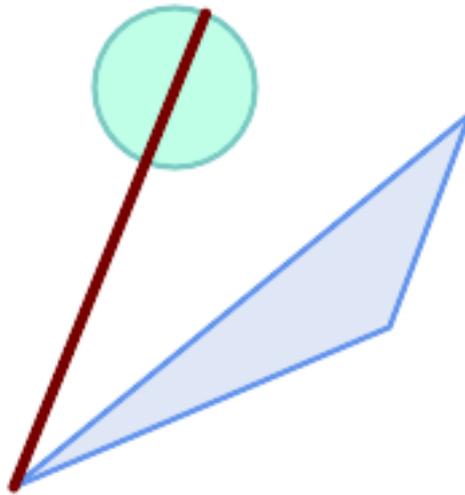
1.5.0 [ST\\_MaxDistance](#)

**Examples**



```
SELECT ST_AsText( ST_LongestLine(
  'POINT (160 40)',
  'LINESTRING (10 30, 50 50, 30 110, 70 90, 180 140, 130 190)' )
) AS lline;
```

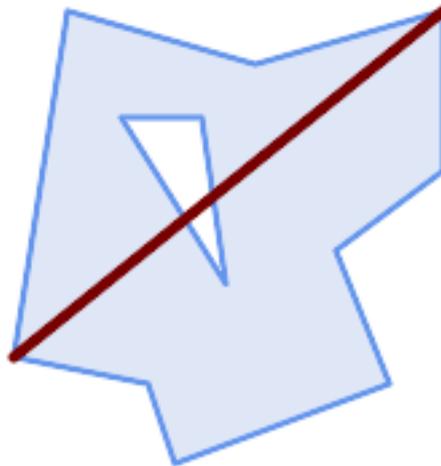
```
-----
LINESTRING(160 40,130 190)
```



The longest line through a point is the diameter of the minimum bounding circle of the polygon.

```

SELECT ST_AsText( ST_LongestLine(
  'POLYGON ((190 150, 20 10, 160 70, 190 150))',
  ST_Buffer('POINT(80 160)', 30)
) ) AS llinewkt;
-----
LINESTRING(20 10,105.3073372946034 186.95518130045156)
  
```



Longest line across a single geometry. The length of the line is equal to the Maximum Distance. The line is a diameter of the Minimum Bounding Circle.

```

SELECT ST_AsText( ST_LongestLine( geom, geom)) AS llinewkt,
       ST_MaxDistance( geom, geom) AS max_dist,
       ST_Length( ST_LongestLine(geom, geom)) AS lenl1
FROM (SELECT 'POLYGON ((40 180, 110 160, 180 180, 180 120, 140 90, 160 40, 80 10, 70 40, 20 ←
  50, 40 180),
  (60 140, 99 77.5, 90 140, 60 140))'::geometry AS geom) AS t;
-----
      llinewkt          |          max_dist          |          lenl1
-----+-----+-----
LINESTRING(20 50,180 180) | 206.15528128088303 | 206.15528128088303
  
```

**ST\_3DLongestLine**

[ST\\_MaxDistance](#), [ST\\_ShortestLine](#), [ST\\_3DLongestLine](#), [ST\\_MinimumBoundingCircle](#)

### 8.12.17 ST\_3DLongestLine

**ST\_3DLongestLine** — Returns the longest line segment within a 3D geometry.

#### Synopsis

geometry **ST\_3DLongestLine**(geometry g1, geometry g2);

**ST\_3DLongestLine**

Returns the longest line segment within a 3D geometry. The function is implemented as a recursive query. The function returns a geometry of type `LINESTRING`. The function is implemented as a recursive query. The function returns a geometry of type `LINESTRING`. The function is implemented as a recursive query. The function returns a geometry of type `LINESTRING`.

2.0.0 Returns the longest line segment within a 3D geometry.

2.2.0 Returns the longest line segment within a 3D geometry. The function is implemented as a recursive query. The function returns a geometry of type `LINESTRING`. The function is implemented as a recursive query. The function returns a geometry of type `LINESTRING`.

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

**ST\_3DLongestLine**

```

SELECT ST_AsEWKT(ST_3DLongestLine(line,pt)) AS lol3d_line_pt,
       ST_AsEWKT(ST_LongestLine(line,pt)) As lol2d_line_pt
FROM (SELECT 'POINT(100 100 30)::geometry As pt,
            'LINESTRING (20 80 20, 98 190 1, 110 180 3, 50 75 1000)>:: geometry As line
      ) As foo;

```

| lol3d_line_pt                     | lol2d_line_pt              |
|-----------------------------------|----------------------------|
| LINESTRING(50 75 1000,100 100 30) | LINESTRING(98 190,100 100) |

```
&#xb77c;&#xc778;&#xc2a4;&#xd2b8;&#xb9c1;&#xacfc; &#xba40;&#xd2f0;&#xd3ec;&#xc778;&#xd2b8; -- 3D, 2D
&#xbaa8;&#xb450;&#xc758; &#xcd5c;&#xc7a5; &#xb77c;&#xc778;
```

```
SELECT ST_AsEWKT(ST_3DLongestLine(line,pt)) AS lol3d_line_pt,
       ST_AsEWKT(ST_LongestLine(line,pt)) As lol2d_line_pt
FROM (SELECT 'MULTIPOINT(100 100 30, 50 74 1000)>::geometry As pt,
            'LINESTRING (20 80 20, 98 190 1, 110 180 3, 50 75 900)>:: ←
       geometry As line
       ) As foo;
```

```

           lol3d_line_pt           |           lol2d_line_pt
-----+-----
LINESTRING(98 190 1,50 74 1000) | LINESTRING(98 190,50 74)
```

```
&#xba40;&#xd2f0;&#xb77c;&#xc778;&#xc2a4;&#xd2b8;&#xb9c1;&#xacfc; &#xd3f4;&#xb9ac;&#xace4; -- 3D, 2D
&#xbaa8;&#xb450;&#xc758; &#xcd5c;&#xc7a5; &#xb77c;&#xc778;
```

```
SELECT ST_AsEWKT(ST_3DLongestLine(poly, mline)) As lol3d,
       ST_AsEWKT(ST_LongestLine(poly, mline)) As lol2d
FROM (SELECT ST_GeomFromEWKT('POLYGON((175 150 5, 20 40 5, 35 45 5, 50 60 5, ←
100 100 5, 175 150 5))') As poly,
       ST_GeomFromEWKT('MULTILINESTRING((175 155 2, 20 40 20, 50 60 -2, 125 ←
100 1, 175 155 1),
       (1 10 2, 5 20 1))') As mline ) As foo;
```

```

           lol3d           |           lol2d
-----+-----
LINESTRING(175 150 5,1 10 2) | LINESTRING(175 150,1 10)
```

**&#xc38;&#xace0;**

[ST\\_3DClosestPoint](#), [ST\\_3DDistance](#), [ST\\_LongestLine](#), [ST\\_3DShortestLine](#), [ST\\_3DMaxDistance](#)

## 8.12.18 ST\_MaxDistance

ST\_MaxDistance — &#xb450; &#xb3c4;&#xd615; &#xc0ac;&#xc774;&#xc758; 2&#xc28;&#xc6d0; &#xcd5c;&#xc7a5; &#xac70;&#xb9ac;&#xb97c; &#xd22c;&#xc601; &#xb2e8;&#xc704;&#xb85c; &#xbc18;&#xd658;&#xd569;&#xb2c8;&#xb2e4;

### Synopsis

float ST\_MaxDistance(geometry g1, geometry g2);

**&#xc124;&#xba85;**

&#xb450; &#xb3c4;&#xd615; &#xc0ac;&#xc774;&#xc758; 2&#xc28;&#xc6d0; &#xcd5c;&#xc7a5; &#xac70;&#xb9ac;&#xb97c; &#xd22c;&#xc601; &#xb2e8;&#xc704;&#xb85c; &#xbc18;&#xd658;&#xd569;&#xb2c8;&#xb2e4;. g1&#xacfc; g2&#xac00; &#xb3d9;&#xc77c;&#xd55c; &#xb3c4;&#xd615;&#xc77c; &#xacbd;&#xc6b0; &#xc774; &#xd568;&#xc218;&#xb294; &#xd574;&#xb3c4;&#xd615; &#xb0b4;&#xc5d0;&#xc11c; &#xc11c;&#xb85c; &#xac00;&#xc7a5; &#xba40;&#xb9ac; &#xc788;&#xb294; &#xb450; &#xaf2d;&#xc9d3;&#xc810; &#xc0ac;&#xc774;&#xc758; &#xac70;&#xb9ac;&#xb97c; &#xbc18;&#xd658;&#xd569;&#xb2c8;&#xb2e4;

&#xb450; &#xb3c4;&#xd615; &#xc0ac;&#xc774;&#xc758; 2&#xc28;&#xc6d0; &#xcd5c;&#xc7a5; &#xac70;&#xb9ac;&#xb97c; &#xd22c;&#xc601; &#xb2e8;&#xc704;&#xb85c; &#xbc18;&#xd658;&#xd569;&#xb2c8;&#xb2e4;. g1&#xacfc; g2&#xac00; &#xb3d9;&#xc77c;&#xd55c; &#xb3c4;&#xd615;&#xc77c; &#xacbd;&#xc6b0; &#xc774; &#xd568;&#xc218;&#xb294; &#xd574;&#xb3c4;&#xd615; &#xb0b4;&#xc5d0;&#xc11c; &#xc11c;&#xb85c; &#xac00;&#xc7a5; &#xba40;&#xb9ac; &#xc788;&#xb294; &#xb450; &#xaf2d;&#xc9d3;&#xc810; &#xc0ac;&#xc774;&#xc758; &#xac70;&#xb9ac;&#xb97c; &#xbc18;&#xd658;&#xd569;&#xb2c8;&#xb2e4;

1.5.0 &#xbc84;&#xc804;&#xbd80;&#xd130; &#xc0ac;&#xc6a9;&#xd560; &#xc218; &#xc788;&#xc2b5;&#xb2c8;&#xb2e4;

**Examples**

Distance between two points:

```
SELECT ST_MaxDistance('POINT(0 0)::geometry', 'LINESTRING ( 2 0, 0 2 )::geometry');
```

```
-----
2
```

```
SELECT ST_MaxDistance('POINT(0 0)::geometry', 'LINESTRING ( 2 2, 2 2 )::geometry');
```

```
-----
2.82842712474619
```

Maximum distance between vertices of a single geometry.

```
SELECT ST_MaxDistance('POLYGON ((10 10, 10 0, 0 0, 10 10))::geometry',
                      'POLYGON ((10 10, 10 0, 0 0, 10 10))::geometry');
```

```
-----
14.142135623730951
```

**See also**

[ST\\_Distance](#), [ST\\_LongestLine](#), [ST\\_DFullyWithin](#)

**8.12.19 ST\_3DMaxDistance**

**ST\_3DMaxDistance** — Returns the maximum distance between any two vertices of two 3D geometries. The distance is calculated using the 3D Euclidean distance formula. The function is implemented using the `ST_Distance` function with the `3D` flag. The function is implemented using the `ST_Distance` function with the `3D` flag.

**Synopsis**

```
float ST_3DMaxDistance(geometry g1, geometry g2);
```

**Options**

The function supports the following options:

- `3D`: Calculate the 3D distance.
- `2D`: Calculate the 2D distance.
- `0`: Calculate the 0D distance (point-to-point).
- `1`: Calculate the 1D distance (line-to-line).
- `2`: Calculate the 2D distance (area-to-area).
- `3`: Calculate the 3D distance (volume-to-volume).



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



This function supports Polyhedral surfaces.

2.0.0: Returns the maximum distance between any two vertices of two 3D geometries.

2.2.0: Returns the maximum distance between any two vertices of two 3D geometries. The function is implemented using the `ST_Distance` function with the `3D` flag.

**Examples**

```

-- &#xb3c4;&#xd615; &#xc608;&#xc2dc; - &#xbbf8;&#xd130; &#xb2e8;&#xc704; (SRID 2163 ←
&#xbbf8;&#xad6d; &#xb4f1;&#xc801; &#xc804;&#xb3c4;) (2D &#xd3ec;&#xc778;&#xd2b8; ←
&#xbc0f; &#xb77c;&#xc778;&#xacfc; &#be44;&#xad50;&#xd55c; 3D &#xd3ec;&#xc778;&#xd2b8; ←
&#xbc0f; &#xb77c;&#xc778;)
-- &#xc8fc;&#xc758;: &#xd604;&#xc7ac; &#xc218;&#xc9c1; &#xb370;&#xc774;&#xd130;&#xb97c; ←
&#xc9c0;&#xc6d0;&#xd558;&#xc9c0; &#xc54a;&#xc73c;&#xbbc0;&#xb85c; Z&#xb97c; ←
&#xbcc0;&#xd658;&#xd558;&#xc9c0; &#xc54a;&#xace0; &#xcd5c;&#xc885; ←
&#xacb0;&#xacfc;&#xbb3c;&#xacfc; &#xb3d9;&#xc77c;&#xd55c; &#xb2e8;&#xc704;&#xb85c; ←
&#xac00;&#xc815;&#xd569;&#xb2c8;&#xb2e4;.
SELECT ST_3DMaxDistance (
    ST_Transform(ST_GeomFromEWKT ('SRID=4326;POINT(-72.1235 42.3521 ←
10000)'), 2163),
    ST_Transform(ST_GeomFromEWKT ('SRID=4326;LINESTRING(-72.1260 42.45 ←
15, -72.123 42.1546 20)'), 2163)
) As dist_3d,
ST_MaxDistance (
    ST_Transform(ST_GeomFromEWKT ('SRID=4326;POINT(-72.1235 42.3521 ←
10000)'), 2163),
    ST_Transform(ST_GeomFromEWKT ('SRID=4326;LINESTRING(-72.1260 42.45 ←
15, -72.123 42.1546 20)'), 2163)
) As dist_2d;

    dist_3d      |      dist_2d
-----+-----
24383.7467488441 | 22247.8472107251

```

**&#xcc38;&#xace0;**

[ST\\_Distance](#), [ST\\_3DDWithin](#), [ST\\_3DMaxDistance](#), [ST\\_Transform](#)

## 8.12.20 ST\_MinimumClearance

`ST_MinimumClearance` — `&#xb3c4;&#xd615;&#xc758; &#xd2bc;&#xd2bc;&#xd568;(robustness)&#xc758; &#xcc99;&#xb3c4;&#xb3c4;&#xd615;&#xc758; &#xcd5c;&#xc18c; &#xc5ec;&#xc720;(clearance)&#xb97c; &#xbc18;&#xd658;&#xd569;&#xb2c8;&#xb2e4;`

### Synopsis

`float ST_MinimumClearance(geometry g);`

**&#xc124;&#xba85;**

`(&#xd3f4;&#xb9ac;&#xace4;&#xc77c; &#xacbd;&#xc6b0;) ST_IsValid &#xb610;&#xb294; (&#xb77c;&#xc778;&#xc77c; &#xacbd;&#xc6b0; &#xd568;&#xc218;&#xc5d0; &#xb530;&#xb77c; &#xc720;&#xd6a8;&#xc131;&#xc5d0; &#xb300;&#xd55c; &#xae30;&#xc900;&#xc744; &#xb9cc;&#xc871;&#xc2dc;&#xd0a4;&#xc9c0;&#xb9cc;, &#xd14d;&#xc2a4;&#xd2b8; &#xae30;&#xc900;&#xd615;&#xc2dd;(WKT, KML, GML GeoJSON &#xb4f1;) &#xb610;&#xb294; &#xc774;&#xc911; &#xc815;&#xbc00;&#xb3c4; &#xbd80;&#xb3d9;&#xc18c;&#xc218;&#xc810; &#xc88c;&#xd45c;&#xb97c; &#xc774;&#xc6a9;&#xd558;&#xc9c0; &#xc54a;&#xb97c; &#xb3c4;&#xc774;&#xb108;&#xb9ac; &#xd615;&#xc2dd;(MapInfo TAB &#xb4f1;) &#xc73c;&#xb85c; &#xbcc0;&#xd658;&#xd558; &#xacfc;&#xc815;&#xc5d0;&#xc11c; &#xc77c;&#xc5b4;&#xb0a0; &#xc218; &#xc788;&#xb294; &#xac83;&#xcc98;&#xb7fc;, &#xaf2d;&#xc9d3;&#xc810; &#xac00;&#xc6b4;&#xb370; &#xd558;&#xb098;&#xac00; &#xc0b4;&#xc9dd; &#xc774;&#xb3d9;&#xc720;&#xd6a8;&#xd558;&#xc9c0; &#xc54a;&#xc544;&#xc9c0;&#xb294; &#xb3c4;&#xd615;&#xc774; &#xadf8;&#xb807;&#xb2e4;&#xd76c;&#xad0;&#xd55c; &#xac83;&#xc740; &#xc544;&#xb2d9;&#xb2c8;&#xb2e4;`

The minimum clearance is a quantitative measure of a geometry's robustness to change in coordinate precision. It is the largest distance by which vertices of the geometry can be moved without creating an invalid geometry. Larger values of minimum clearance indicate greater robustness.

ST\_MinimumClearanceLine, ST\_Crosses, ST\_Dimension, ST\_Intersects

- ST\_MinimumClearanceLine, ST\_Crosses, ST\_Dimension, ST\_Intersects
- ST\_MinimumClearanceLine, ST\_Crosses, ST\_Dimension, ST\_Intersects

ST\_MinimumClearanceLine, ST\_Crosses, ST\_Dimension, ST\_Intersects

To avoid validity issues caused by precision loss, `ST_ReducePrecision` can reduce coordinate precision while ensuring that polygonal geometry remains valid.

2.3.0 ST\_MinimumClearanceLine, ST\_Crosses, ST\_Dimension, ST\_Intersects

### ST\_MinimumClearanceLine

```
SELECT ST_MinimumClearanceLine('POLYGON ((0 0, 1 0, 1 1, 0.5 3.2e-4, 0 0))');
st_minimumclearance
-----
0.00032
```

### ST\_MinimumClearanceLine

ST\_MinimumClearanceLine, ST\_Crosses, ST\_Dimension, ST\_Intersects

## 8.12.21 ST\_MinimumClearanceLine

ST\_MinimumClearanceLine — Returns the two-point LineString spanning a geometry's minimum clearance. If the geometry does not have a minimum clearance, `LINestring EMPTY` is returned.

### Synopsis

Geometry `ST_MinimumClearanceLine`(geometry g);

### ST\_MinimumClearanceLine

Returns the two-point LineString spanning a geometry's minimum clearance. If the geometry does not have a minimum clearance, `LINestring EMPTY` is returned.

GEOS 3.6.0 ST\_MinimumClearanceLine, ST\_Crosses, ST\_Dimension, ST\_Intersects

2.3.0 ST\_MinimumClearanceLine, ST\_Crosses, ST\_Dimension, ST\_Intersects

GEOS 3.6.0 ST\_MinimumClearanceLine, ST\_Crosses, ST\_Dimension, ST\_Intersects



```
SELECT ST_Perimeter(ST_GeomFromText('POLYGON((743238 2967416,743238 2967450,743265 2967450,743265.625 2967416,743238 2967416))', 2249));
st_perimeter
-----
122.630744000095
(1 row)
```

```
SELECT ST_Perimeter(ST_GeomFromText('MULTIPOLYGON(((763104.471273676 2949418.44119003,763104.477769673 2949418.42538203,763104.189609677 2949418.22343004,763104.471273676 2949418.44119003)),((763104.471273676 2949418.44119003,763095.804579742 2949436.33850239,763086.132105649 2949451.46730207,763078.452329651 2949462.11549407,763075.354136904 2949466.17407812,763064.362142565 2949477.64291974,763059.953961626 2949481.28983009,762994.637609571 2949532.04103014,762990.568508415 2949535.06640477,762986.710889563 2949539.61421415,763117.237897679 2949709.50493431,763235.236617789 2949617.95619822,763287.718121842 2949562.20592617,763111.553321674 2949423.91664605,763104.471273676 2949418.44119003)))', 2249));
st_perimeter
-----
845.227713366825
(1 row)
```

**Perimeter of a polygon in meters and feet**

Example: A polygon with vertices at (-71.1776848522251 42.3902896512902, -71.1776843766326 42.3903829478009, -71.1775844305465 42.3903826677917, -71.1775825927231 42.3902893647987, -71.1776848522251 42.3902896512902) in WGS84 projection.

```
SELECT ST_Perimeter(geog) As per_meters, ST_Perimeter(geog)/0.3048 As per_ft
FROM ST_GeogFromText('POLYGON((-71.1776848522251 42.3902896512902,-71.1776843766326 ↵
42.3903829478009,
-71.1775844305465 42.3903826677917,-71.1775825927231 42.3902893647987,-71.1776848522251 ↵
42.3902896512902))') As geog;

per_meters | per_ft
-----+-----
37.3790462565251 | 122.634666195949
```

```
-- Example: A polygon with vertices at (-71.1044543107478 42.340674480411, -71.1044542869917 42.3406744369506,
-71.1044553562977 42.340673886454, -71.1044543107478 42.340674480411),
((-71.1044543107478 42.340674480411,-71.1044860600303 42.3407237015564,-71.1045215770124 ↵
42.3407653385914,
-71.1045498002983 42.3407946553165,-71.1045611902745 42.3408058316308,-71.1046016507427 ↵
42.340837442371,
-71.104617893173 42.3408475056957,-71.1048586153981 42.3409875993595,-71.1048736143677 ↵
42.3409959528211,
-71.1048878050242 42.3410084812078,-71.1044020965803 42.3414730072048,
-71.1039672113619 42.3412202916693,-71.1037740497748 42.3410666421308,
-71.1044280218456 42.3406894151355,-71.1044543107478 42.340674480411))) As geog;

per_meters | per_sphere_meters | per_ft
-----+-----+-----
257.634283683311 | 257.412311446337 | 845.256836231335
```

**ST\_Perimeter2D**

[ST\\_GeogFromText](#), [ST\\_GeomFromText](#), [ST\\_Length](#)

### 8.12.23 ST\_Perimeter2D

`ST_Perimeter2D` — Returns the 2D perimeter of a polygonal geometry. Alias for `ST_Perimeter`.

#### Synopsis

```
float ST_Perimeter2D(geometry geomA);
```

**Notes**

ST\_Perimeter2D(geomA) returns the 2D perimeter of the geometry geomA. The result is of type float. The function is commutative. The function is not commutative with respect to the order of the arguments. The function is not commutative with respect to the order of the arguments.

#### Note



ST\_Perimeter2D(geomA) returns the 2D perimeter of the geometry geomA. The result is of type float. The function is commutative. The function is not commutative with respect to the order of the arguments. The function is not commutative with respect to the order of the arguments.

**ST\_Perimeter**

[ST\\_Perimeter](#)

### 8.12.24 ST\_3DPerimeter

`ST_3DPerimeter` — Returns the 3D perimeter of a polygonal geometry. The result is of type float. The function is commutative. The function is not commutative with respect to the order of the arguments. The function is not commutative with respect to the order of the arguments.

#### Synopsis

```
float ST_3DPerimeter(geometry geomA);
```

**Notes**

ST\_3DPerimeter(geomA) returns the 3D perimeter of the geometry geomA. The result is of type float. The function is commutative. The function is not commutative with respect to the order of the arguments. The function is not commutative with respect to the order of the arguments.



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



This method implements the SQL/MM specification. SQL-MM ISO/IEC 13249-3: 8.1, 10.5

ST\_3DPerimeter(geomA) returns the 3D perimeter of the geometry geomA. The result is of type float. The function is commutative. The function is not commutative with respect to the order of the arguments. The function is not commutative with respect to the order of the arguments.

**ST\_3DPerimeter**

Computes the 3D perimeter of a geometry. The result is the sum of the lengths of all edges of the geometry.

```
SELECT ST_3DPerimeter(the_geom), ST_Perimeter2d(the_geom), ST_Perimeter(the_geom) FROM
  (SELECT ST_GeomFromEWKT('SRID=2249;POLYGON((743238 2967416 2, 743238 ←
    2967450 1,
  743265.625 2967416 1, 743238 2967416 2))') As the_geom) As foo;
```

| ST_3DPerimeter   | st_perimeter2d   | st_perimeter     |
|------------------|------------------|------------------|
| 105.465793597674 | 105.432997272188 | 105.432997272188 |

**ST\_GeomFromEWKT**

[ST\\_GeomFromEWKT](#), [ST\\_Perimeter](#), [ST\\_Perimeter2D](#)

**8.12.25 ST\_Project**

**ST\_Project** — Computes the point on a line segment at a given distance and azimuth from the start point. The result is the point on the line segment that is the given distance from the start point in the direction of the azimuth.

**Synopsis**

geography **ST\_Project**(geography g1, float distance, float azimuth);

**ST\_Project**

Computes the point on a line segment at a given distance and azimuth from the start point. The result is the point on the line segment that is the given distance from the start point in the direction of the azimuth. The distance is measured along the line segment. The azimuth is measured in radians from the positive x-axis.

Enhanced: 2.4.0 Allow negative distance and non-normalized azimuth.

**ST\_Project**: Computes the point on a line segment at a given distance and azimuth from the start point. The result is the point on the line segment that is the given distance from the start point in the direction of the azimuth.

```
SELECT ST_AsText(ST_Project('POINT(0 0)::geography, 100000, radians(45.0)));
```

```
POINT(0.635231029125537 0.639472334729198)
```

**ST\_ShortestLine**

[ST\\_Azimuth](#), [ST\\_Distance](#), [PostgreSQL Math Functions](#)

### 8.12.26 ST\_ShortestLine

**ST\_ShortestLine** — Returns the 2-dimensional shortest line between two geometries. The line returned starts in `geom1` and ends in `geom2`. If `geom1` and `geom2` intersect the result is a line with start and end at an intersection point. The length of the line is the same as [ST\\_Distance](#) returns for `g1` and `g2`.

#### Synopsis

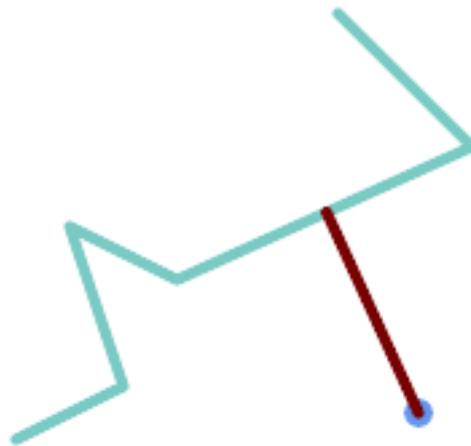
```
geometry ST_ShortestLine(geometry geom1, geometry geom2);
```

#### Parameters

Returns the 2-dimensional shortest line between two geometries. The line returned starts in `geom1` and ends in `geom2`. If `geom1` and `geom2` intersect the result is a line with start and end at an intersection point. The length of the line is the same as [ST\\_Distance](#) returns for `g1` and `g2`.

1.5.0-2015-08-17

#### Examples



*Shortest line between Point and LineString*

```
SELECT ST_AsText( ST_ShortestLine(
    'POINT (160 40)',
    'LINESTRING (10 30, 50 50, 30 110, 70 90, 180 140, 130 190)'
) As sline;
-----
LINESTRING(160 40,125.75342465753425 115.34246575342466)
```



g1#xc5d0;#xc11c; #xc2dc;#xc791; g2#xc5d0;#xc11c; #xb05d;#xb0a9;#xb2c8;#xb2e4;. #xc774; #xd568;#xc218;#xac00;#xc18;#xd658;#xd558;#xb77c;#xc778;#xc758; 3#xc28;#xc6d0; #xae38;#xc774;#xb294; **ST\_3DDistance** #xd568;#xc218;#xac00; g1#xacfc; g2#xc5d0; #xb300;#xd574; #xc18;#xd658;#xd558;#xb294; #xae38;#xc774;#xc640; #xc5b8;#xc81;#xb3d9;#xc77c;#xd569;#xb2c8;#xb2e4;.

2.0.0 #xc84;#xc804;#xbd80;#xd130; #xc0ac;#xc6a9;#xd560; #xc218; #xc788;#xc2b5;#xb2c8;#xb2e4;.

#xbcc0;#xacbd; #xc0ac;#xd56d;. 2.2.0 #xc84;#xc804;#xbd80;#xd130; 2D #xb3c4;#xd615; #xb450; #xac1c;#xc785;#xb825;#xd560; #xacbd;#xc6b0;. (#xc874;#xc7ac;#xd558;#xc9c0; #xc54a;#xb294; Z#xc744; 0#xc73c;#xb85c; #xac00;#xc815;#xd558;#xb294; #xc608;#xc804; #xc2b5;#xc131; #xb300;#xc2e0;) 2D #xd3ec;#xc778;#xd2b8;#xb97c; #xc18;#xd658;#xd569;#xb2c8;#xb2e4;. 2D #xc0f; 3D#xc758; #xacbd;#xc0ac;#xb354; #xc774;#xc0c1; Z#xac00; #xc5c6;#xc744; #xb54c; Z#xb97c; 0#xc73c;#xb85c; #xac00;#xc815;#xd558;#xc54a;#xc2b5;#xb2c8;#xb2e4;.



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



This function supports Polyhedral surfaces.

### #xc608;#xc2dc;

```
#xb77c;#xc778;#xc2a4;#xd2b8;#xb9c1;#xacfc; #xd3ec;#xc778;#xd2b8; -- 3D, 2D
#xbaa8;#xb450;#xc758; #xcd5c;#xb2e8; #xb77c;#xc778;

SELECT ST_AsEWKT(ST_3DShortestLine(line,pt)) AS shl3d_line_pt,
       ST_AsEWKT(ST_ShortestLine(line,pt)) As shl2d_line_pt
FROM (SELECT 'POINT(100 100 30)::geometry As pt,
            'LINESTRING (20 80 20, 98 190 1, 110 180 3, 50 75 1000)>::: ←
      geometry As line
      ) As foo;

shl3d_line_pt | ←
-----+-----
shl2d_line_pt | ←

LINESTRING(54.6993798867619 128.935022917228 11.5475869506606,100 100 30) | ←
LINESTRING(73.0769230769231 115.384615384615,100 100)

#xb77c;#xc778;#xc2a4;#xd2b8;#xb9c1;#xacfc; #xba40;#xd2f0;#xd3ec;#xc778;#xd2b8; -- 3D, 2D
#xbaa8;#xb450;#xc758; #xcd5c;#xb2e8; #xb77c;#xc778;

SELECT ST_AsEWKT(ST_3DShortestLine(line,pt)) AS shl3d_line_pt,
       ST_AsEWKT(ST_ShortestLine(line,pt)) As shl2d_line_pt
FROM (SELECT 'MULTIPOINT(100 100 30, 50 74 1000)>:::geometry As pt,
            'LINESTRING (20 80 20, 98 190 1, 110 180 3, 50 75 900)>::: ←
      geometry As line
      ) As foo;

shl2d_line_pt | ←
shl3d_line_pt | ←
-----+-----

LINESTRING(54.6993798867619 128.935022917228 11.5475869506606,100 100 30) | LINESTRING ←
(50 75,50 74)
```

```
-- 3D, 2D
-->
```

```
SELECT ST_AsEWKT(ST_3DShortestLine(poly, mline)) As shl3d,
       ST_AsEWKT(ST_ShortestLine(poly, mline)) As shl2d
FROM (SELECT ST_GeomFromEWKT('POLYGON((175 150 5, 20 40 5, 35 45 5, 50 60 5,
100 100 5, 175 150 5))') As poly,
      ST_GeomFromEWKT('MULTILINESTRING((175 155 2, 20 40 20, 50 60 -2, 125
100 1, 175 155 1),
      (1 10 2, 5 20 1))') As mline ) As foo;
shl3d | shl2d
-----|-----
LINESTRING(39.993580415989 54.1889925532825 5,40.4078575708294 53.6052383805529
5.03423778139177) | LINESTRING(20 40,20 40)
```

**ST\_3DClosestPoint, ST\_3DDistance, ST\_LongestLine, ST\_ShortestLine, ST\_3DMaxDistance**

**ST\_3DClosestPoint, ST\_3DDistance, ST\_LongestLine, ST\_ShortestLine, ST\_3DMaxDistance**

## 8.13 Overlay Functions

### 8.13.1 ST\_ClipByBox2D

**ST\_ClipByBox2D** — Computes the portion of a geometry falling within a rectangle.

#### Synopsis

geometry **ST\_ClipByBox2D**(geometry geom, box2d box);

#### Description

Clips a geometry by a 2D box in a fast and tolerant but possibly invalid way. Topologically invalid input geometries do not result in exceptions being thrown. The output geometry is not guaranteed to be valid (in particular, self-intersections for a polygon may be introduced).

Performed by the GEOS module.

Availability: 2.2.0

#### Examples

```
-- Rely on implicit cast from geometry to box2d for the second parameter
SELECT ST_ClipByBox2D(geom, ST_MakeEnvelope(0,0,10,10)) FROM mytab;
```

#### See Also

**ST\_Intersection, ST\_MakeBox2D, ST\_MakeEnvelope**

### 8.13.2 ST\_Difference

**ST\_Difference** — Computes a geometry representing the part of geometry A that does not intersect geometry B.

## Synopsis

```
geometry ST_Difference(geometry geomA, geometry geomB, float8 gridSize = -1);
```

## Description

Returns a geometry representing the part of geometry A that does not intersect geometry B. This is equivalent to  $A - \text{ST\_Intersect}(A, B)$ . If A is completely contained in B then an empty atomic geometry of appropriate type is returned.



### Note

This is the only overlay function where input order matters. `ST_Difference(A, B)` always returns a portion of A.

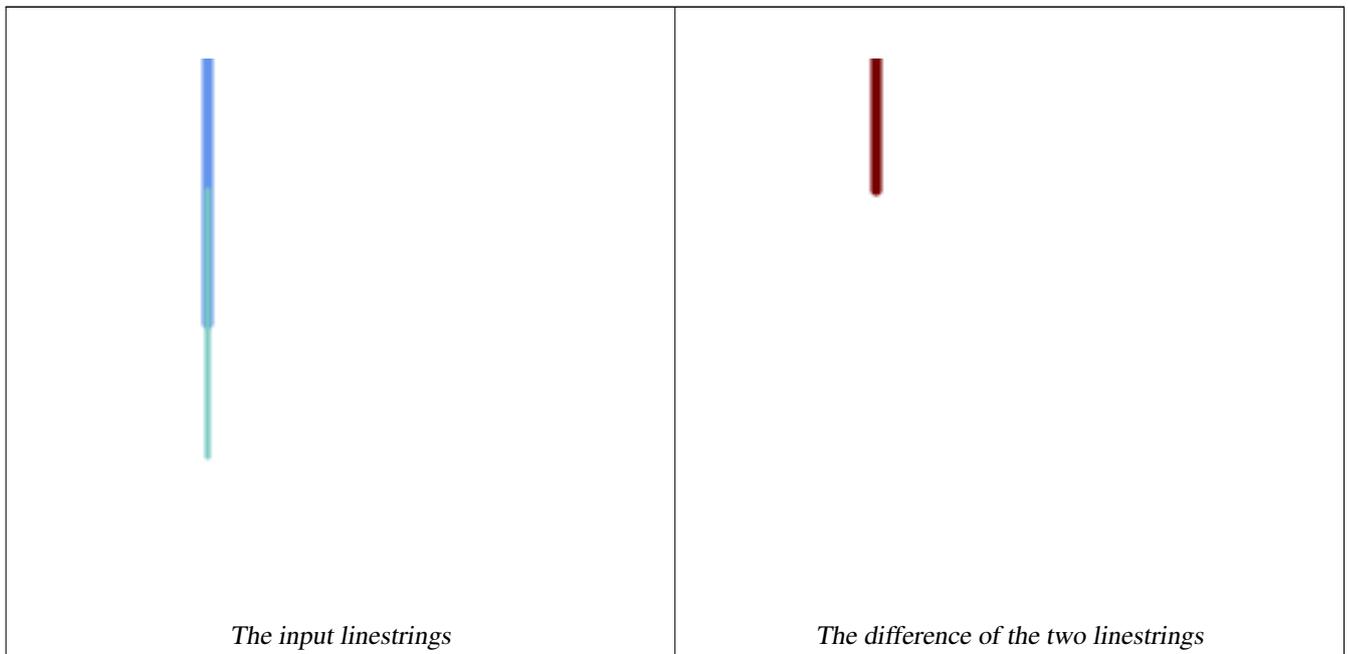
If the optional `gridSize` argument is provided, the inputs are snapped to a grid of the given size, and the result vertices are computed on that same grid. (Requires GEOS-3.9.0 or higher)

Performed by the GEOS module

Enhanced: 3.1.0 accept a `gridSize` parameter - requires GEOS  $\geq$  3.9.0

-  This method implements the [OGC Simple Features Implementation Specification for SQL 1.1](#). s2.1.1.3
-  This method implements the SQL/MM specification. SQL-MM 3: 5.1.20
-  This function supports 3d and will not drop the z-index. However, the result is computed using XY only. The result Z values are copied, averaged or interpolated.

## Examples



The difference of 2D linestrings.

```
SELECT ST_AsText (
```

```

ST_Difference(
  'LINESTRING(50 100, 50 200)::geometry,
  'LINESTRING(50 50, 50 150)::geometry
)
);

st_astext
-----
LINESTRING(50 150,50 200)

```

The difference of 3D points.

```

SELECT ST_AsEWKT( ST_Difference(
  'MULTIPOINT(-118.58 38.38 5,-118.60 38.329 6,-118.614 38.281 7)' :: geometry,
  'POINT(-118.614 38.281 5)' :: geometry
) );

st_asewkt
-----
MULTIPOINT(-118.6 38.329 6,-118.58 38.38 5)

```

#### See Also

[ST\\_SymDifference](#), [ST\\_Intersection](#), [ST\\_Union](#)

### 8.13.3 ST\_Intersection

**ST\_Intersection** — Computes a geometry representing the shared portion of geometries A and B.

#### Synopsis

```

geometry ST_Intersection( geometry geomA , geometry geomB , float8 gridSize = -1 );
geography ST_Intersection( geography geogA , geography geogB );

```

#### Description

Returns a geometry representing the point-set intersection of two geometries. In other words, that portion of geometry A and geometry B that is shared between the two geometries.

If the geometries have no points in common (i.e. are disjoint) then an empty atomic geometry of appropriate type is returned.

If the optional `gridSize` argument is provided, the inputs are snapped to a grid of the given size, and the result vertices are computed on that same grid. (Requires GEOS-3.9.0 or higher)

**ST\_Intersection** in conjunction with **ST\_Intersects** is useful for clipping geometries such as in bounding box, buffer, or region queries where you only require the portion of a geometry that is inside a country or region of interest.

---

#### Note



Geography: For geography this is really a thin wrapper around the geometry implementation. It first determines the best SRID that fits the bounding box of the 2 geography objects (if geography objects are within one half zone UTM but not same UTM will pick one of those) (favoring UTM or Lambert Azimuthal Equal Area (LAEA) north/south pole, and falling back on mercator in worst case scenario) and then intersection in that best fit planar spatial ref and retransforms back to WGS84 geography.

---

**Warning**

This function will drop the M coordinate values if present.

**Warning**

If working with 3D geometries, you may want to use SFCGAL based [ST\\_3DIntersection](#) which does a proper 3D intersection for 3D geometries. Although this function works with Z-coordinate, it does an averaging of Z-Coordinate.

Performed by the GEOS module

Enhanced: 3.1.0 accept a gridSize parameter - requires GEOS >= 3.9.0

Changed: 3.0.0 does not depend on SFCGAL.

Availability: 1.5 support for geography data type was introduced.



This method implements the [OGC Simple Features Implementation Specification for SQL 1.1](#). s2.1.1.3



This method implements the SQL/MM specification. SQL-MM 3: 5.1.18



This function supports 3d and will not drop the z-index. However, the result is computed using XY only. The result Z values are copied, averaged or interpolated.

**Examples**

```
SELECT ST_AsText(ST_Intersection('POINT(0 0)::geometry, 'LINESTRING ( 2 0, 0 2 ) '::
  geometry));
 st_astext
-----
GEOMETRYCOLLECTION EMPTY

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

Clip all lines (trails) by country. Here we assume country geom are POLYGON or MULTIPOLYGONS. NOTE: we are only keeping intersections that result in a LINESTRING or MULTILINESTRING because we don't care about trails that just share a point. The dump is needed to expand a geometry collection into individual single MULT\* parts. The below is fairly generic and will work for polys, etc. by just changing the where clause.

```
select clipped.gid, clipped.f_name, clipped_geom
from (
  select trails.gid, trails.f_name,
         (ST_Dump(ST_Intersection(country.geom, trails.geom))).geom clipped_geom
  from country
       inner join trails on ST_Intersects(country.geom, trails.geom)
) as clipped
where ST_Dimension(clipped.clipped_geom) = 1;
```

For polys e.g. polygon landmarks, you can also use the sometimes faster hack that buffering anything by 0.0 except a polygon results in an empty geometry collection. (So a geometry collection containing polys, lines and points buffered by 0.0 would only leave the polygons and dissolve the collection shell.)

```

select poly.gid,
       ST_Multi(
         ST_Buffer(
           ST_Intersection(country.geom, poly.geom),
           0.0
         )
       ) clipped_geom
from country
     inner join poly on ST_Intersects(country.geom, poly.geom)
where not ST_IsEmpty(ST_Buffer(ST_Intersection(country.geom, poly.geom), 0.0));

```

### Examples: 2.5Dish

Note this is not a true intersection, compare to the same example using [ST\\_3DIntersection](#).

```

select ST_AsText(ST_Intersection(linestring, polygon)) As wkt
from ST_GeomFromText('LINESTRING Z (2 2 6,1.5 1.5 7,1 1 8,0.5 0.5 8,0 0 10)') AS linestring
     CROSS JOIN ST_GeomFromText('POLYGON((0 0 8, 0 1 8, 1 1 8, 1 0 8, 0 0 8))') AS polygon;

-----
           st_astext
-----
LINESTRING Z (1 1 8,0.5 0.5 8,0 0 10)

```

### See Also

[ST\\_3DIntersection](#), [ST\\_Difference](#), [ST\\_Union](#), [ST\\_Dimension](#), [ST\\_Dump](#), [ST\\_Force2D](#), [ST\\_SymDifference](#), [ST\\_Intersects](#), [ST\\_Multi](#)

## 8.13.4 ST\_MemUnion

`ST_MemUnion` — Aggregate function which unions geometries in a memory-efficient but slower way

### Synopsis

```
geometry ST_MemUnion(geometry set geomfield);
```

### Description

An aggregate function that unions the input geometries, merging them to produce a result geometry with no overlaps. The output may be a single geometry, a MultiGeometry, or a Geometry Collection.



#### Note

Produces the same result as [ST\\_Union](#), but uses less memory and more processor time. This aggregate function works by unioning the geometries incrementally, as opposed to the `ST_Union` aggregate which first accumulates an array and then unions the contents using a fast algorithm.



This function supports 3d and will not drop the z-index. However, the result is computed using XY only. The result Z values are copied, averaged or interpolated.

## Examples

```
SELECT id,
       ST_MemUnion(geom) as singlegeom
FROM sometable f
GROUP BY id;
```

## See Also

[ST\\_Union](#)

### 8.13.5 ST\_Node

`ST_Node` — Nodes a collection of lines.

#### Synopsis

geometry **ST\_Node**(geometry geom);

#### Description

Returns a (Multi)LineString representing the fully noded version of a collection of linestrings. The noding preserves all of the input nodes, and introduces the least possible number of new nodes. The resulting linework is dissolved (duplicate lines are removed).

This is a good way to create fully-noded linework suitable for use as input to [ST\\_Polygonize](#).



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

Performed by the GEOS module.

Availability: 2.0.0

Changed: 2.4.0 this function uses `GEOSNode` internally instead of `GEOSUnaryUnion`. This may cause the resulting linestrings to have a different order and direction compared to PostGIS < 2.4.

## Examples

Noding a 3D LineString which self-intersects

```
SELECT ST_AsText (
    ST_Node('LINESTRINGZ(0 0 0, 10 10 10, 0 10 5, 10 0 3)::geometry')
) As output;
output
-----
MULTILINESTRING Z ((0 0 0,5 5 4.5),(5 5 4.5,10 10 10,0 10 5,5 5 4.5),(5 5 4.5,10 0 3))
```

Noding two LineStrings which share common linework. Note that the result linework is dissolved.

```
SELECT ST_AsText (
    ST_Node('MULTILINESTRING ((2 5, 2 1, 7 1), (6 1, 4 1, 2 3, 2 5))::geometry')
) As output;
output
-----
MULTILINESTRING((2 5,2 3),(2 3,2 1,4 1),(4 1,2 3),(4 1,6 1),(6 1,7 1))
```

## See Also

[ST\\_UnaryUnion](#)

### 8.13.6 ST\_Split

`ST_Split` — Returns a collection of geometries created by splitting a geometry by another geometry.

#### Synopsis

```
geometry ST_Split(geometry input, geometry blade);
```

#### Description

The function supports splitting a `LineString` by a `(Multi)Point`, `(Multi)LineString` or `(Multi)Polygon` boundary, or a `(Multi)Polygon` by a `LineString`. When a `(Multi)Polygon` is used as the blade, its linear components (the boundary) are used for splitting the input. The result geometry is always a collection.

This function is in a sense the opposite of [ST\\_Union](#). Applying `ST_Union` to the returned collection should theoretically yield the original geometry (although due to numerical rounding this may not be exactly the case).



#### Note

If the the input and blade do not intersect due to numerical precision issues, the input may not be split as expected. To avoid this situation it may be necessary to snap the input to the blade first, using [ST\\_Snap](#) with a small tolerance.

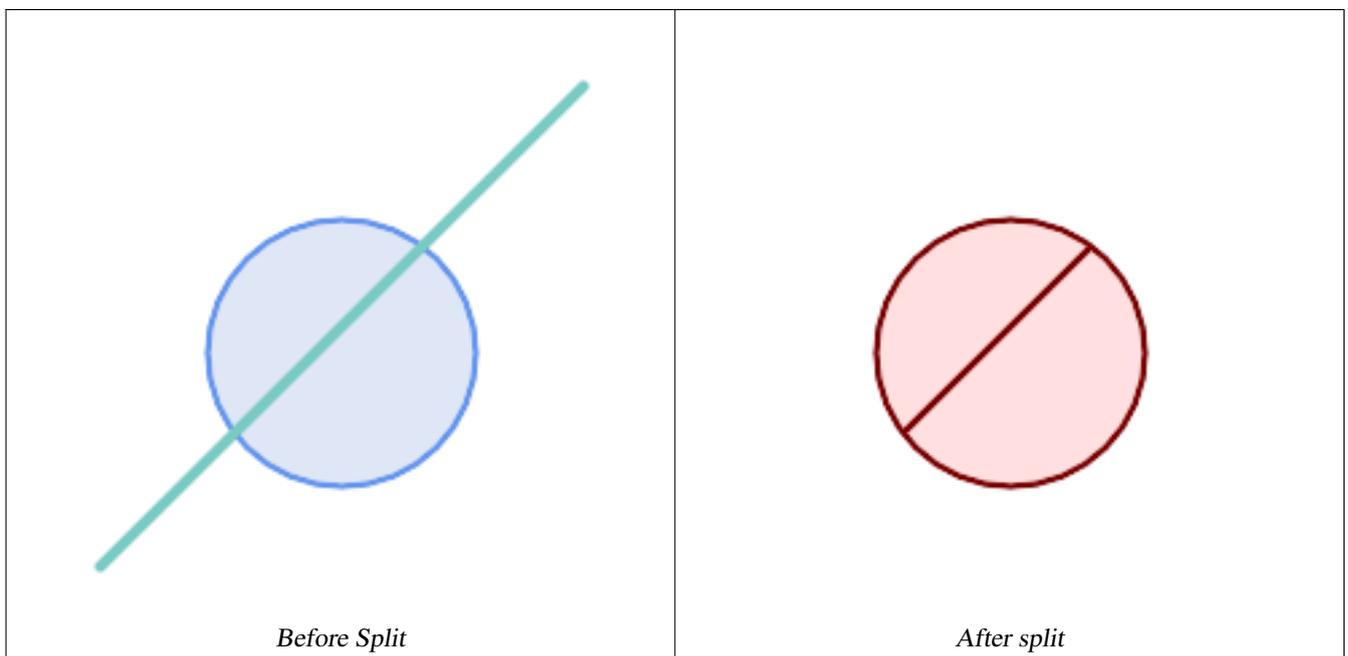
Availability: 2.0.0 requires GEOS

Enhanced: 2.2.0 support for splitting a line by a multiline, a multipoint or (multi)polygon boundary was introduced.

Enhanced: 2.5.0 support for splitting a polygon by a multiline was introduced.

#### Examples

Polygon split by a Line



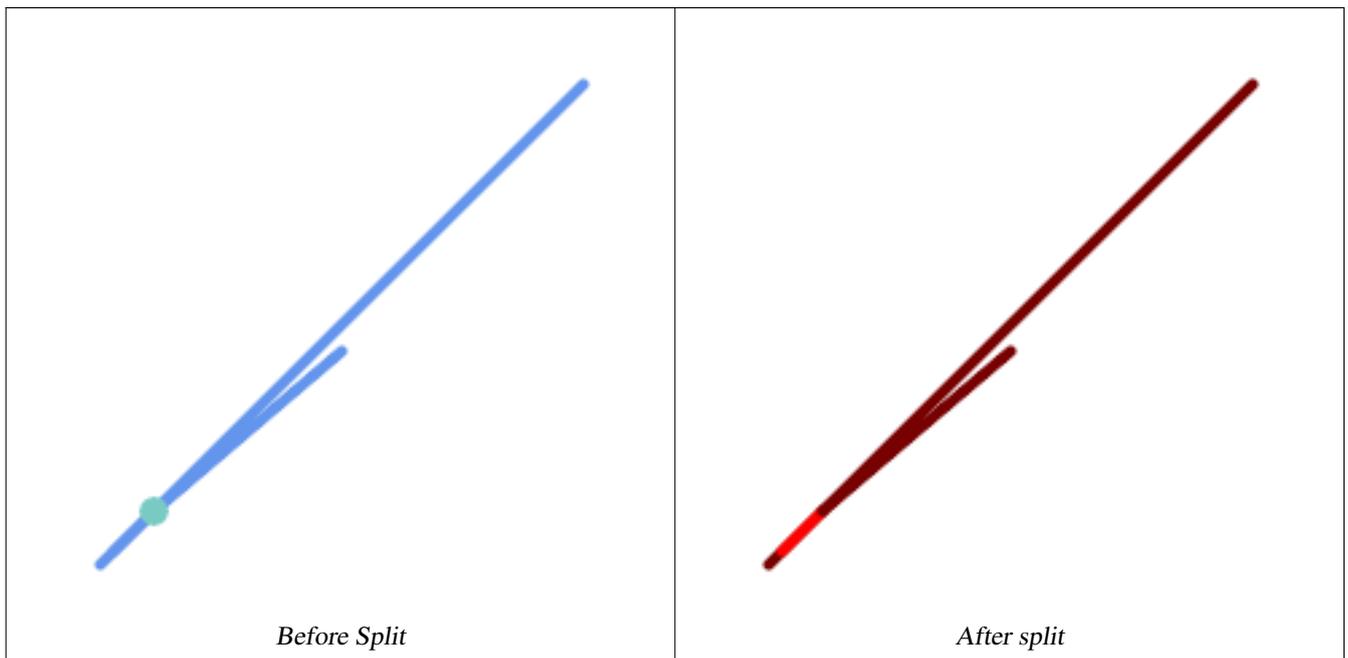
```

SELECT ST_AsText( ST_Split(
    ST_Buffer(ST_GeomFromText('POINT(100 90)'), 50), -- circle
    ST_MakeLine(ST_Point(10, 10),ST_Point(190, 190)) -- line
));

-- result --
GEOMETRYCOLLECTION(
  POLYGON((150 90,149.039264020162 80.2454838991936,146.193976625564 ←
    70.8658283817455,..),
  POLYGON(..)
)

```

MultiLineString split by a Point, where the point lies exactly on both LineStrings.



```

SELECT ST_AsText(ST_Split(
  'MULTILINESTRING((10 10, 190 191), (15 15, 30 30, 100 90))',
  ST_Point(30,30))) As split;

split
-----
GEOMETRYCOLLECTION(
  LINESTRING(10 10,30 30),
  LINESTRING(30 30,190 190),
  LINESTRING(15 15,30 30),
  LINESTRING(30 30,100 90)
)

```

LineString split by a Point, where the point does not lie exactly on the line. Shows using **ST\_Snap** to snap the line to the point to allow it to be split.

```

WITH data AS (SELECT
  'LINESTRING(0 0, 100 100)::geometry AS line,
  'POINT(51 50):: geometry AS point
)
SELECT ST_AsText( ST_Split(line, point)) AS no_split,
  ST_AsText( ST_Split( ST_Snap(line, point, 1), point)) AS split

```

```
FROM data;
```

```
no_split
```

```
|
```

```
split
```

```
-----+-----
GEOMETRYCOLLECTION(LINESTRING(0 0,100 100)) | GEOMETRYCOLLECTION(LINESTRING(0 0,51 50), ↵
LINESTRING(51 50,100 100))
```

## See Also

[ST\\_Snap](#), [ST\\_Union](#)

## 8.13.7 ST\_Subdivide

`ST_Subdivide` — Computes a rectilinear subdivision of a geometry.

### Synopsis

```
setof geometry ST_Subdivide(geometry geom, integer max_vertices=256, float8 gridSize = -1);
```

### Description

Returns a set of geometries that are the result of dividing `geom` into parts using rectilinear lines, with each part containing no more than `max_vertices`.

`max_vertices` must be 5 or more, as 5 points are needed to represent a closed box. `gridSize` can be specified to have clipping work in fixed-precision space (requires GEOS-3.9.0+).

Point-in-polygon and other spatial operations are normally faster for indexed subdivided datasets. Since the bounding boxes for the parts usually cover a smaller area than the original geometry bbox, index queries produce fewer "hit" cases. The "hit" cases are faster because the spatial operations executed by the index recheck process fewer points.



#### Note

This is a [set-returning function](#) (SRF) that return a set of rows containing single geometry values. It can be used in a SELECT list or a FROM clause to produce a result set with one record for each result geometry.

Performed by the GEOS module.

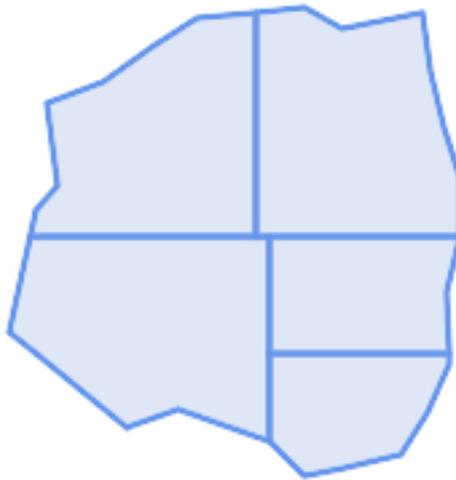
Availability: 2.2.0

Enhanced: 2.5.0 reuses existing points on polygon split, vertex count is lowered from 8 to 5.

Enhanced: 3.1.0 accept a `gridSize` parameter, requires GEOS  $\geq$  3.9.0 to use this new feature.

### Examples

**Example:** Subdivide a polygon into parts with no more than 10 vertices, and assign each part a unique id.



*Subdivided to maximum 10 vertices*

```
SELECT row_number() OVER() As rn, ST_AsText(geom) As wkt
FROM (SELECT ST_SubDivide(
  'POLYGON((132 10,119 23,85 35,68 29,66 28,49 42,32 56,22 64,32 110,40 119,36 150,
  57 158,75 171,92 182,114 184,132 186,146 178,176 184,179 162,184 141,190 122,
  190 100,185 79,186 56,186 52,178 34,168 18,147 13,132 10))'::geometry,10)) AS f( ↵
  geom);
```

```
rn          wkt
1          POLYGON((119 23,85 35,68 29,66 28,32 56,22 64,29.8260869565217 100,119 100,119 ↵
  23))
2          POLYGON((132 10,119 23,119 56,186 56,186 52,178 34,168 18,147 13,132 10))
3          POLYGON((119 56,119 100,190 100,185 79,186 56,119 56))
4          POLYGON((29.8260869565217 100,32 110,40 119,36 150,57 158,75 171,92 182,114 ↵
  184,114 100,29.8260869565217 100))
5          POLYGON((114 184,132 186,146 178,176 184,179 162,184 141,190 122,190 100,114 ↵
  100,114 184))
```

**Example:** Densify a long geography line using `ST_Segmentize(geography, distance)`, and use `ST_Subdivide` to split the resulting line into sublines of 8 vertices.



*The densified and split lines.*

```
SELECT ST_AsText( ST_Subdivide(
    ST_Segmentize('LINESTRING(0 0, 85 85)::geography,
        1200000)::geometry, 8));
```

```
LINESTRING(0 0,0.487578359029357 5.57659056746196,0.984542144675897 ↔
    11.1527721155093,1.50101059639722 16.7281035483571,1.94532113630331 21.25)
LINESTRING(1.94532113630331 21.25,2.04869538062779 22.3020741387339,2.64204641967673 ↔
    27.8740533545155,3.29994062412787 33.443216802941,4.04836719489742 ↔
    39.0084282520239,4.59890468420694 42.5)
LINESTRING(4.59890468420694 42.5,4.92498503922732 44.5680389206321,5.98737409390639 ↔
    50.1195229244701,7.3290919767674 55.6587646879025,8.79638749938413 60.1969505994924)
LINESTRING(8.79638749938413 60.1969505994924,9.11375579533779 ↔
    61.1785363177625,11.6558166691368 66.6648504160202,15.642041247655 ↔
    72.0867690601745,22.8716627200212 77.3609628116894,24.6991785131552 77.8939011989848)
LINESTRING(24.6991785131552 77.8939011989848,39.4046096622744 ↔
    82.1822848017636,44.7994523421035 82.5156766227011)
LINESTRING(44.7994523421035 82.5156766227011,85 85)
```

**Example:** Subdivide the complex geometries of a table in-place. The original geometry records are deleted from the source table, and new records for each subdivided result geometry are inserted.

```
WITH complex_areas_to_subdivide AS (
    DELETE from polygons_table
    WHERE ST_NPoints(geom) > 255
    RETURNING id, column1, column2, column3, geom
)
INSERT INTO polygons_table (fid, column1, column2, column3, geom)
SELECT fid, column1, column2, column3,
    ST_Subdivide(geom, 255) as geom
FROM complex_areas_to_subdivide;
```

**Example:** Create a new table containing subdivided geometries, retaining the key of the original geometry so that the new table can be joined to the source table. Since `ST_Subdivide` is a set-returning (table) function that returns a set of single-value rows, this syntax automatically produces a table with one row for each result part.

```
CREATE TABLE subdivided_geoms AS
SELECT pkey, ST_Subdivide(geom) AS geom
FROM original_geoms;
```

## See Also

[ST\\_ClipByBox2D](#), [ST\\_Segmentize](#), [ST\\_Split](#), [ST\\_NPoints](#)

## 8.13.8 ST\_SymDifference

`ST_SymDifference` — Computes a geometry representing the portions of geometries A and B that do not intersect.

### Synopsis

geometry `ST_SymDifference`(geometry geomA, geometry geomB, float8 gridSize = -1);

### Description

Returns a geometry representing the portions of geometries A and B that do not intersect. This is equivalent to `ST_Union(A, B) - ST_Intersection(A, B)`. It is called a symmetric difference because `ST_SymDifference(A, B) = ST_SymDifference(B, A)`.

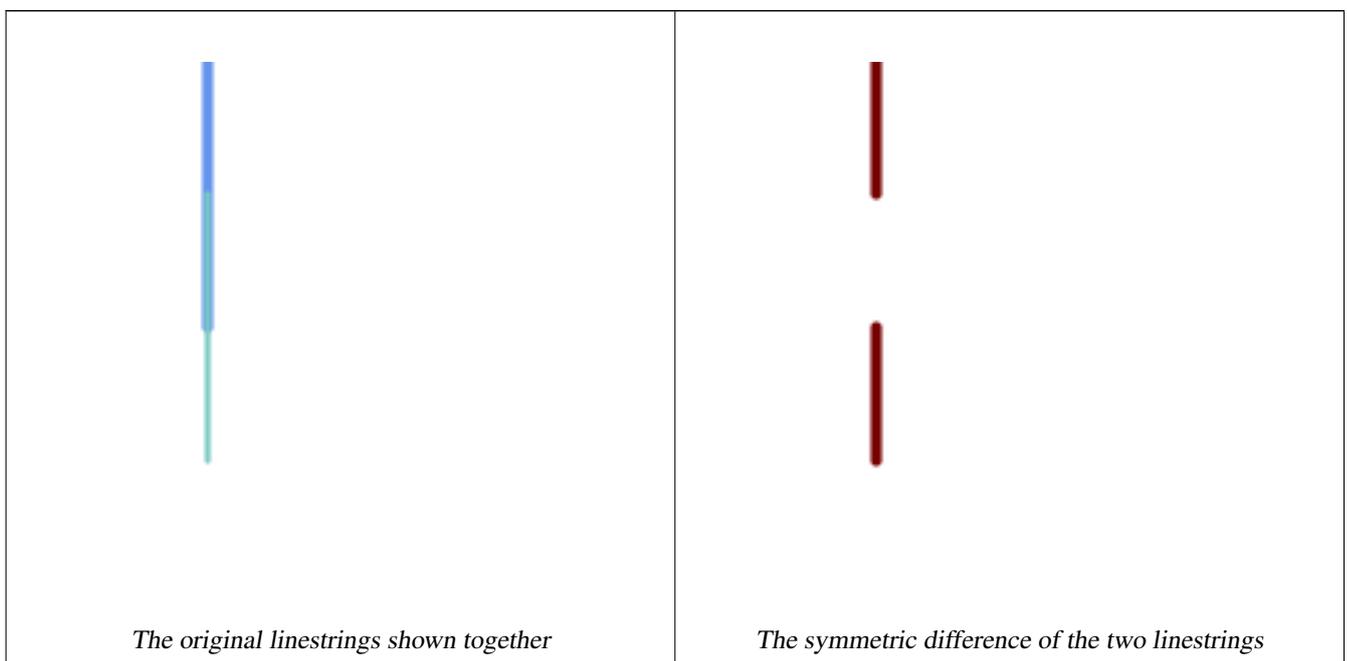
If the optional `gridSize` argument is provided, the inputs are snapped to a grid of the given size, and the result vertices are computed on that same grid. (Requires GEOS-3.9.0 or higher)

Performed by the GEOS module

Enhanced: 3.1.0 accept a `gridSize` parameter - requires GEOS  $\geq$  3.9.0

- ✔ This method implements the [OGC Simple Features Implementation Specification for SQL 1.1. s2.1.1.3](#)
- ✔ This method implements the SQL/MM specification. SQL-MM 3: 5.1.21
- ✔ This function supports 3d and will not drop the z-index. However, the result is computed using XY only. The result Z values are copied, averaged or interpolated.

## Examples



```
--Safe for 2d - symmetric difference of 2 linestrings
SELECT ST_AsText(
  ST_SymDifference(
    ST_GeomFromText('LINESTRING(50 100, 50 200)'),
    ST_GeomFromText('LINESTRING(50 50, 50 150)')
  )
);
```

```
st_astext
-----
MULTILINESTRING((50 150,50 200),(50 50,50 100))
```

```
--When used in 3d doesn't quite do the right thing
SELECT ST_AsEWKT(ST_SymDifference(ST_GeomFromEWKT('LINESTRING(1 2 1, 1 4 2)'),
  ST_GeomFromEWKT('LINESTRING(1 1 3, 1 3 4)')))
```

```
st_astext
-----
MULTILINESTRING((1 3 2.75,1 4 2),(1 1 3,1 2 2.25))
```

## See Also

[ST\\_Difference](#), [ST\\_Intersection](#), [ST\\_Union](#)

### 8.13.9 ST\_UnaryUnion

`ST_UnaryUnion` — Computes the union of the components of a single geometry.

#### Synopsis

```
geometry ST_UnaryUnion(geometry geom, float8 gridSize = -1);
```

#### Description

A single-input variant of [ST\\_Union](#). The input may be a single geometry, a `MultiGeometry`, or a `GeometryCollection`. The union is applied to the individual elements of the input.

This function can be used to fix `MultiPolygons` which are invalid due to overlapping components. However, the input components must each be valid. An invalid input component such as a bow-tie polygon may cause an error. For this reason it may be better to use [ST\\_MakeValid](#).

Another use of this function is to node and dissolve a collection of linestrings which cross or overlap to make them [simple](#). (To add nodes but not dissolve duplicate linework use [ST\\_Node](#).)

It is possible to combine `ST_UnaryUnion` with [ST\\_GeomCollFromText](#) to fine-tune how many geometries are be unioned at once. This allows trading off between memory usage and compute time, striking a balance between `ST_Union` and [ST\\_MemUnion](#).

If the optional `gridSize` argument is provided, the inputs are snapped to a grid of the given size, and the result vertices are computed on that same grid. (Requires GEOS-3.9.0 or higher)



This function supports 3d and will not drop the z-index. However, the result is computed using XY only. The result Z values are copied, averaged or interpolated.

Enhanced: 3.1.0 accept a `gridSize` parameter - requires GEOS >= 3.9.0

Availability: 2.0.0

## See Also

[ST\\_Union](#), [ST\\_MemUnion](#), [ST\\_MakeValid](#), [ST\\_GeomCollFromText](#), [ST\\_Node](#)

### 8.13.10 ST\_Union

`ST_Union` — Computes a geometry representing the point-set union of the input geometries.

#### Synopsis

```
geometry ST_Union(geometry g1, geometry g2);  
geometry ST_Union(geometry g1, geometry g2, float8 gridSize);  
geometry ST_Union(geometry[] g1_array);  
geometry ST_Union(geometry set g1field);  
geometry ST_Union(geometry set g1field, float8 gridSize);
```

## Description

Unions the input geometries, merging geometry to produce a result geometry with no overlaps. The output may be an atomic geometry, a MultiGeometry, or a Geometry Collection. Comes in several variants:

**Two-input variant:** returns a geometry that is the union of two input geometries. If either input is NULL, then NULL is returned.

**Array variant:** returns a geometry that is the union of an array of geometries.

**Aggregate variant:** returns a geometry that is the union of a rowset of geometries. The ST\_Union() function is an "aggregate" function in the terminology of PostgreSQL. That means that it operates on rows of data, in the same way the SUM() and AVG() functions do and like most aggregates, it also ignores NULL geometries.

See [ST\\_UnaryUnion](#) for a non-aggregate, single-input variant.

The ST\_Union array and set variants use the fast Cascaded Union algorithm described in <http://blog.cleverelephant.ca/2009/01/-must-faster-unions-in-postgis-14.html>

A `gridSize` can be specified to work in fixed-precision space. The inputs are snapped to a grid of the given size, and the result vertices are computed on that same grid. (Requires GEOS-3.9.0 or higher)



### Note

[ST\\_GeomCollFromText](#) may sometimes be used in place of ST\_Union, if the result is not required to be non-overlapping. ST\_Collect is usually faster than ST\_Union because it performs no processing on the collected geometries.

Performed by the GEOS module.

ST\_Union creates MultiLineString and does not sew LineStrings into a single LineString. Use [ST\\_LineMerge](#) to sew LineStrings.

NOTE: this function was formerly called `GeomUnion()`, which was renamed from "Union" because UNION is an SQL reserved word.

Enhanced: 3.1.0 accept a `gridSize` parameter - requires GEOS  $\geq$  3.9.0

Changed: 3.0.0 does not depend on SFCGAL.

Availability: 1.4.0 - ST\_Union was enhanced. ST\_Union(`geomarray`) was introduced and also faster aggregate collection in PostgreSQL.



This method implements the [OGC Simple Features Implementation Specification for SQL 1.1](#). s2.1.1.3



### Note

Aggregate version is not explicitly defined in OGC SPEC.



This method implements the SQL/MM specification. SQL-MM 3: 5.1.19 the z-index (elevation) when polygons are involved.



This function supports 3d and will not drop the z-index. However, the result is computed using XY only. The result Z values are copied, averaged or interpolated.

## Examples

### Aggregate example

```
SELECT id,
       ST_Union(geom) as singlegeom
FROM sometable f
GROUP BY id;
```

### Non-Aggregate example

```
select ST_AsText(ST_Union('POINT(1 2)' :: geometry, 'POINT(-2 3)' :: geometry))

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

select ST_AsText(ST_Union('POINT(1 2)' :: geometry, 'POINT(1 2)' :: geometry))

st_astext
-----
POINT(1 2)
```

### 3D example - sort of supports 3D (and with mixed dimensions!)

```
select ST_AsEWKT(ST_Union(geom))
from (
  select 'POLYGON((-7 4.2,-7.1 4.2,-7.1 4.3, -7 4.2))'::geometry geom
  union all
  select 'POINT(5 5 5)'::geometry geom
  union all
  select 'POINT(-2 3 1)'::geometry geom
  union all
  select 'LINESTRING(5 5 5, 10 10 10)'::geometry geom
) as foo;

st_asewkt
-----
GEOMETRYCOLLECTION(POINT(-2 3 1),LINESTRING(5 5 5,10 10 10),POLYGON((-7 4.2 5,-7.1 4.2 5,-7.1 4.3 5,-7 4.2 5)));
```

### 3d example not mixing dimensions

```
select ST_AsEWKT(ST_Union(geom))
from (
  select 'POLYGON((-7 4.2 2,-7.1 4.2 3,-7.1 4.3 2, -7 4.2 2))'::geometry geom
  union all
  select 'POINT(5 5 5)'::geometry geom
  union all
  select 'POINT(-2 3 1)'::geometry geom
  union all
  select 'LINESTRING(5 5 5, 10 10 10)'::geometry geom
) as foo;

st_asewkt
-----
GEOMETRYCOLLECTION(POINT(-2 3 1),LINESTRING(5 5 5,10 10 10),POLYGON((-7 4.2 2,-7.1 4.2 3,-7.1 4.3 2,-7 4.2 2)))

--Examples using new Array construct
SELECT ST_Union(ARRAY(SELECT geom FROM sometable));

SELECT ST_AsText(ST_Union(ARRAY[ST_GeomFromText('LINESTRING(1 2, 3 4)'),
  ST_GeomFromText('LINESTRING(3 4, 4 5)']))) As wktunion;

--wktunion---
MULTILINESTRING((3 4,4 5),(1 2,3 4))
```

**See Also**

[ST\\_GeomCollFromText](#), [ST\\_UnaryUnion](#), [ST\\_MemUnion](#), [ST\\_Intersection](#), [ST\\_Difference](#), [ST\\_SymDifference](#)

**8.14 `ST_Buffer`****8.14.1 `ST_Buffer`**

`ST_Buffer` — Computes a geometry covering all points within a given distance from a geometry.

**Synopsis**

```
geometry ST_Buffer(geometry g1, float radius_of_buffer, text buffer_style_parameters = '');
geometry ST_Buffer(geometry g1, float radius_of_buffer, integer num_seg_quarter_circle);
geography ST_Buffer(geography g1, float radius_of_buffer, text buffer_style_parameters);
geography ST_Buffer(geography g1, float radius_of_buffer, integer num_seg_quarter_circle);
```

**Description**

Computes a POLYGON or MULTIPOLYGON that represents all points whose distance from a geometry/geography is less than or equal to a given distance. A negative distance shrinks the geometry rather than expanding it. A negative distance may shrink a polygon completely, in which case POLYGON EMPTY is returned. For points and lines negative distances always return empty results.

For geometry, the distance is specified in the units of the Spatial Reference System of the geometry. For geography, the distance is specified in meters.

The optional third parameter controls the buffer accuracy and style. The accuracy of circular arcs in the buffer is specified as the number of line segments used to approximate a quarter circle (default is 8). The buffer style can be specified by providing a list of blank-separated key=value pairs as follows:

- `'quad_segs=#'` : number of line segments used to approximate a quarter circle (default is 8).
- `'endcap=round|flat|square'` : endcap style (defaults to "round"). `'butt'` is accepted as a synonym for `'flat'`.
- `'join=round|mitre|bevel'` : join style (defaults to "round"). `'miter'` is accepted as a synonym for `'mitre'`.
- `'mitre_limit=#.#'` : mitre ratio limit (only affects mitered join style). `'miter_limit'` is accepted as a synonym for `'mitre_limit'`.
- `'side=both|left|right'` : `'left'` or `'right'` performs a single-sided buffer on the geometry, with the buffered side relative to the direction of the line. This is only applicable to LINESTRING geometry and does not affect POINT or POLYGON geometries. By default end caps are square.

**Note**

For geography, this is a wrapper around the geometry implementation. It determines a planar spatial reference system that best fits the bounding box of the geography object (trying UTM, Lambert Azimuthal Equal Area (LAEA) North/South pole, and finally Mercator). The buffer is computed in the planar space, and then transformed back to WGS84. This may not produce the desired behavior if the input object is much larger than a UTM zone or crosses the dateline

**Note**

Buffer output is always a valid polygonal geometry. Buffer can handle invalid inputs, so buffering by distance 0 is sometimes used as a way of repairing invalid polygons. [ST\\_MakeValid](#) can also be used for this purpose.

**Note**

Buffering is sometimes used to perform a within-distance search. For this use case it is more efficient to use [ST\\_DWithin](#).

**Note**

This function ignores the Z dimension. It always gives a 2D result even when used on a 3D geometry.

Enhanced: 2.5.0 - `ST_Buffer` geometry support was enhanced to allow for side buffering specification `side=both|left|right`.

Availability: 1.5 - `ST_Buffer` was enhanced to support different endcaps and join types. These are useful for example to convert road linestrings into polygon roads with flat or square edges instead of rounded edges. Thin wrapper for geography was added.

GEOS [#baa8;#b4c8;#b85c; #c2e4;#d589](#);

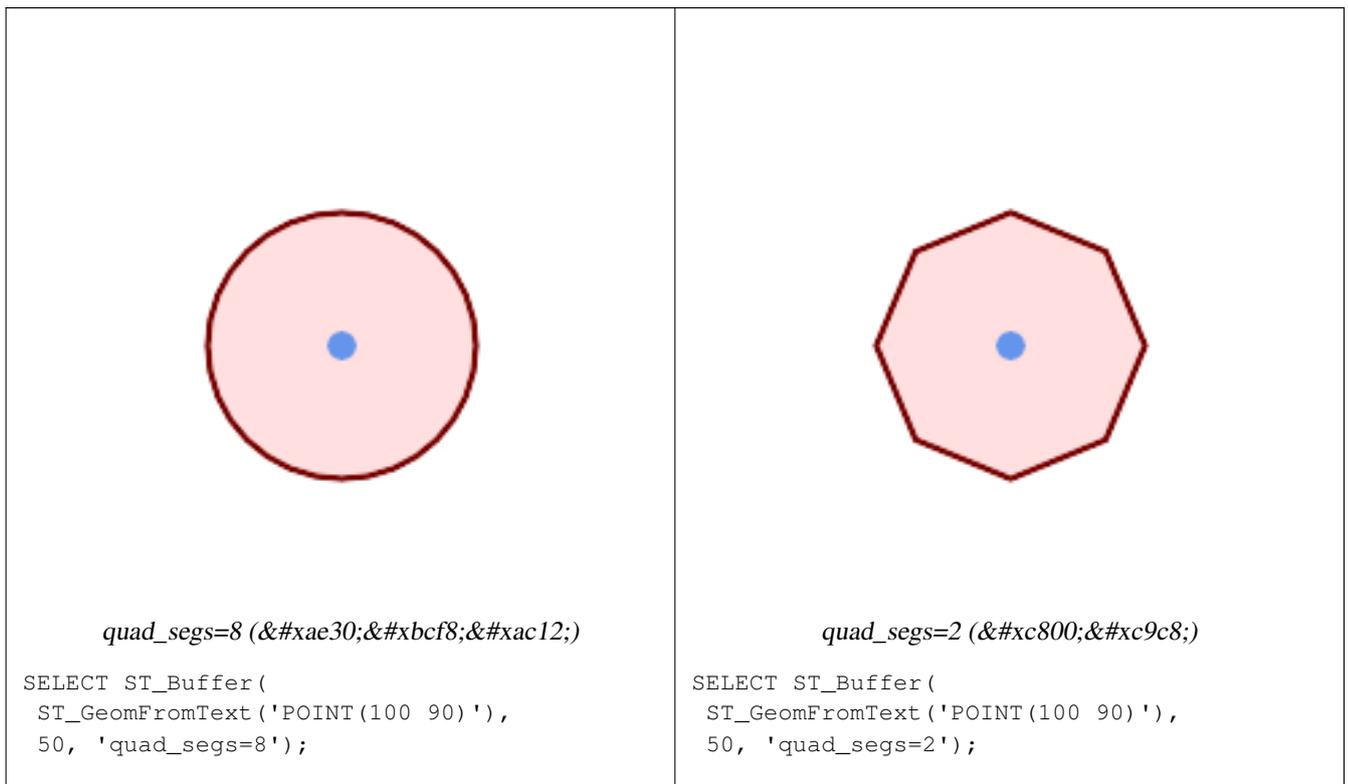


This method implements the [OGC Simple Features Implementation Specification for SQL 1.1. s2.1.1.3](#)

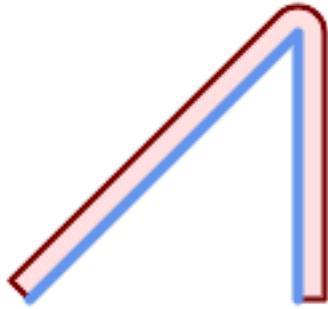


This method implements the SQL/MM specification. SQL-MM IEC 13249-3: 5.1.30

**⌘**



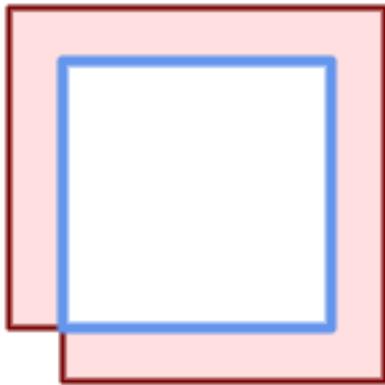


*side=left*

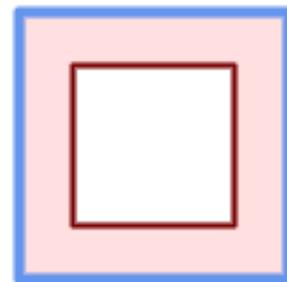
```
SELECT ST_Buffer(
  ST_GeomFromText(
    'LINESTRING(50 50,150 150,150 50)'
  ), 10, 'side=left');
```

*side=right*

```
SELECT ST_Buffer(
  ST_GeomFromText(
    'LINESTRING(50 50,150 150,150 50)'
  ), 10, 'side=right');
```

*right-hand-winding, polygon boundary side=left*

```
SELECT ST_Buffer(
  ST_ForceRHR(
    ST_Boundary(
      ST_GeomFromText(
        'POLYGON ((50 50, 50 150, 150 150, 150 50, 50 50))'
      )
    ), 20, 'side=left');
```

*right-hand-winding, polygon boundary side=right*

```
SELECT ST_Buffer(
  ST_ForceRHR(
    ST_Boundary(
      ST_GeomFromText(
        'POLYGON ((50 50, 50 150, 150 150, 150 50, 50 50))'
      )
    ), 20, 'side=right');
```

```
--A buffered point approximates a circle
-- A buffered point forcing approximation of (see diagram)
```

```

-- 2 points per quarter circle is poly with 8 sides (see diagram)
SELECT ST_NPoints(ST_Buffer(ST_GeomFromText('POINT(100 90)'), 50)) As ←
    promisingcircle_pcount,
ST_NPoints(ST_Buffer(ST_GeomFromText('POINT(100 90)'), 50, 2)) As lamecircle_pcount;

promisingcircle_pcount | lamecircle_pcount
-----+-----
                33 |                9

--A lighter but lamer circle
-- only 2 points per quarter circle is an octagon
--Below is a 100 meter octagon
-- Note coordinates are in NAD 83 long lat which we transform
to Mass state plane meter and then buffer to get measurements in meters;
SELECT ST_AsText(ST_Buffer(
ST_Transform(
ST_SetSRID(ST_Point(-71.063526, 42.35785), 4269), 26986)
,100,2)) As octagon;
-----
POLYGON((236057.59057465 900908.759918696,236028.301252769 900838.049240578,235
957.59057465 900808.759918696,235886.879896532 900838.049240578,235857.59057465
900908.759918696,235886.879896532 900979.470596815,235957.59057465 901008.759918
696,236028.301252769 900979.470596815,236057.59057465 900908.759918696))

```

**ST\_GeomCollFromText**

[ST\\_GeomCollFromText](#), [ST\\_DWithin](#), [ST\\_SetSRID](#), [ST\\_Transform](#), [ST\\_Union](#), [ST\\_MakeValid](#)

## 8.14.2 ST\_BuildArea

**ST\_BuildArea** — Creates a polygonal geometry formed by the linework of a geometry.

### Synopsis

geometry **ST\_BuildArea**(geometry geom);

**Note**

Creates an areal geometry formed by the constituent linework of the input geometry. The input can be **LINESTRINGS**, **MULTILINESTRINGS**, **POLYGONS**, **MULTIPOLYGONS**, and **GeometryCollections**. The result is a **Polygon** or **MultiPolygon**, depending on input. If the input linework does not form polygons, **NULL** is returned.

This function assumes all inner geometries represent holes



#### Note

Creates an areal geometry formed by the constituent linework of the input geometry. The input can be **LINESTRINGS**, **MULTILINESTRINGS**, **POLYGONS**, **MULTIPOLYGONS**, and **GeometryCollections**. The result is a **Polygon** or **MultiPolygon**, depending on input. If the input linework does not form polygons, **NULL** is returned.

1.1.0 **ST\_BuildArea**(geometry geom);

**ST\_BuildArea**



*These will create a donut*

```
--using polygons
SELECT ST_BuildArea(ST_Collect(smallc,bigc))
FROM (SELECT
  ST_Buffer(
    ST_GeomFromText('POINT(100 90)'), 25) As smallc,
  ST_Buffer(ST_GeomFromText('POINT(100 90)'), 50) As bigc) As foo;

--using linestrings
SELECT ST_BuildArea(ST_Collect(smallc,bigc))
FROM (SELECT
  ST_ExteriorRing(ST_Buffer(
    ST_GeomFromText('POINT(100 90)'), 25)) As smallc,
  ST_ExteriorRing(ST_Buffer(ST_GeomFromText('POINT(100 90)'), 50)) As bigc) As foo;
```

**;**

[ST\\_Node](#), [ST\\_MakePolygon](#), [ST\\_MakeValid](#), [ST\\_BdPolyFromText](#), [ST\\_BdMPolyFromText](#) (wrappers to this function with standard OGC interface)

### 8.14.3 ST\_Centroid

`ST_Centroid` — [ST\\_Centroid\(geometry g1\)](#)

#### Synopsis

geometry **ST\_Centroid**(geometry g1);  
 geography **ST\_Centroid**(geography g1, boolean use\_spheroid=true);

**;**

Computes a point which is the geometric center of mass of a geometry. For [MULTI]POINTS, the centroid is the arithmetic mean of the input coordinates. For [MULTI]LINESTRINGS, the centroid is computed using the weighted length of each line segment. For [MULTI]POLYGONS, the centroid is computed in terms of area. If an empty geometry is supplied, an empty

GEOMETRYCOLLECTION is returned. If NULL is supplied, NULL is returned. If CIRCULARSTRING or COMPOUNDCURVE are supplied, they are converted to linestring with CurveToLine first, then same than for LINESTRING

For mixed-dimension input, the result is equal to the centroid of the component Geometries of highest dimension (since the lower-dimension geometries contribute zero "weight" to the centroid).

Note that for polygonal geometries the centroid does not necessarily lie in the interior of the polygon. For example, see the diagram below of the centroid of a C-shaped polygon. To construct a point guaranteed to lie in the interior of a polygon use [ST\\_PointOnSurface](#).

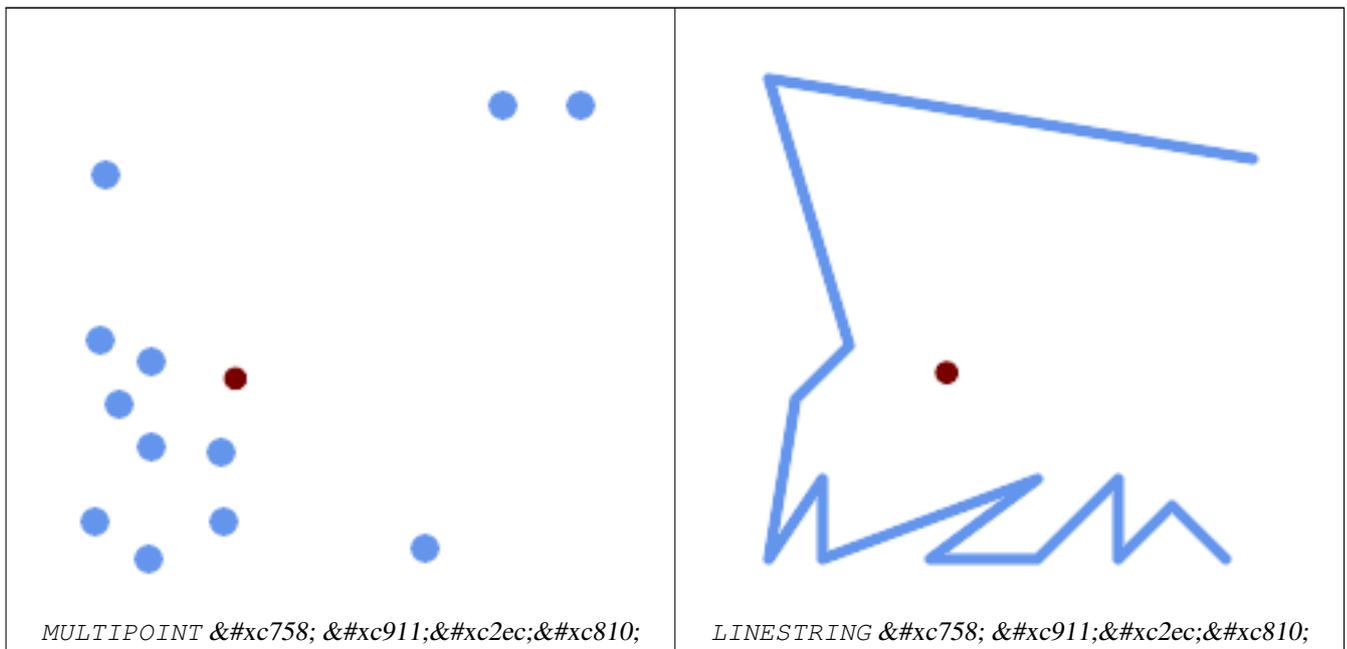
New in 2.3.0 : supports CIRCULARSTRING and COMPOUNDCURVE (using CurveToLine)

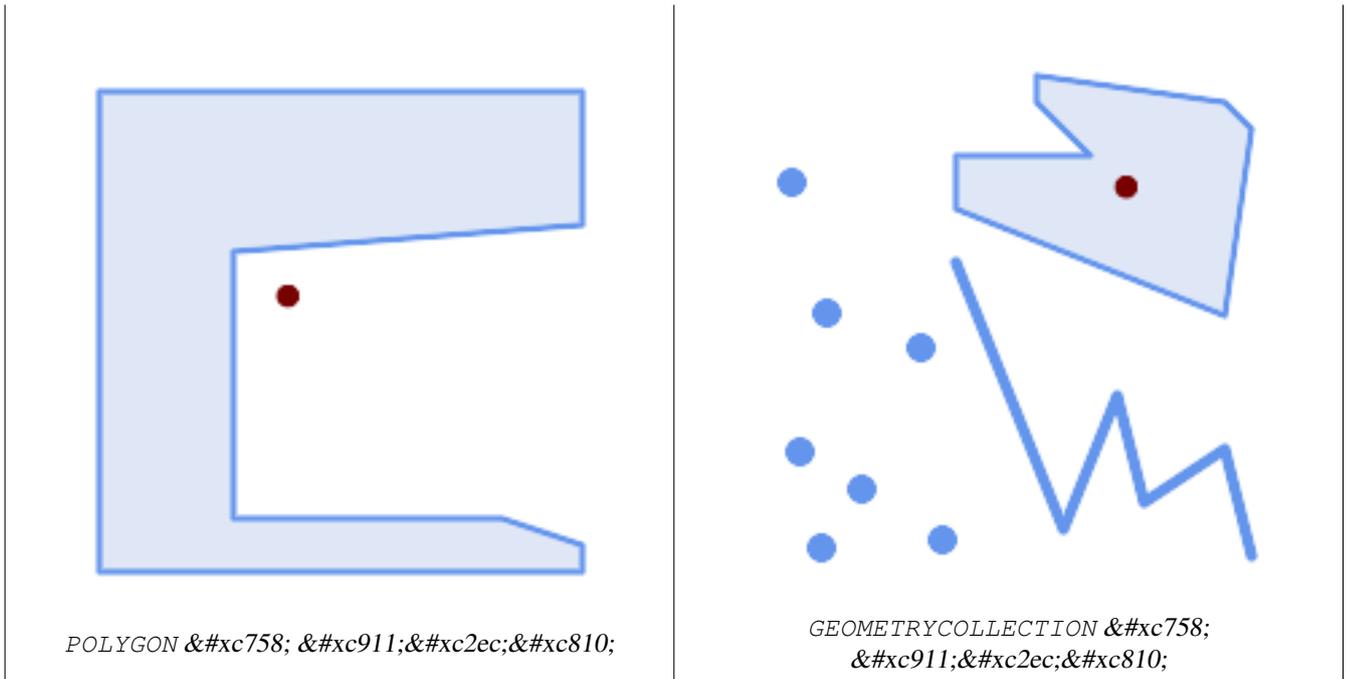
Availability: 2.4.0 support for geography was introduced.

- ✔ This method implements the [OGC Simple Features Implementation Specification for SQL 1.1](#).
- ✔ This method implements the SQL/MM specification. SQL-MM 3: 8.1.4, 9.5.5

### Centroid

In the following illustrations the red dot is the centroid of the source geometry.





```

SELECT ST_AsText(ST_Centroid('MULTIPOINT ( -1 0, -1 2, -1 3, -1 4, -1 7, 0 1, 0 3, 1 1, 2
0, 6 0, 7 8, 9 8, 10 6 )'));
      st_astext
-----
POINT(2.30769230769231 3.30769230769231)
(1 row)

SELECT ST_AsText(ST_Centroid(g))
FROM   ST_GeomFromText('CIRCULARSTRING(0 2, -1 1,0 0, 0.5 0, 1 0, 2 1, 1 2, 0.5 2, 0 2)')
      AS g ;
-----
POINT(0.5 1)

SELECT ST_AsText(ST_Centroid(g))
FROM   ST_GeomFromText('COMPOUNDCURVE(CIRCULARSTRING(0 2, -1 1,0 0),(0 0, 0.5 0, 1 0),
      CIRCULARSTRING( 1 0, 2 1, 1 2),(1 2, 0.5 2, 0 2))' ) AS g;
-----
POINT(0.5 1)

```

**ST\_Centroid**

[ST\\_PointOnSurface](#), [ST\\_GeometricMedian](#)

#### 8.14.4 ST\_ChaikinSmoothing

**ST\_ChaikinSmoothing** — Returns a smoothed version of a geometry, using the Chaikin algorithm

##### Synopsis

geometry **ST\_ChaikinSmoothing**(geometry geom, integer nIterations = 1, boolean preserveEndpoints = false);

**ChaikinSmoothing**

Returns a "smoothed" version of the given geometry using the Chaikin algorithm. See [Chaikins-Algorithm](#) for an explanation of the process. For each iteration the number of vertex points will double. The function puts new vertex points at 1/4 of the line before and after each point and removes the original point. To reduce the number of points use one of the simplification functions on the result. The new points gets interpolated values for all included dimensions, also z and m.

Second argument, number of iterations is limited to max 5 iterations

Note third argument is only valid for polygons, and will be ignored for linestrings

`ST_ChaikinSmoothing(geometry, integer, integer)`

**Note**

Note that returned geometry will get more points than the original. To reduce the number of points again use one of the simplification functions on the result. (see [ST\\_Simplify](#) and [ST\\_SimplifyVW](#))

Availability: 2.5.0

**ST\_Smooth**

A triangle is smoothed

```
select ST_AsText(ST_ChaikinSmoothing(geom)) smoothed
FROM (SELECT 'POLYGON((0 0, 8 8, 0 16, 0 0))'::geometry geom) As foo;
-- smoothed
POLYGON((0 0, 2 2, 6 6, 6 10, 2 14, 0 12, 0 4, 2 2))
```

**ST\_Simplify**

[ST\\_Simplify](#), [ST\\_SimplifyVW](#)

**8.14.5 ST\_ConcaveHull**

`ST_ConcaveHull` — Computes a possibly concave geometry that encloses all input geometry vertices

**Synopsis**

geometry `ST_ConcaveHull`(geometry param\_geom, float param\_pctconvex, boolean param\_allow\_holes = false);

**ConcaveHull**

A concave hull of a geometry is a possibly concave geometry that encloses the vertices of the input geometry. In the general case the concave hull is a Polygon. The polygon will not contain holes unless the optional `param_allow_holes` argument is specified as true. The concave hull of two or more collinear points is a two-point LineString. The concave hull of one or more identical points is a Point.

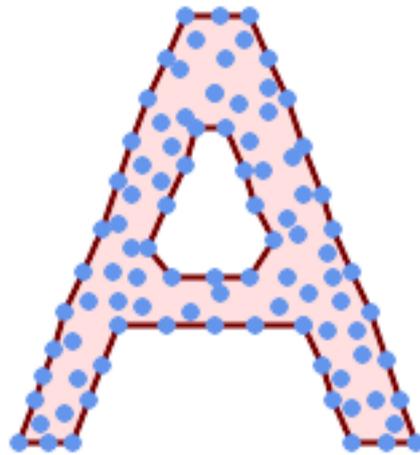
One can think of a concave hull as "shrink-wrapping" a set of points. This is different to the [convex hull](#), which is more like wrapping a rubber band around the points. The concave hull generally has a smaller area and represents a more natural boundary



```

    , (150 146), (38 136), (111 170), (124 94), (26 59), (60 41), (71 162), (41 64), ←
    (88 110), (122 34), (151 97), (157 56), (39 146), (88 33), (159 45), (47 56), ←
    (138 40), (129 165), (33 48), (106 31), (169 147), (37 122), (71 109), (163 89), ←
    (37 156), (82 170), (180 72), (29 142), (46 41), (59 155), (124 106), (157 80), ←
    (175 82), (56 50), (62 116), (113 95), (144 167))',
    0.1 ) );
---st_astext--
POLYGON ((18 142, 21 151, 27 160, 35 167, 44 172, 54 175, 64 178, 74 180, 84 181, 94 181, ←
104 181, 114 181, 124 181, 134 179, 144 177, 153 173, 162 168, 171 162, 177 154, 182 ←
145, 184 135, 173 134, 161 134, 150 133, 139 132, 136 142, 128 149, 119 153, 109 155, 99 ←
155, 89 155, 79 153, 69 150, 61 144, 63 134, 72 128, 82 125, 92 123, 102 121, 112 119, ←
122 118, 132 116, 142 113, 151 110, 161 106, 170 102, 178 96, 185 88, 189 78, 190 68, ←
189 58, 185 49, 179 41, 171 34, 162 29, 153 25, 143 23, 133 21, 123 19, 113 19, 102 19, ←
92 19, 82 19, 72 21, 62 22, 52 25, 43 29, 33 34, 25 41, 19 49, 14 58, 10 72, 21 73, 31 ←
74, 42 74, 53 76, 56 66, 63 58, 71 51, 81 48, 91 46, 101 45, 111 46, 121 47, 131 50, 140 ←
55, 145 64, 144 74, 135 80, 125 83, 115 85, 105 87, 95 89, 85 91, 75 93, 65 95, 55 98, ←
45 102, 37 107, 29 114, 22 122, 19 132, 18 142))

```



*Concave Hull of a MultiPoint, allowing holes*

```

SELECT ST_AsText( ST_ConcaveHull(
    'MULTIPOINT ((132 64), (114 64), (99 64), (81 64), (63 64), (57 49), (52 36), (46 ←
    20), (37 20), (26 20), (32 36), (39 55), (43 69), (50 84), (57 100), (63 118), ←
    (68 133), (74 149), (81 164), (88 180), (101 180), (112 180), (119 164), (126 ←
    149), (132 131), (139 113), (143 100), (150 84), (157 69), (163 51), (168 36), ←
    (174 20), (163 20), (150 20), (143 36), (139 49), (132 64), (99 151), (92 138), ←
    (88 124), (81 109), (74 93), (70 82), (83 82), (99 82), (112 82), (126 82), (121 ←
    96), (114 109), (110 122), (103 138), (99 151), (34 27), (43 31), (48 44), (46 ←
    58), (52 73), (63 73), (61 84), (72 71), (90 69), (101 76), (123 71), (141 62), ←
    (166 27), (150 33), (159 36), (146 44), (154 53), (152 62), (146 73), (134 76), ←
    (143 82), (141 91), (130 98), (126 104), (132 113), (128 127), (117 122), (112 ←
    133), (119 144), (108 147), (119 153), (110 171), (103 164), (92 171), (86 160), ←
    (88 142), (79 140), (72 124), (83 131), (79 118), (68 113), (63 102), (68 93), ←
    (35 45))',
    0.15, true ) );
---st_astext--
POLYGON ((43 69, 50 84, 57 100, 63 118, 68 133, 74 149, 81 164, 88 180, 101 180, 112 180, ←
119 164, 126 149, 132 131, 139 113, 143 100, 150 84, 157 69, 163 51, 168 36, 174 20, 163 ←
20, 150 20, 143 36, 139 49, 132 64, 114 64, 99 64, 81 64, 63 64, 57 49, 52 36, 46 20, ←
37 20, 26 20, 32 36, 35 45, 39 55, 43 69), (88 124, 81 109, 74 93, 83 82, 99 82, 112 82, ←
121 96, 114 109, 110 122, 103 138, 92 138, 88 124))

```

Using with `ST_Collect` to compute the concave hull of a geometry set.

```
-- Compute estimate of infected area based on point observations
SELECT disease_type,
       ST_ConcaveHull( ST_Collect(obs_pnt), 0.3 ) AS geom
FROM disease_obs
GROUP BY disease_type;
```

**See also:**

[ST\\_ConvexHull](#), [ST\\_GeomCollFromText](#), [ST\\_AlphaShape](#), [ST\\_OptimalAlphaShape](#)

### 8.14.6 ST\_ConvexHull

`ST_ConvexHull` — Computes the convex hull of a geometry.

#### Synopsis

geometry `ST_ConvexHull`(geometry geomA);

**See also:**

Computes the convex hull of a geometry. The convex hull is the smallest convex geometry that encloses all geometries in the input.

One can think of the convex hull as the geometry obtained by wrapping an rubber band around a set of geometries. This is different from a **concave hull** which is analogous to "shrink-wrapping" the geometries. A convex hull is often used to determine an affected area based on a set of point observations.

In the general case the convex hull is a Polygon. The convex hull of two or more collinear points is a two-point LineString. The convex hull of one or more identical points is a Point.

This is not an aggregate function. To compute the convex hull of a set of geometries, use [ST\\_GeomCollFromText](#) to aggregate them into a geometry collection (e.g. `ST_ConvexHull(ST_Collect(geom))`).

GEOS [See also:](#) [ST\\_ConvexHull](#); [ST\\_ConcaveHull](#); [ST\\_GeomCollFromText](#);



This method implements the [OGC Simple Features Implementation Specification for SQL 1.1](#). s2.1.1.3

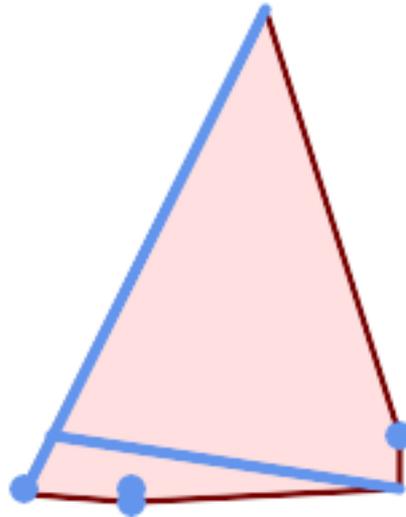


This method implements the SQL/MM specification. SQL-MM IEC 13249-3: 5.1.16



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

**Convex Hull**



*Convex Hull of a MultiLineString and a MultiPoint*

```
SELECT ST_AsText(ST_ConvexHull(
  ST_Collect(
    ST_GeomFromText('MULTILINESTRING((100 190,10 8),(150 10, 20 30)'),
    ST_GeomFromText('MULTIPOINT(50 5, 150 30, 50 10, 10 10)')
  )));
---st_astext--
POLYGON((50 5,10 8,10 10,100 190,150 30,150 10,50 5))
```

Using with `ST_Collect` to compute the convex hulls of geometry sets.

```
--Get estimate of infected area based on point observations
SELECT d.disease_type,
  ST_ConvexHull(ST_Collect(d.geom)) As geom
FROM disease_obs As d
GROUP BY d.disease_type;
```

**ST\_GeomCollFromText**

[ST\\_GeomCollFromText](#), [ST\\_ConcaveHull](#), [ST\\_MinimumBoundingCircle](#)

### 8.14.7 ST\_DelaunayTriangles

`ST_DelaunayTriangles` — Returns the Delaunay triangulation of the vertices of a geometry.

#### Synopsis

geometry `ST_DelaunayTriangles`(geometry g1, float tolerance, int4 flags);

**Return Value**

Return the **Delaunay triangulation** of the vertices of the input geometry. Output is a `COLLECTION` of polygons (for flags=0) or a `MULTILINESTRING` (for flags=1) or `TIN` (for flags=2). The tolerance, if any, is used to snap input vertices together.

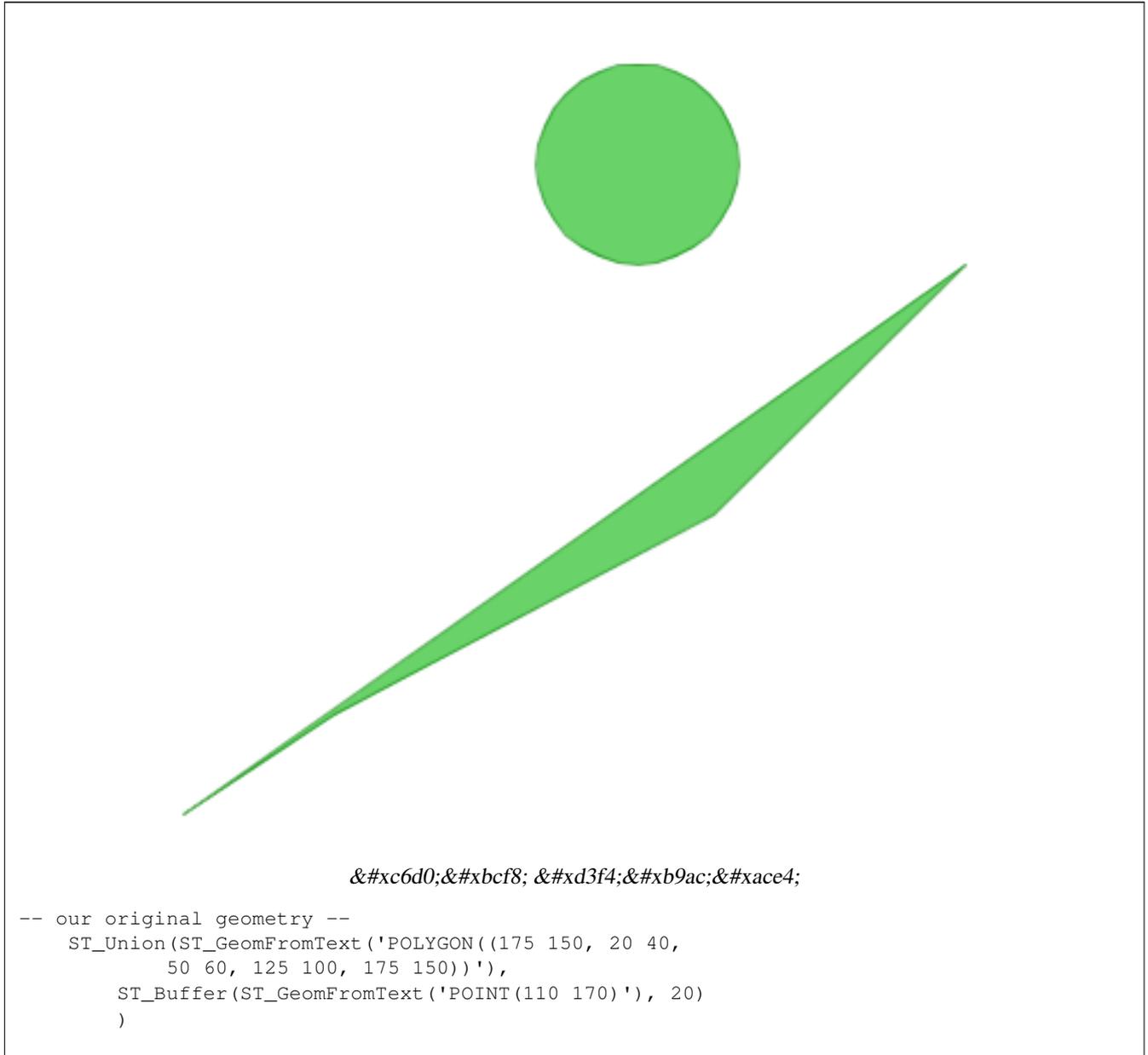
GEOS

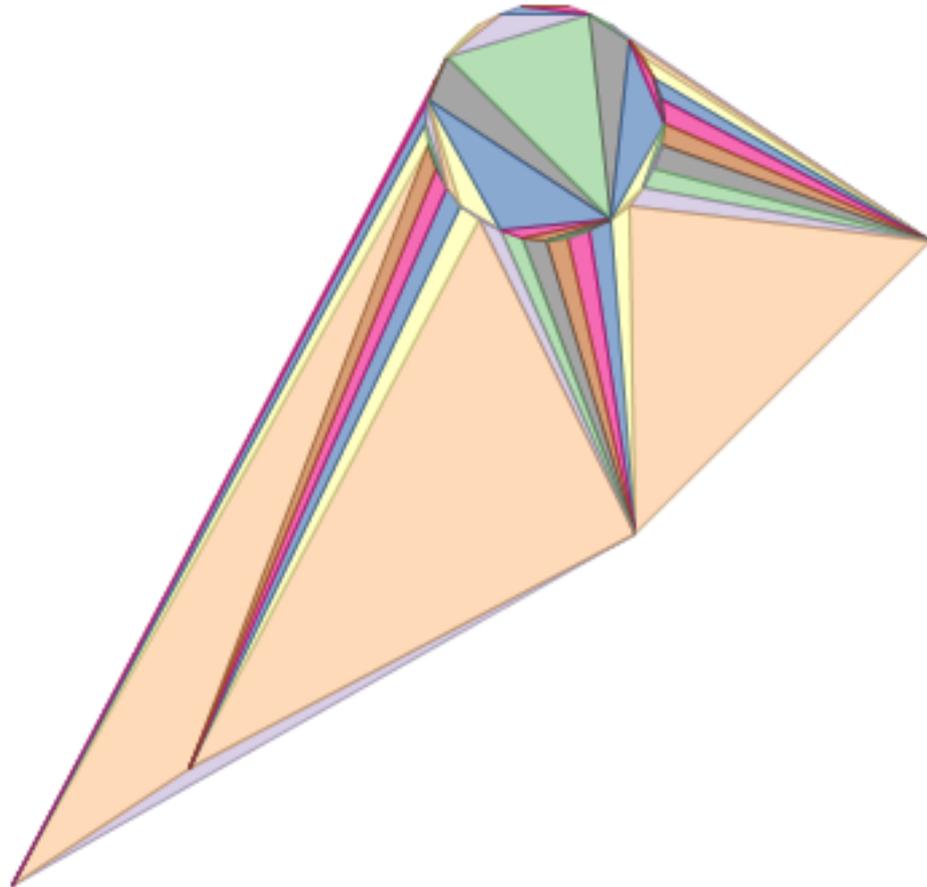
2.1.0

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

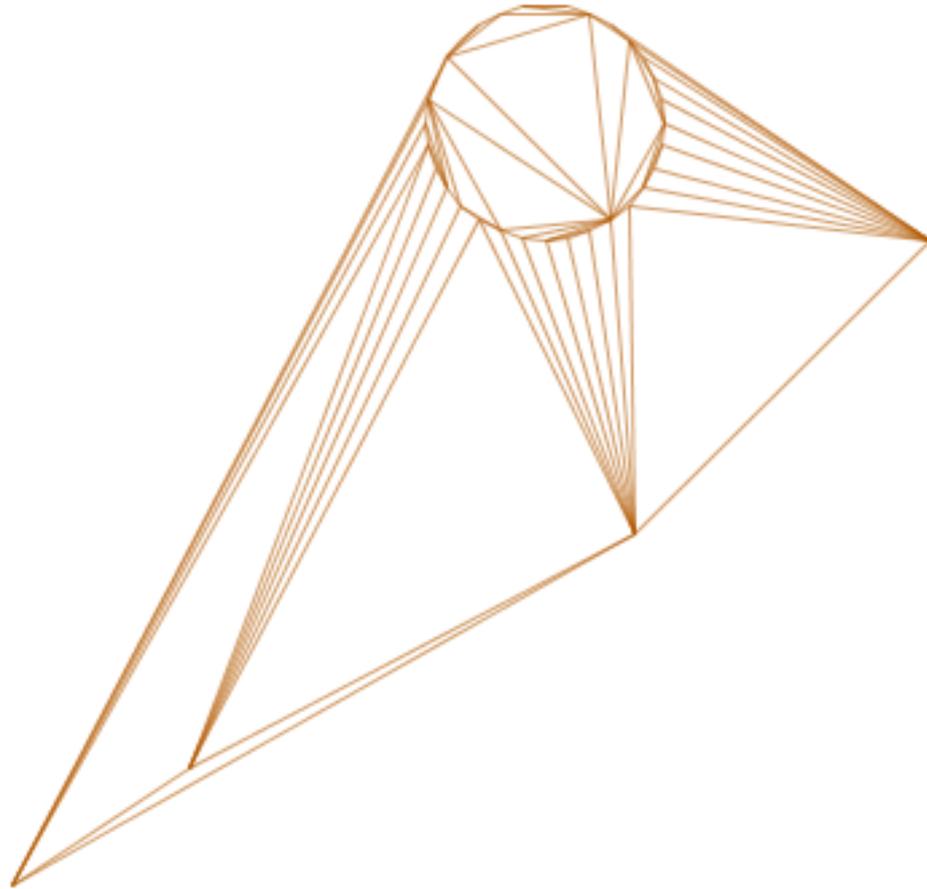
✔ This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).

**2**





```
ST_DelaunayTriangles(  
  ST_Union(ST_GeomFromText('POLYGON((175 150, 20 40,  
    50 60, 125 100, 175 150))'),  
  ST_Buffer(ST_GeomFromText('POINT(110 170)'), 20)  
))  
As dtriag;
```



```
-- &#xc0ac;&#xc6a9;&#xc790; &#xc9c0;&#xce68;&#xc11c;  
    &#xba40;&#xd2f0;&#xb77c;&#xc778;&#xc2a4;&#xd2b8;&#xb9c1;&#xc778; &#xb4e4;&#xb85c;&#xb124;  
    &#xc0bc;&#xac01;&#xd615;
```

```
SELECT  
  ST_DelaunayTriangles(  
    ST_Union(ST_GeomFromText('POLYGON((175 150, 20 40,  
      50 60, 125 100, 175 150))'),  
    ST_Buffer(ST_GeomFromText('POINT(110 170)'), 20)  
  ), 0.001, 1)  
  As dtriag;
```



## 8.14.8 ST\_FilterByM

ST\_FilterByM — Removes vertices based on their M value

### Synopsis

geometry **ST\_FilterByM**(geometry geom, double precision min, double precision max = null, boolean returnM = false);

### Notes

Filters out vertex points based on their M-value. Returns a geometry with only vertex points that have a M-value larger or equal to the min value and smaller or equal to the max value. If max-value argument is left out only min value is considered. If fourth argument is left out the m-value will not be in the resulting geometry. If resulting geometry have too few vertex points left for its geometry type an empty geometry will be returned. In a geometry collection geometries without enough points will just be left out silently.

This function is mainly intended to be used in conjunction with ST\_SetEffectiveArea. ST\_EffectiveArea sets the effective area of a vertex in its m-value. With ST\_FilterByM it then is possible to get a simplified version of the geometry without any calculations, just by filtering



#### Note

There is a difference in what ST\_SimplifyVW returns when not enough points meet the criteria compared to ST\_FilterByM. ST\_SimplifyVW returns the geometry with enough points while ST\_FilterByM returns an empty geometry



#### Note

Note that the returned geometry might be invalid



#### Note

This function returns all dimensions, including the Z and M values

Availability: 2.5.0

### Examples

A linestring is filtered

```
SELECT ST_AsText(ST_FilterByM(geom,30)) simplified
FROM (SELECT ST_SetEffectiveArea('LINESTRING(5 2, 3 8, 6 20, 7 25, 10 10)::geometry) geom ←
) As foo;
-result
      simplified
-----
LINESTRING(5 2,7 25,10 10)
```

### See also

[ST\\_SetEffectiveArea](#), [ST\\_SimplifyVW](#)

### 8.14.9 ST\_GeneratePoints

ST\_GeneratePoints — Generates random points contained in a Polygon or MultiPolygon.

#### Synopsis

```
geometry ST_GeneratePoints( g geometry , npoints integer );
geometry ST_GeneratePoints( geometry g , integer npoints , integer seed );
```

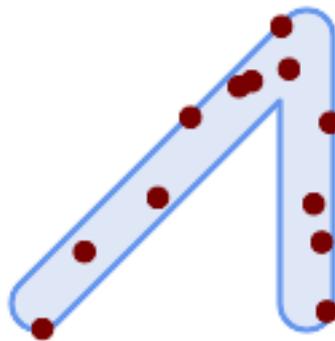
#### Parameters

ST\_GeneratePoints generates a given number of pseudo-random points which lie within the input area. The optional *seed* is used to regenerate a deterministic sequence of points, and must be greater than zero.

2.3.0 Added *seed* parameter.

Enhanced: 3.0.0, added *seed* parameter

#### Examples



*Generated 12 Points overlaid on top of original polygon using a random seed value 1996*

```
SELECT ST_GeneratePoints(geom, 12, 1996)
FROM (
  SELECT ST_Buffer(
    ST_GeomFromText(
      'LINESTRING(50 50,150 150,150 50)'),
    10, 'endcap=round join=round') AS geom
) AS s;
```

### 8.14.10 ST\_GeometricMedian

ST\_GeometricMedian — Returns the geometric median of a set of points. The geometric median is the point that minimizes the sum of the distances to all the points in the set.

## Synopsis

geometry **ST\_GeometricMedian** ( geometry geom, float8 tolerance = NULL, int max\_iter = 10000, boolean fail\_if\_not\_converged = false);

### Details

Computes the approximate geometric median of a MultiPoint geometry using the Weiszfeld algorithm. The geometric median is the point minimizing the sum of distances to the input points. It provides a centrality measure that is less sensitive to outlier points than the centroid (center of mass).

The algorithm iterates until the distance change between successive iterations is less than the supplied `tolerance` parameter. If this condition has not been met after `max_iterations` iterations, the function produces an error and exits, unless `fail_if_not_converged` is set to `false` (the default).

If a `tolerance` argument is not provided, the tolerance value is calculated based on the extent of the input geometry.

If present, the input point M values are interpreted as their relative weights.

2.3.0 Enhanced: 2.5.0 Added support for M as weight of points.

Enhanced: 2.5.0 Added support for M as weight of points.

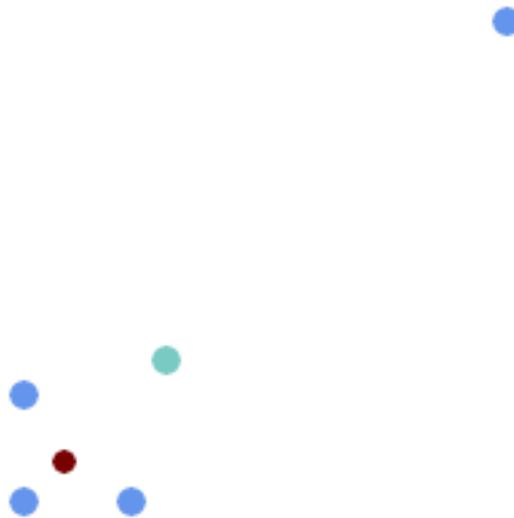


This function supports 3d and will not drop the z-index.



This function supports M coordinates.

### Examples



*Comparison of the geometric median (red) and centroid (turquoise) of a MultiPoint.*

```
WITH test AS (
SELECT 'MULTIPOINT((10 10), (10 40), (40 10), (190 190))'::geometry geom)
SELECT
  ST_AsText(ST_Centroid(geom)) centroid,
  ST_AsText(ST_GeometricMedian(geom)) median
FROM test;
```

| centroid         | median                                     |
|------------------|--|
| POINT(62.5 62.5) | POINT(25.01778421249728 25.01778421249728) |

(1 row)

---

**ST\_Centroid**

**ST\_Centroid**

### 8.14.11 ST\_LineMerge

ST\_LineMerge — Return the lines formed by sewing together a MultiLineString.

#### Synopsis

```
geometry ST_LineMerge(geometry amultilinestring);  
geometry ST_LineMerge(geometry amultilinestring, boolean directed);
```

**ST\_LineMerge**

Returns a LineString or MultiLineString formed by joining together the line elements of a MultiLineString. Lines are joined at their endpoints at 2-way intersections. Lines are not joined across intersections of 3-way or greater degree.

If **directed** is TRUE, then ST\_LineMerge will not change point order within LineStrings, so lines with opposite directions will not be merged



#### Note

Only use with MultiLineString/LineStrings. Other geometry types return an empty GeometryCollection

---

GEOS 3.10.0-3.11.0

Enhanced: 3.3.0 accept a directed parameter - requires GEOS >= 3.11.0

1.1.0-3.11.0

---

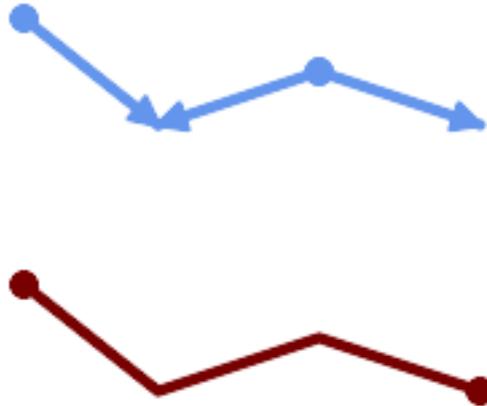


#### Warning

This function strips the M dimension.

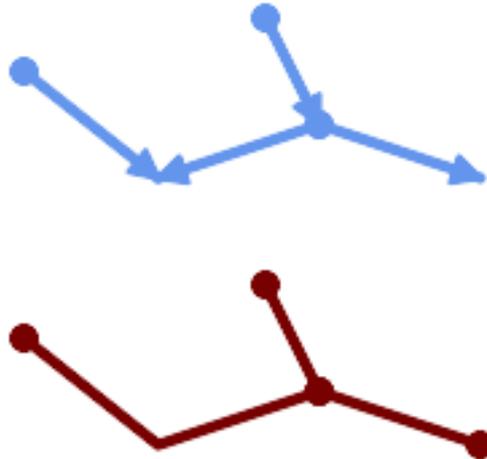
---

&#xc608;&#xc2dc;



*Merging lines with different orientation.*

```
SELECT ST_AsText(ST_LineMerge(
'MULTILINESTRING((10 160, 60 120), (120 140, 60 120), (120 140, 180 120))'
));
-----
LINESTRING(10 160,60 120,120 140,180 120)
```

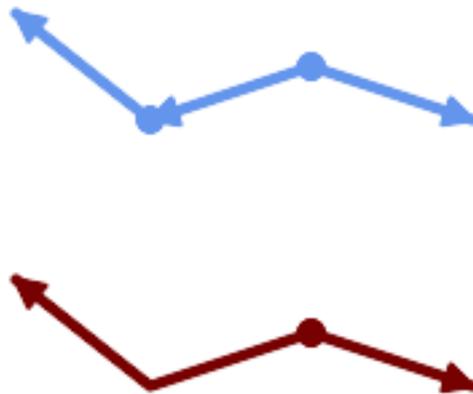


*Lines are not merged across intersections with degree > 2.*

```
SELECT ST_AsText(ST_LineMerge(
'MULTILINESTRING((10 160, 60 120), (120 140, 60 120), (120 140, 180 120), (100 180, 120
140))'
));
-----
MULTILINESTRING((10 160,60 120,120 140),(100 180,120 140),(120 140,180 120))
```

If merging is not possible due to non-touching lines, the original MultiLineString is returned.

```
SELECT ST_AsText(ST_LineMerge(
'MULTILINESTRING((-29 -27,-30 -29.7,-36 -31,-45 -33), (-45.2 -33.2,-46 -32))'
));
-----
MULTILINESTRING((-45.2 -33.2,-46 -32), (-29 -27,-30 -29.7,-36 -31,-45 -33))
```



*Lines with opposite directions are not merged if directed = TRUE.*

```
SELECT ST_AsText(ST_LineMerge(
'MULTILINESTRING((60 30, 10 70), (120 50, 60 30), (120 50, 180 30))',
TRUE));
-----
MULTILINESTRING((120 50,60 30,10 70), (120 50,180 30))
```

Example showing Z-dimension handling.

```
SELECT ST_AsText(ST_LineMerge(
'MULTILINESTRING((-29 -27 11,-30 -29.7 10,-36 -31 5,-45 -33 6), (-29 -27 12,-30 -29.7 ←
5), (-45 -33 1,-46 -32 11))'
));
-----
LINESTRING Z (-30 -29.7 5,-29 -27 11,-30 -29.7 10,-36 -31 5,-45 -33 1,-46 -32 11)
```

**&#xccc38;&#xace0;**

[ST\\_Segmentize](#), [ST\\_LineSubstring](#)

### 8.14.12 ST\_MaximumInscribedCircle

`ST_MaximumInscribedCircle` — [&#xb3c4;&#xd615;&#xc758;](#) [&#xae30;&#xd558;&#xd559;&#xc801;](#) [&#xc911;&#xc2ec;&#xc744;](#) [&#xbc18;&#xd658;&#xd569;&#xb2c8;&#xb2e4;](#).

#### Synopsis

(geometry, geometry, double precision) `ST_MaximumInscribedCircle`(geometry geom);

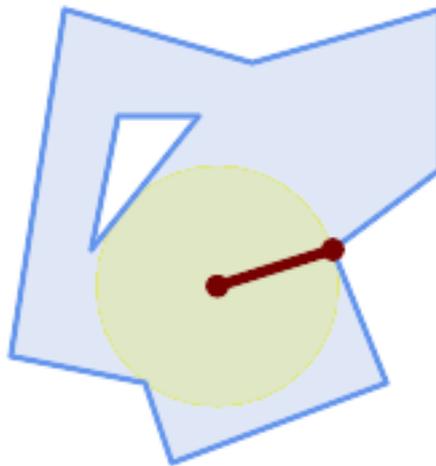
**ST\_MaximumInscribedCircle**

Finds the largest circle that is contained within a (multi)polygon, or which does not overlap any lines and points. Returns a record with fields:

- `center` - center point of the circle
- `nearest` - a point on the geometry nearest to the center
- `radius` - radius of the circle

For polygonal inputs, the circle is inscribed within the boundary rings, using the internal rings as boundaries. For linear and point inputs, the circle is inscribed within the convex hull of the input, using the input lines and points as further boundaries.

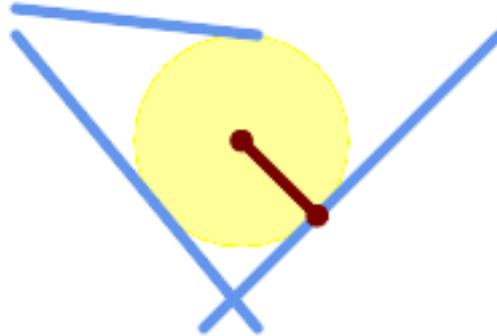
Availability: 3.1.0 - requires GEOS >= 3.9.0.

**ST\_MinimumBoundingCircle****ST\_MinimumBoundingCircle****ST\_MinimumBoundingCircle**

*Maximum inscribed circle of a polygon. Center, nearest point, and radius are returned.*

```
SELECT radius, ST_AsText(center) AS center, ST_AsText(nearest) AS nearest
FROM ST_MaximumInscribedCircle(
  'POLYGON ((40 180, 110 160, 180 180, 180 120, 140 90, 160 40, 80 10, 70 40, 20 50, ←
    40 180),
    (60 140, 50 90, 90 140, 60 140))');
```

| radius          | center                     | nearest       |
|-----------------|----------------------------|---------------|
| 45.165845650018 | POINT(96.953125 76.328125) | POINT(140 90) |



Maximum inscribed circle of a multi-linestring. Center, nearest point, and radius are returned.

`ST_MinimumBoundingRadius`

`ST_MinimumBoundingRadius`

### 8.14.13 ST\_MinimumBoundingCircle

`ST_MinimumBoundingCircle` — Returns the smallest circle polygon that contains a geometry.

#### Synopsis

geometry `ST_MinimumBoundingCircle`(geometry geomA, integer num\_segs\_per\_qt\_circ=48);

`ST_MinimumBoundingCircle`

Returns the smallest circle polygon that contains a geometry.

---

#### Note

`ST_MinimumBoundingCircle` returns the smallest circle polygon that contains a geometry. The circle is defined by its center point and radius. The center point is the point on the geometry that is closest to the center of the circle. The radius is the distance from the center point to the circle's edge. The number of segments per quarter circle (num\_segs\_per\_qt\_circ) is an optional parameter that defaults to 48. A higher value results in a smoother circle but at the cost of performance.



This function is not an aggregate. It can be used with `ST_GeomCollFromText` to get the minimum bounding circle of a set of geometries.

`ST_MinimumBoundingCircle` (geometry) geometry

GEOS 3.10.0

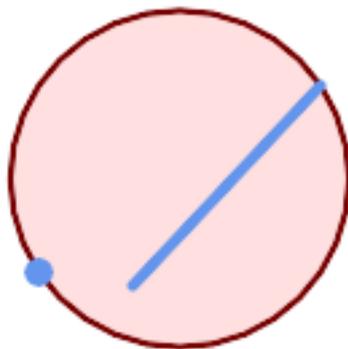
1.4.0

**ST\_MinimumBoundingCircle**

`ST_MinimumBoundingCircle`, `ST_MinimumBoundingRadius`

**ST\_MinimumBoundingCircle**

```
SELECT d.disease_type,
       ST_MinimumBoundingCircle(ST_Collect(d.geom)) As geom
FROM disease_obs As d
GROUP BY d.disease_type;
```



`ST_MinimumBoundingCircle` (geometry) geometry

```
SELECT ST_AsText(ST_MinimumBoundingCircle(
  ST_Collect(
    ST_GeomFromText('LINESTRING(55 75,125 150)'),
    ST_Point(20, 80)), 8
  )) As wktmbc;
```

wktmbc

```
-----
POLYGON((135.59714732062 115,134.384753327498 102.690357210921,130.79416296937 90.8537670908995,124.963360620072 79.9451031602111,117.116420743937 70.3835792560632,107.554896839789 62.5366393799277,96.6462329091006 56.70583703063,84.8096427890789 53.115246672502,72.5000000000001 51.9028526793802,60.1903572109213 53.1152466725019,48.3537670908996 56.7058370306299,37.4451031602112 62.5366393799276,27.8835792560632 90.8537670908995)↔
```

```

70.383579256063,20.0366393799278 79.9451031602109,14.20583703063 ←
90.8537670908993,10.615246672502 102.690357210921,9.40285267938019 115,10.6152466725019 ↔
127.309642789079,14.2058370306299 139.1462329091,20.0366393799275 ←
150.054896839789,27.883579256063 159.616420743937,
37.4451031602108 167.463360620072,48.3537670908992 173.29416296937,60.190357210921 ↔
176.884753327498,
72.4999999999998 178.09714732062,84.8096427890786 176.884753327498,96.6462329091003 ↔
173.29416296937,107.554896839789 167.463360620072,
117.116420743937 159.616420743937,124.963360620072 150.054896839789,130.79416296937 ↔
139.146232909101,134.384753327498 127.309642789079,135.59714732062 115))

```

**ST\_MinimumBoundingRadius**

[ST\\_GeomCollFromText](#), [ST\\_MinimumBoundingRadius](#)

### 8.14.14 ST\_MinimumBoundingRadius

**ST\_MinimumBoundingRadius** — Returns the center point and radius of the smallest circle that contains a geometry.

#### Synopsis

(geometry, double precision) **ST\_MinimumBoundingRadius**(geometry geom);

**ST\_MinimumBoundingRadius**

Computes the center point and radius of the smallest circle that contains a geometry. Returns a record with fields:

- center - center point of the circle
- radius - radius of the circle

Use in conjunction with [ST\\_GeomCollFromText](#) to get the minimum bounding circle of a set of geometries.

2.3.0 `ST_MinimumBoundingRadius(ST_GeomCollFromText('POLYGON((26426 65078,26531 65242,26075 65136,26096 65427,26426 65078))'))`

**ST\_MinimumBoundingCircle**

[ST\\_GeomCollFromText](#), [ST\\_MinimumBoundingCircle](#)

**ST\_MinimumBoundingCircle**

```

SELECT ST_AsText(center), radius FROM ST_MinimumBoundingRadius('POLYGON((26426 65078,26531 65242,26075 65136,26096 65427,26426 65078))');

```

| st_astext                                | radius           |
|--|------------------|
| POINT(26284.8418027133 65267.1145090825) | 247.436045591407 |

### 8.14.15 ST\_OrientedEnvelope

**ST\_OrientedEnvelope** — Returns a minimum-area rectangle containing a geometry.

**Synopsis**

geometry **ST\_OrientedEnvelope**( geometry geom );

**&#xc124;&#xba85;**

Returns the minimum-area rotated rectangle enclosing a geometry. Note that more than one such rectangle may exist. May return a Point or LineString in the case of degenerate inputs.

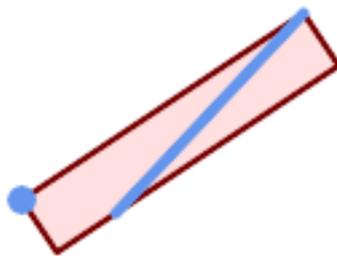
Availability: 2.5.0

**&#xcc38;&#xace0;**

**ST\_Envelope** **ST\_MinimumBoundingCircle**

**&#xc608;&#xc2dc;**

```
SELECT ST_AsText(ST_OrientedEnvelope('MULTIPOINT ((0 0), (-1 -1), (3 2))'));
      st_astext
      -----
POLYGON((3 2,2.88 2.16,-1.12 -0.84,-1 -1,3 2))
```



*Oriented envelope of a point and linestring.*

```
SELECT ST_AsText(ST_OrientedEnvelope(
  ST_Collect(
    ST_GeomFromText('LINESTRING(55 75,125 150)'),
    ST_Point(20, 80))
  )) As wktenv;
wktenv
-----
POLYGON((19.9999999999997 79.9999999999999,33.0769230769229 ↵
  60.3846153846152,138.076923076924 130.384615384616,125.000000000001 ↵
  150.000000000001,19.9999999999997 79.9999999999999))
```

## 8.14.16 ST\_OffsetCurve

ST\_OffsetCurve — Returns an offset line at a given distance and side from an input line.

### Synopsis

```
geometry ST_OffsetCurve(geometry line, float signed_distance, text style_parameters=');
```

### Return Value

Return an offset line at a given distance and side from an input line. All points of the returned geometries are not further than the given distance from the input geometry. Useful for computing parallel lines about a center line.

For positive distance the offset is on the left side of the input line and retains the same direction. For a negative distance it is on the right side and in the opposite direction.

For a distance of zero, the function returns the input geometry. For a distance of zero, the function returns the input geometry.

Note that output may be a MULTILINESTRING or EMPTY for some jigsaw-shaped input geometries.

For a distance of zero, the function returns the input geometry. For a distance of zero, the function returns the input geometry.

- 'quad\_segs=#' : (quarter circle)
- 'join=round|mitre|bevel' : (mitre) | (miter) | (bevel)
- 'mitre\_limit=#.#' : (mitre\_limit)

GEOS 2.0 - added support for GEOMETRYCOLLECTION and MULTILINESTRING

Enhanced: 2.5 - added support for GEOMETRYCOLLECTION and MULTILINESTRING



### Note

This function ignores the Z dimension. It always gives a 2D result even when used on a 3D geometry.

### Example

```
SELECT ST_Union(
```

```
  ST_OffsetCurve(f.geom, f.width/2, 'quad_segs=4 join=round'),
  ST_OffsetCurve(f.geom, -f.width/2, 'quad_segs=4 join=round')
) as track
FROM someroadstable;
```



```

--offset 15, 'quad_segs=4
--join=round'
--color red
--width 15

```

```

SELECT ST_AsText(ST_OffsetCurve(
  ST_GeomFromText(
'LINESTRING(164 16,144 16,124 16,104
  16,84 16,64 16,
  44 16,24 16,20 16,18 16,17 17,
  16 18,16 20,16 40,16 60,16 80,16 100,
  16 120,16 140,16 160,16 180,16 195)')
  ,
  15, 'quad_segs=4 join=round'));
--output --
LINESTRING(164 1,18 1,12.2597485145237
  2.1418070123307,
  7.39339828220179 5.39339828220179,
  5.39339828220179 7.39339828220179,
  2.14180701233067 12.2597485145237,1
  18,1 195)

```



```

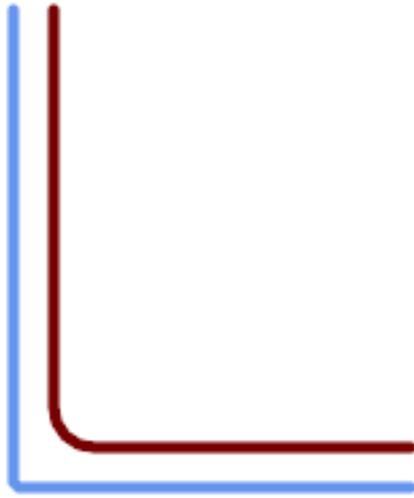
--offset -15, 'quad_segs=4
--join=round'
--color red
--width 15

```

```

SELECT ST_AsText(ST_OffsetCurve(geom,
  -15, 'quad_segs=4 join=round')) As
  notsocurvy
FROM ST_GeomFromText(
'LINESTRING(164 16,144 16,124 16,104
  16,84 16,64 16,
  44 16,24 16,20 16,18 16,17 17,
  16 18,16 20,16 40,16 60,16 80,16 100,
  16 120,16 140,16 160,16 180,16 195)')
  As geom;
-- notsocurvy --
LINESTRING(31 195,31 31,164 31)

```



```

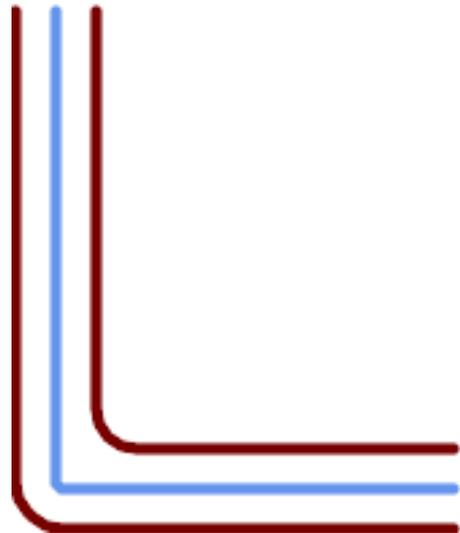
-- morecurvy --
SELECT ST_AsText(ST_OffsetCurve(
  ST_OffsetCurve(geom,
    -30, 'quad_segs=4 join=round'), -15,
    'quad_segs=4 join=round')) As morecurvy
FROM ST_GeomFromText(
  'LINESTRING(164 16,144 16,124 16,104
    16,84 16,64 16,
    44 16,24 16,20 16,18 16,17 17,
    16 18,16 20,16 40,16 60,16 80,16 100,
    16 120,16 140,16 160,16 180,16 195)')
  As geom;

```

```

SELECT ST_AsText(ST_OffsetCurve(
  ST_OffsetCurve(geom,
    -30, 'quad_segs=4 join=round'), -15,
    'quad_segs=4 join=round')) As morecurvy
FROM ST_GeomFromText(
  'LINESTRING(164 16,144 16,124 16,104
    16,84 16,64 16,
    44 16,24 16,20 16,18 16,17 17,
    16 18,16 20,16 40,16 60,16 80,16 100,
    16 120,16 140,16 160,16 180,16 195)')
  As geom;
-- morecurvy --
LINESTRING(164 31,46 31,40.2597485145236
  32.1418070123307,
  35.3933982822018 35.3933982822018,
  32.1418070123307 40.2597485145237,31
  46,31 195)

```



```

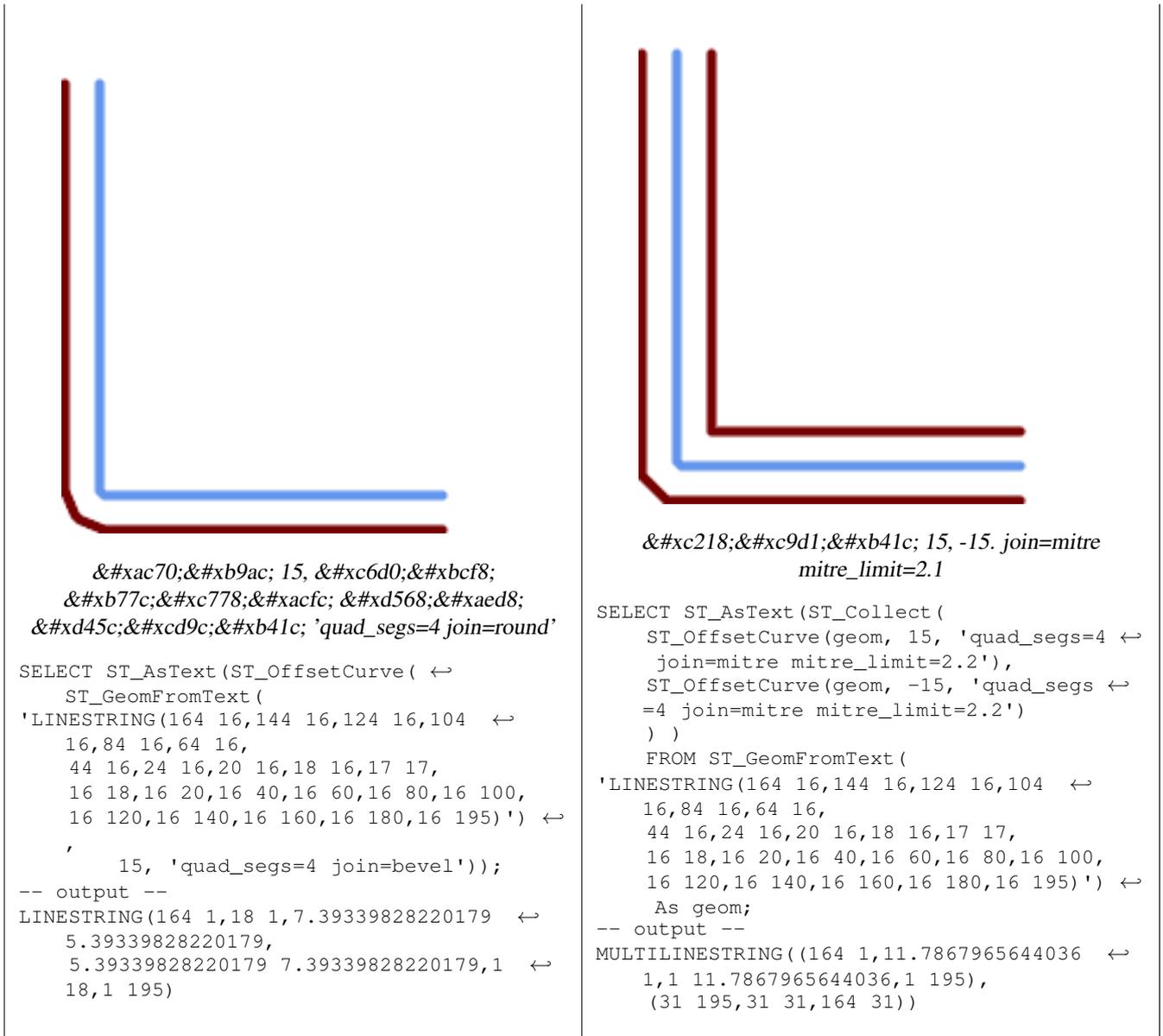
SELECT ST_AsText(ST_Collect(
  ST_OffsetCurve(geom, 15, 'quad_segs=4
    join=round'),
  ST_OffsetCurve(ST_OffsetCurve(geom,
    -30, 'quad_segs=4 join=round'), -15,
    'quad_segs=4 join=round')
  ) As parallel_curves
FROM ST_GeomFromText(
  'LINESTRING(164 16,144 16,124 16,104
    16,84 16,64 16,
    44 16,24 16,20 16,18 16,17 17,
    16 18,16 20,16 40,16 60,16 80,16 100,
    16 120,16 140,16 160,16 180,16 195)')
  As geom;
-- parallel curves --
MULTILINESTRING((164 1,18
  1,12.2597485145237 2.1418070123307,
  7.39339828220179
  5.39339828220179,5.39339828220179 7.393398282201
  2.14180701233067 12.2597485145237,1 18,1
  195),
  (164 31,46 31,40.2597485145236
  32.1418070123307,35.3933982822018 35.39339828220
  32.1418070123307 40.2597485145237,31
  46,31 195))

```

```

SELECT ST_AsText(ST_Collect(
  ST_OffsetCurve(geom, 15, 'quad_segs=4
    join=round'),
  ST_OffsetCurve(ST_OffsetCurve(geom,
    -30, 'quad_segs=4 join=round'), -15,
    'quad_segs=4 join=round')
  ) As parallel_curves
FROM ST_GeomFromText(
  'LINESTRING(164 16,144 16,124 16,104
    16,84 16,64 16,
    44 16,24 16,20 16,18 16,17 17,
    16 18,16 20,16 40,16 60,16 80,16 100,
    16 120,16 140,16 160,16 180,16 195)')
  As geom;
-- parallel curves --
MULTILINESTRING((164 1,18
  1,12.2597485145237 2.1418070123307,
  7.39339828220179
  5.39339828220179,5.39339828220179 7.393398282201
  2.14180701233067 12.2597485145237,1 18,1
  195),
  (164 31,46 31,40.2597485145236
  32.1418070123307,35.3933982822018 35.39339828220
  32.1418070123307 40.2597485145237,31
  46,31 195))

```



**ST\_Buffer**

**ST\_Buffer**

### 8.14.17 ST\_PointOnSurface

**ST\_PointOnSurface** — Computes a point guaranteed to lie in a polygon, or on a geometry.

#### Synopsis

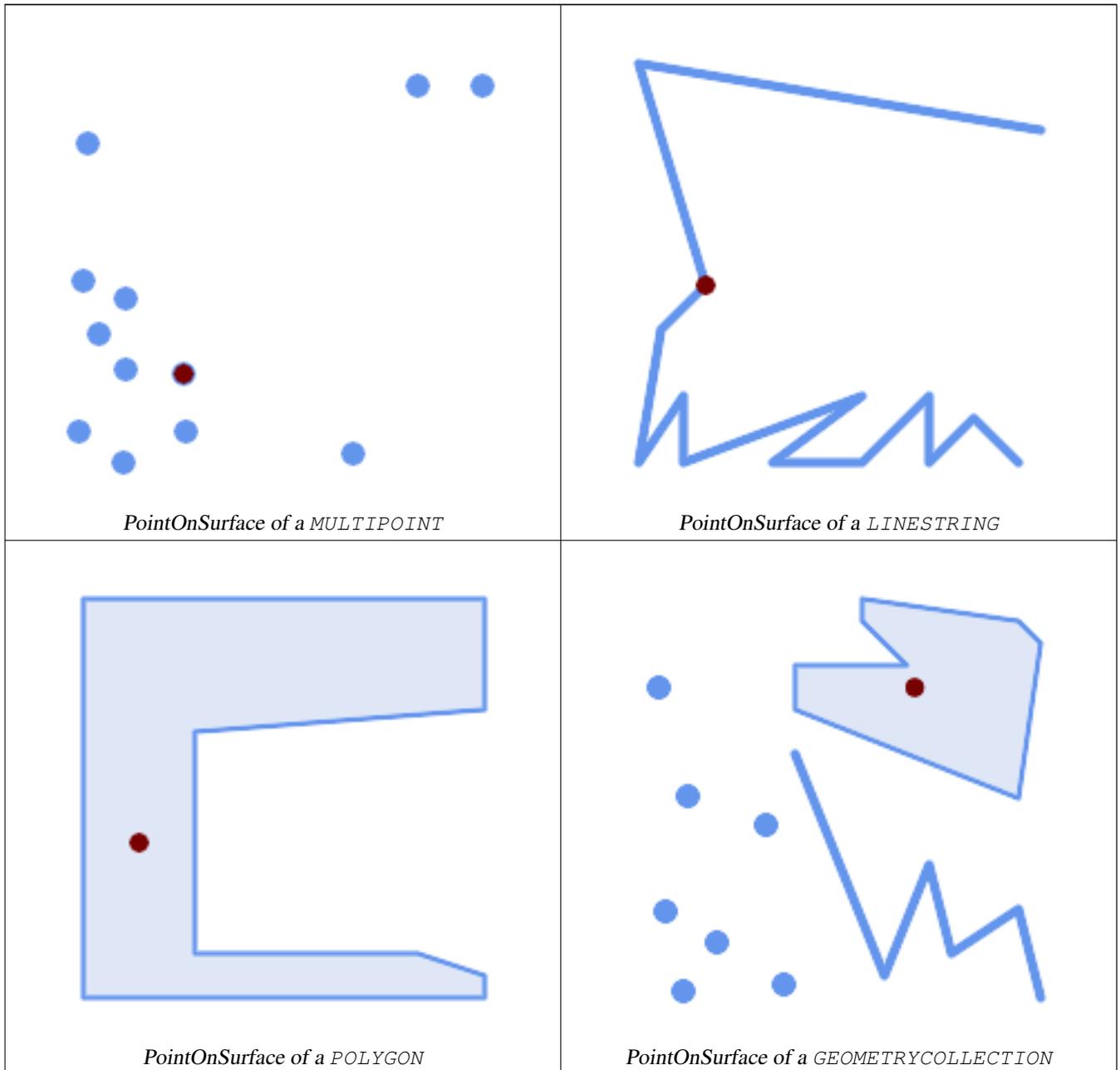
geometry **ST\_PointOnSurface**(geometry g1);

**ST\_PointOnSurface**

Returns a POINT which is guaranteed to lie in the interior of a surface (POLYGON, MULTIPOLYGON, and CURVED POLYGON). In PostGIS this function also works on line and point geometries.

- ✔ This method implements the [OGC Simple Features Implementation Specification for SQL 1.1](#). s3.2.14.2 // s3.2.18.2
- ✔ This method implements the SQL/MM specification. SQL-MM 3: 8.1.5, 9.5.6. The specifications define ST\_PointOnSurface for surface geometries only. PostGIS extends the function to support all common geometry types. Other databases (Oracle, DB2, ArcSDE) seem to support this function only for surfaces. SQL Server 2008 supports all common geometry types.
- ✔ This function supports 3d and will not drop the z-index.

⌘



```
SELECT ST_AsText (ST_PointOnSurface ('POINT (0 5) '::geometry));
-----
POINT (0 5)
```

```

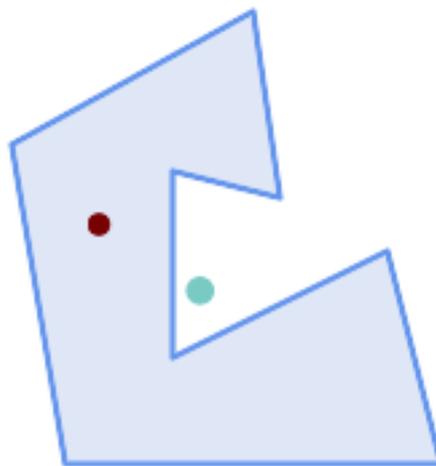
SELECT ST_AsText(ST_PointOnSurface('LINESTRING(0 5, 0 10)::geometry));
-----
POINT(0 5)

SELECT ST_AsText(ST_PointOnSurface('POLYGON((0 0, 0 5, 5 5, 5 0, 0 0))::geometry));
-----
POINT(2.5 2.5)

SELECT ST_AsEWKT(ST_PointOnSurface(ST_GeomFromEWKT('LINESTRING(0 5 1, 0 0 1, 0 10 2'))));
-----
POINT(0 0 1)

```

**Example:** The result of `ST_PointOnSurface` is guaranteed to lie within polygons, whereas the point computed by `ST_Centroid` may be outside.



*Red: point on surface; Green: centroid*

```

SELECT ST_AsText(ST_PointOnSurface(geom)) AS pt_on_surf,
       ST_AsText(ST_Centroid(geom)) AS centroid
FROM (SELECT 'POLYGON ((130 120, 120 190, 30 140, 50 20, 190 20,
                       170 100, 90 60, 90 130, 130 120))'::geometry AS geom) AS t;

pt_on_surf | centroid
-----+-----
POINT(62.5 110) | POINT(100.18264840182648 85.11415525114155)

```

**See also:**

[ST\\_Centroid](#), [ST\\_MaximumInscribedCircle](#)

### 8.14.18 ST\_Polygonize

`ST_Polygonize` — Computes a collection of polygons formed from the linework of a set of geometries.

#### Synopsis

```

geometry ST_Polygonize(geometry set geomfield);
geometry ST_Polygonize(geometry[] geom_array);

```



## Synopsis

geometry **ST\_ReducePrecision**(geometry g, float8 gridsize);

### Return Value

Returns a valid geometry with all points rounded to the provided grid tolerance, and features below the tolerance removed.

Unlike **ST\_SnapToGrid** the returned geometry will be valid, with no ring self-intersections or collapsed components.

Precision reduction can be used to:

- match coordinate precision to the data accuracy
- reduce the number of coordinates needed to represent a geometry
- ensure valid geometry output to formats which use lower precision (e.g. text formats such as WKT, GeoJSON or KML when the number of output decimal places is limited).
- export valid geometry to systems which use lower or limited precision (e.g. SDE, Oracle tolerance value)

Availability: 3.1.0 - requires GEOS >= 3.9.0.

### Examples

```
SELECT ST_AsText(ST_ReducePrecision('POINT(1.412 19.323)', 0.1));
      st_astext
-----
POINT(1.4 19.3)

SELECT ST_AsText(ST_ReducePrecision('POINT(1.412 19.323)', 1.0));
      st_astext
-----
POINT(1 19)

SELECT ST_AsText(ST_ReducePrecision('POINT(1.412 19.323)', 10));
      st_astext
-----
POINT(0 20)
```

### Precision reduction can reduce number of vertices

```
SELECT ST_AsText(ST_ReducePrecision('LINESTRING (10 10, 19.6 30.1, 20 30, 20.3 30, 40 40)', 1));
      st_astext
-----
LINESTRING (10 10, 20 30, 40 40)
```

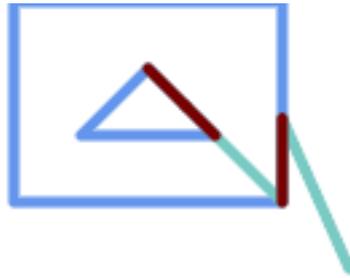
### Precision reduction splits polygons if needed to ensure validity

```
SELECT ST_AsText(ST_ReducePrecision('POLYGON ((10 10, 60 60.1, 70 30, 40 40, 50 10, 10 10))', 10));
      st_astext
-----
MULTIPOLYGON (((60 60, 70 30, 40 40, 60 60)), ((40 40, 50 10, 10 10, 40 40)))
```

### See Also

[ST\\_SnapToGrid](#), [ST\\_Simplify](#), [ST\\_SimplifyVW](#)





```

-- Example 1: Shared paths between a square and a triangle
-- The red line is shared between the square and the triangle
-- The green line is shared between the square and the triangle
-- The red vertical bar is the intersection of the red and green lines

```

```

SELECT ST_AsText(
  ST_SharedPaths(
    ST_GeomFromText('MULTILINESTRING((26 125,26 200,126 200,126 125,26 125),
      (51 150,101 150,76 175,51 150))'),
    ST_GeomFromText('LINESTRING(151 100,126 156.25,126 125,90 161, 76 175)')
  )
) As wkt

-----
wkt

```

```

-----
GEOMETRYCOLLECTION(MULTILINESTRING((126 156.25,126 125),
(101 150,90 161),(90 161,76 175)),MULTILINESTRING EMPTY)

```

```

-- same example but linestring orientation flipped

```

```

SELECT ST_AsText(
  ST_SharedPaths(
    ST_GeomFromText('LINESTRING(76 175,90 161,126 125,126 156.25,151 100)'),
    ST_GeomFromText('MULTILINESTRING((26 125,26 200,126 200,126 125,26 125),
      (51 150,101 150,76 175,51 150))')
  )
) As wkt

-----
wkt

```

```

-----
GEOMETRYCOLLECTION(MULTILINESTRING EMPTY,
MULTILINESTRING((76 175,90 161),(90 161,101 150),(126 125,126 156.25)))

```

**ST\_Dump**

**ST\_Dump**, **ST\_GeometryN**, **ST\_NumGeometries**

## 8.14.21 ST\_Simplify

ST\_Simplify — Returns a simplified version of a geometry, using the Douglas-Peucker algorithm.

### Synopsis

```
geometry ST_Simplify(geometry geomA, float tolerance);
geometry ST_Simplify(geometry geomA, float tolerance, boolean preserveCollapsed);
```

### Note

The Douglas-Peucker algorithm (Douglas-Peucker) simplifies a geometry by removing vertices that are not needed to define the shape. The tolerance parameter specifies the maximum distance between a vertex and the line segment defined by the other two vertices. Vertices that are further than the tolerance from the line segment are kept, while those closer are removed. The algorithm starts with the first vertex and iteratively finds the vertex furthest from the line segment. If this distance is greater than the tolerance, the vertex is kept and the process repeats. If not, the vertex is removed and the process repeats with the next vertex. The process continues until all vertices are either kept or removed. The result is a simplified geometry that is topologically equivalent to the original but has fewer vertices. The `preserveCollapsed` flag will retain objects that would otherwise be too small given the tolerance. For example, a 1m long line simplified with a 10m tolerance. If `preserveCollapsed` argument is specified as true, the line will not disappear. This flag is useful for rendering engines, to avoid having large numbers of very small objects disappear from a map leaving surprising gaps.



### Note

The `ST_IsSimple` function can be used to check if a geometry is simple. For example, a line with a self-intersection is not simple. The `ST_IsSimple` function returns true if the geometry is simple and false otherwise.



### Note

The `ST_SimplifyPreserveTopology` function can be used to simplify a geometry while preserving its topology. For example, a line with a self-intersection will remain a line with a self-intersection after simplification. The `ST_SimplifyPreserveTopology` function returns a simplified geometry that is topologically equivalent to the original but has fewer vertices.

1.2.2 The `ST_Simplify` function can be used to simplify a geometry while preserving its topology. For example, a line with a self-intersection will remain a line with a self-intersection after simplification. The `ST_SimplifyPreserveTopology` function returns a simplified geometry that is topologically equivalent to the original but has fewer vertices.

### Note

The `ST_Simplify` function can be used to simplify a geometry while preserving its topology. For example, a line with a self-intersection will remain a line with a self-intersection after simplification. The `ST_SimplifyPreserveTopology` function returns a simplified geometry that is topologically equivalent to the original but has fewer vertices.

```
SELECT ST_Npoints(geom) AS np_before,
       ST_NPoints(ST_Simplify(geom,0.1)) AS np01_notbadcircle,
       ST_NPoints(ST_Simplify(geom,0.5)) AS np05_notquitecircle,
       ST_NPoints(ST_Simplify(geom,1)) AS np1_octagon,
       ST_NPoints(ST_Simplify(geom,10)) AS np10_triangle,
       (ST_Simplify(geom,100) is null) AS np100_geometrygoesaway
FROM
  (SELECT ST_Buffer('POINT(1 3)', 10,12) As geom) AS foo;
```

| np_before              | np01_notbadcircle | np05_notquitecircle | np1_octagon | np10_triangle | ↔ |
|------------------------|-------------------|---------------------|-------------|---------------|---|
| np100_geometrygoesaway |                   |                     |             |               |   |
| 49                     | 33                | 17                  | 9           | 4             | t |

**ST\_IsSimple, ST\_SimplifyPreserveTopology, ST\_SimplifyVW, Topology ST\_Simplify**

### 8.14.22 ST\_SimplifyPreserveTopology

ST\_SimplifyPreserveTopology — Returns a simplified and valid version of a geometry, using the Douglas-Peucker algorithm.

#### Synopsis

geometry **ST\_SimplifyPreserveTopology**(geometry geomA, float tolerance);

**ST\_SimplifyPreserveTopology**

(Douglas-Peucker) (geometry geomA, float tolerance);

ST\_SimplifyPreserveTopology(geometry geomA, float tolerance) — Returns a simplified and valid version of a geometry, using the Douglas-Peucker algorithm.

GEOS (Douglas-Peucker) (geometry geomA, float tolerance);

1.3.3 (Douglas-Peucker) (geometry geomA, float tolerance);

**ST\_SimplifyPreserveTopology**

ST\_SimplifyPreserveTopology(geometry geomA, float tolerance) — Returns a simplified and valid version of a geometry, using the Douglas-Peucker algorithm.

```
SELECT ST_Npoints(geom) As np_before, ST_NPoints(ST_SimplifyPreserveTopology(geom,0.1)) As np01_notbadcircle, ST_NPoints(ST_SimplifyPreserveTopology(geom,0.5)) As np05_notquitecircle, ST_NPoints(ST_SimplifyPreserveTopology(geom,1)) As np1_octagon, ST_NPoints(ST_SimplifyPreserveTopology(geom,10)) As np10_square, ST_NPoints(ST_SimplifyPreserveTopology(geom,100)) As np100_stillsquare FROM (SELECT ST_Buffer('POINT(1 3)', 10,12) As geom) As foo;
```

```
--result--
```

| np_before         | np01_notbadcircle | np05_notquitecircle | np1_octagon | np10_square | ↔ |
|-------------------|-------------------|---------------------|-------------|-------------|---|
| np100_stillsquare |                   |                     |             |             |   |
| 49                | 33                | 17                  | 9           | 5           | ↔ |
|                   | 5                 |                     |             |             |   |

**ST\_Simplify**

**ST\_Simplify**

### 8.14.23 ST\_SimplifyPolygonHull

**ST\_SimplifyPolygonHull** — Computes a simplified topology-preserving outer or inner hull of a polygonal geometry.

#### Synopsis

```
geometry ST_SimplifyPolygonHull(geometry param_geom, float vertex_fraction, boolean is_outer = true);
```

**ST\_SimplifyPolygonHull**

Computes a simplified topology-preserving outer or inner hull of a polygonal geometry. An outer hull completely covers the input geometry. An inner hull is completely covered by the input geometry. The result is a polygonal geometry formed by a subset of the input vertices. MultiPolygons and holes are handled and produce a result with the same structure as the input.

The reduction in vertex count is controlled by the `vertex_fraction` parameter, which is a number in the range 0 to 1. Lower values produce simpler results, with smaller vertex count and less concaveness. For both outer and inner hulls a vertex fraction of 1.0 produces the original geometry. For outer hulls a value of 0.0 produces the convex hull (for a single polygon); for inner hulls it produces a triangle.

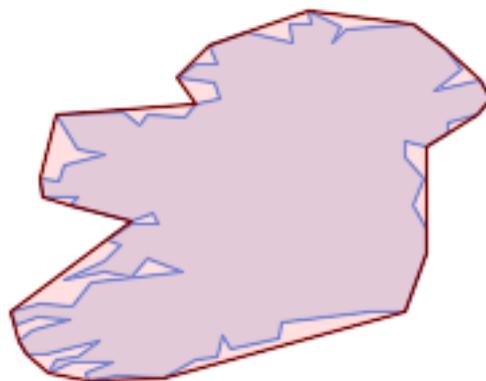
The simplification process operates by progressively removing concave corners that contain the least amount of area, until the vertex count target is reached. It prevents edges from crossing, so the result is always a valid polygonal geometry.

To get better results with geometries that contain relatively long line segments, it might be necessary to "segmentize" the input, as shown below.

GEOS `ST_SimplifyPolygonHull`

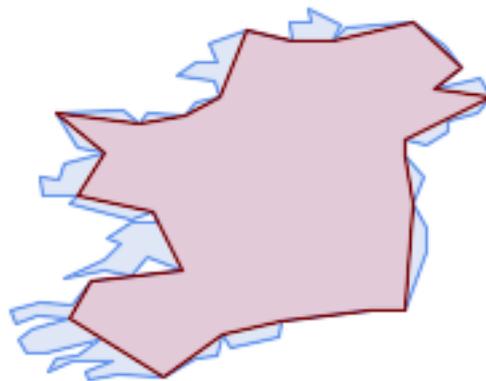
Availability: 3.3.0 - requires GEOS >= 3.11.0

**ST\_SimplifyPolygonHull**



*Outer hull of a Polygon*

```
SELECT ST_SimplifyPolygonHull(  
  'POLYGON ((131 158, 136 163, 161 165, 173 156, 179 148, 169 140, 186 144, 190 137, 185 ↵  
    131, 174 128, 174 124, 166 119, 158 121, 158 115, 165 107, 161 97, 166 88, 166 79, 158 ↵  
      57, 145 57, 112 53, 111 47, 93 43, 90 48, 88 40, 80 39, 68 32, 51 33, 40 31, 39 34, ↵  
    49 38, 34 38, 25 34, 28 39, 36 40, 44 46, 24 41, 17 41, 14 46, 19 50, 33 54, 21 55, 13 ↵  
      52, 11 57, 22 60, 34 59, 41 68, 75 72, 62 77, 56 70, 46 72, 31 69, 46 76, 52 82, 47 ↵  
    84, 56 90, 66 90, 64 94, 56 91, 33 97, 36 100, 23 100, 22 107, 29 106, 31 112, 46 116, ↵  
      36 118, 28 131, 53 132, 59 127, 62 131, 76 130, 80 135, 89 137, 87 143, 73 145, 80 ↵  
    150, 88 150, 85 157, 99 162, 116 158, 115 165, 123 165, 122 170, 134 164, 131 158))',  
  0.3);
```



*Inner hull of a Polygon*

```
SELECT ST_SimplifyPolygonHull(  
  'POLYGON ((131 158, 136 163, 161 165, 173 156, 179 148, 169 140, 186 144, 190 137, 185 ↵  
    131, 174 128, 174 124, 166 119, 158 121, 158 115, 165 107, 161 97, 166 88, 166 79, 158 ↵  
      57, 145 57, 112 53, 111 47, 93 43, 90 48, 88 40, 80 39, 68 32, 51 33, 40 31, 39 34, ↵  
    49 38, 34 38, 25 34, 28 39, 36 40, 44 46, 24 41, 17 41, 14 46, 19 50, 33 54, 21 55, 13 ↵  
      52, 11 57, 22 60, 34 59, 41 68, 75 72, 62 77, 56 70, 46 72, 31 69, 46 76, 52 82, 47 ↵  
    84, 56 90, 66 90, 64 94, 56 91, 33 97, 36 100, 23 100, 22 107, 29 106, 31 112, 46 116, ↵  
      36 118, 28 131, 53 132, 59 127, 62 131, 76 130, 80 135, 89 137, 87 143, 73 145, 80 ↵  
    150, 88 150, 85 157, 99 162, 116 158, 115 165, 123 165, 122 170, 134 164, 131 158))',  
  0.3, false);
```



Outer hull simplification of a MultiPolygon, with segmentization

```
SELECT ST_SimplifyPolygonHull(
  ST_Segmentize(ST_Letters('xt'), 2.0),
  0.1);
```

**See also:**

[ST\\_ConvexHull](#), [ST\\_SimplifyVW](#), [ST\\_ConcaveHull](#), [ST\\_Segmentize](#)

### 8.14.24 ST\_SimplifyVW

`ST_SimplifyVW` — Returns a simplified version of a geometry, using the Visvalingam-Whyatt algorithm

#### Synopsis

geometry `ST_SimplifyVW`(geometry geomA, float tolerance);

**See also:**

(Visvalingam-Whyatt) algorithm. Returns a simplified version of a geometry, using the Visvalingam-Whyatt algorithm. The function takes a geometry and a tolerance value as input. The tolerance value is used to determine the maximum distance between the original geometry and the simplified geometry. The function returns a simplified version of the geometry, where the distance between the original geometry and the simplified geometry is less than or equal to the tolerance value. The function is useful for simplifying complex geometries, such as polygons with many vertices, into a simpler representation that is easier to work with. The function is implemented using the Visvalingam-Whyatt algorithm, which is a topological simplification algorithm that removes vertices from a polygon based on their contribution to the total area of the polygon. The function is available in PostGIS 3.3.6dev.



#### Note

The function `ST_SimplifyVW` is implemented using the Visvalingam-Whyatt algorithm. The function is available in PostGIS 3.3.6dev. The function is useful for simplifying complex geometries, such as polygons with many vertices, into a simpler representation that is easier to work with. The function is implemented using the Visvalingam-Whyatt algorithm, which is a topological simplification algorithm that removes vertices from a polygon based on their contribution to the total area of the polygon. The function is available in PostGIS 3.3.6dev.



&#xc788;&#xc2b5;&#xb2c8;&#xb2e4;. &#xb610;&#xb2e4;&#xb978; &#xc635;&#xc158;&#xc740; &#xc784;&#xacc4;&#xce58;&#xc00&#xc73c;&#xb85c; &#xc124;&#xc815;&#xd558;&#xb294; &#xac83;&#xc785;&#xb2c8;&#xb2e4;. &#xc774;&#xb7f4; &#xacbd;&#xc720;&#xd6a8; &#xbc94;&#xc704;&#xb97c; M&#xac12;&#xc73c;&#xb85c; &#xac00;&#xc9c4; &#xc804;&#xccb4; &#xb3c4;&#xb85c;&#xd658;&#xd558;&#xb294;&#xb370;. &#xc774; &#xb3c4;&#xd615;&#xc744; &#xd074;&#xb77c;&#xc774;&#xc5b8;&#xb9e4;&#xc6b0; &#xbe68;&#xb9ac; &#xb2e8;&#xc21c;&#xd654;&#xd558;&#xb294; &#xb370; &#xc4f8; &#xc218; &#xc788;&#xc2e4;&#xc81c;&#xb85c;&#xb294; [&#xba40;&#xd2f0;]&#xb77c;&#xc778;. [&#xba40;&#xd2f0;]&#xd3f4;&#xb9ac;&#xace4;&#xc791;&#xb3d9;&#xd558;&#xc9c0;&#xb9cc;. &#xc5b4;&#xb5a4; &#xc885;&#xb958;&#xc758; &#xb3c4;&#xd615;&#xb3c4; &#xc785;&#xb825;&#xd560; &#xc218; &#xc788;&#xb2e4;&#xace0; &#xd574;&#xb3c4; &#xacfc;&#xc5b8;&#xc740; &#xc544;&#xac1d;&#xccb4;&#xbcc4; &#xae30;&#xb81c;&#xc73c;&#xb85c; &#xb2e8;&#xc21c;&#xd654; &#xc791;&#xc5c5;&#xc744; &#xd558;&#xae30; &#xb54c;&#xbb38;&#xc5d0; &#xc774; &#xd568;&#xc218;&#xc5d0; &#xb3c4;&#xd615; &#xc9d1;&#xd569;&#xc785;&#xb825;&#xd560; &#xc218; &#xc788;&#xc2b5;&#xb2c8;&#xb2e4;.



**Note**

&#xb81c;&#xd658;&#xb418;&#xb294; &#xb3c4;&#xd615;&#xc774; &#xb2e8;&#xc21c;&#xc131;&#xc744; &#xc783;&#xc744; &#xc218;&#xb3c4; &#xc788;&#xb2e4;&#xb294; &#xc810;&#xc5d0; &#xc8fc;&#xc758;&#xd558;&#xc2ed;&#xc2dc;&#xc624;(ST\_IsSimple &#xcc38;&#xc870;).



**Note**

&#xc704;&#xc0c1;(topology)&#xc774; &#xbcf4;&#xc804;&#xb418;&#xc9c0; &#xc54a;&#xc544; &#xc720;&#xd6a8;&#xd558;&#xc9c0; &#xc54a;&#xc740; &#xb3c4;&#xd615;&#xc774; &#xb81c;&#xd658;&#xb420; &#xc218;&#xb3c4; &#xc788;&#xc2b5;&#xb2c8;&#xb2e4;. &#xc704;&#xc0c1;&#xc744; &#xc720;&#xc9c0;&#xd558;&#xb824;&#xba74; ST\_SimplifyPreserveTopology &#xd568;&#xc218;&#xb97c; &#xc774;&#xc6a9;&#xd558;&#xc2ed;&#xc2dc;&#xc624;.



**Note**

&#xcd9c;&#xb825; &#xb3c4;&#xd615;&#xc740; M&#xac12;&#xc73c;&#xb85c; &#xac00;&#xc9c0;&#xace0; &#xc788;&#xb358; &#xc815;&#xbcf4;&#xb97c; &#xbaa8;&#xb450; &#xc783;&#xac8c; &#xb420; &#xac83;&#xc785;&#xb2c8;&#xb2e4;.



**Note**

&#xc774; &#xd568;&#xc218;&#xb294; 3&#xcc28;&#xc6d0;&#xc744; &#xcc98;&#xb9ac;&#xd558;&#xba70;. &#xc138; &#xb88;&#xc9f8; &#xcc28;&#xc6d0;&#xc774; &#xc720;&#xd6a8; &#xb81c;&#xc704;&#xc5d0; &#xc601;&#xd5a5;&#xc744; &#xbbf8;&#xce60; &#xac83;&#xc785;&#xb2c8;&#xb2e4;.

2.2.0 &#xb84;&#xc804;&#xbd80;&#xd130; &#xc0ac;&#xc6a9;&#xd560; &#xc218; &#xc788;&#xc2b5;&#xb2c8;&#xb2e4;.

**&#xc608;&#xc2dc;**

&#xb77c;&#xc778;&#xc2a4;&#xd2b8;&#xb9c1;&#xc758; &#xc720;&#xd6a8; &#xb81c;&#xc704;&#xb97c; &#xacc4;&#xc0b0;&#xc784;&#xacc4;&#xce58;&#xb97c; 0&#xc73c;&#xb85c; &#xc124;&#xc815;&#xd558;&#xae30; &#xb54c;&#xbb38;&#xc5d0;. &#xc785;&#xb825; &#xb3c4;&#xd615; &#xc548;&#xc5d0; &#xc788;&#xb294; &#xbaa8;&#xb4e0; &#xaf2d;&#xc9d3;&#xc810;&#xb81c;&#xd658;&#xd569;&#xb2c8;&#xb2e4;.

```

select ST_AsText(ST_SetEffectiveArea(geom)) all_pts, ST_AsText(ST_SetEffectiveArea(geom,30) ←
    ) thrshld_30
FROM (SELECT 'LINESTRING(5 2, 3 8, 6 20, 7 25, 10 10)>:::geometry geom) As foo;
-- &#xacb0;&#xacfc; --
all_pts | thrshld_30
-----+-----+
LINESTRING M (5 2 3.40282346638529e+38,3 8 29,6 20 1.5,7 25 49.5,10 10 3.40282346638529e ←
    +38) | LINESTRING M (5 2 3.40282346638529e+38,7 25 49.5,10 10 3.40282346638529e+38)
    
```

**ST\_SimplifyVW**

**ST\_SimplifyVW**

## 8.14.26 ST\_TriangulatePolygon

**ST\_TriangulatePolygon** — Computes the constrained Delaunay triangulation of polygons

### Synopsis

geometry **ST\_TriangulatePolygon**(geometry geom);

**ST\_TriangulatePolygon**

Computes the constrained Delaunay triangulation of polygons. Holes and Multipolygons are supported.

The "constrained Delaunay triangulation" of a polygon is a set of triangles formed from the vertices of the polygon, and covering it exactly, with the maximum total interior angle over all possible triangulations. It provides the "best quality" triangulation of the polygon.

Availability: 3.3.0 - requires GEOS >= 3.11.0

### Example

Triangulation of a square.

```
SELECT ST_AsText (
  ST_TriangulatePolygon('POLYGON((0 0, 0 1, 1 1, 1 0, 0 0))');

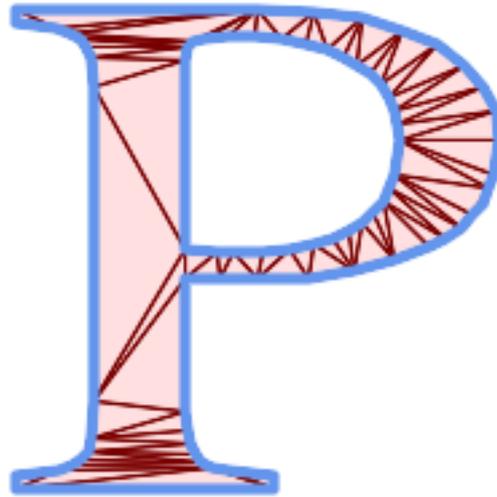
```

| st_astext   |
|---|
| GEOMETRYCOLLECTION(POLYGON((0 0,0 1,1 1,0 0)),POLYGON((1 1,1 0,0 0,1 1))) |

### Example

Triangulation of the letter P.

```
SELECT ST_AsText(ST_TriangulatePolygon(
  'POLYGON ((26 17, 31 19, 34 21, 37 24, 38 29, 39 43, 39 161, 38 172, 36 176, 34 179, 30 ←
    181, 25 183, 10 185, 10 190, 100 190, 121 189, 139 187, 154 182, 167 177, 177 169, ←
    184 161, 189 152, 190 141, 188 128, 186 123, 184 117, 180 113, 176 108, 170 104, 164 ←
    101, 151 96, 136 92, 119 89, 100 89, 86 89, 73 89, 73 39, 74 32, 75 27, 77 23, 79 ←
    20, 83 18, 89 17, 106 15, 106 10, 10 10, 10 15, 26 17), (152 147, 151 152, 149 157, ←
    146 162, 142 166, 137 169, 132 172, 126 175, 118 177, 109 179, 99 180, 89 180, 80 ←
    179, 76 178, 74 176, 73 171, 73 100, 85 99, 91 99, 102 99, 112 100, 121 102, 128 ←
    104, 134 107, 139 110, 143 114, 147 118, 149 123, 151 128, 153 141, 152 147))'
));
```



*Polygon Triangulation*

**ST\_DelaunayTriangles**

[ST\\_DelaunayTriangles](#), [ST\\_ConstrainedDelaunayTriangles](#), [ST\\_Tesselate](#)

#### 8.14.27 ST\_VoronoiLines

`ST_VoronoiLines` — Returns the boundaries of the Voronoi diagram of the vertices of a geometry.

##### Synopsis

geometry `ST_VoronoiLines`( g1 geometry , tolerance float8 , extend\_to geometry );

**ST\_VoronoiLines**

`ST_VoronoiLines` computes a two-dimensional **Voronoi diagram** from the vertices of the supplied geometry and returns the boundaries between cells in that diagram as a MultiLineString. Returns null if input geometry is null. Returns an empty geometry collection if the input geometry contains only one vertex. Returns an empty geometry collection if the `extend_to` envelope has zero area.

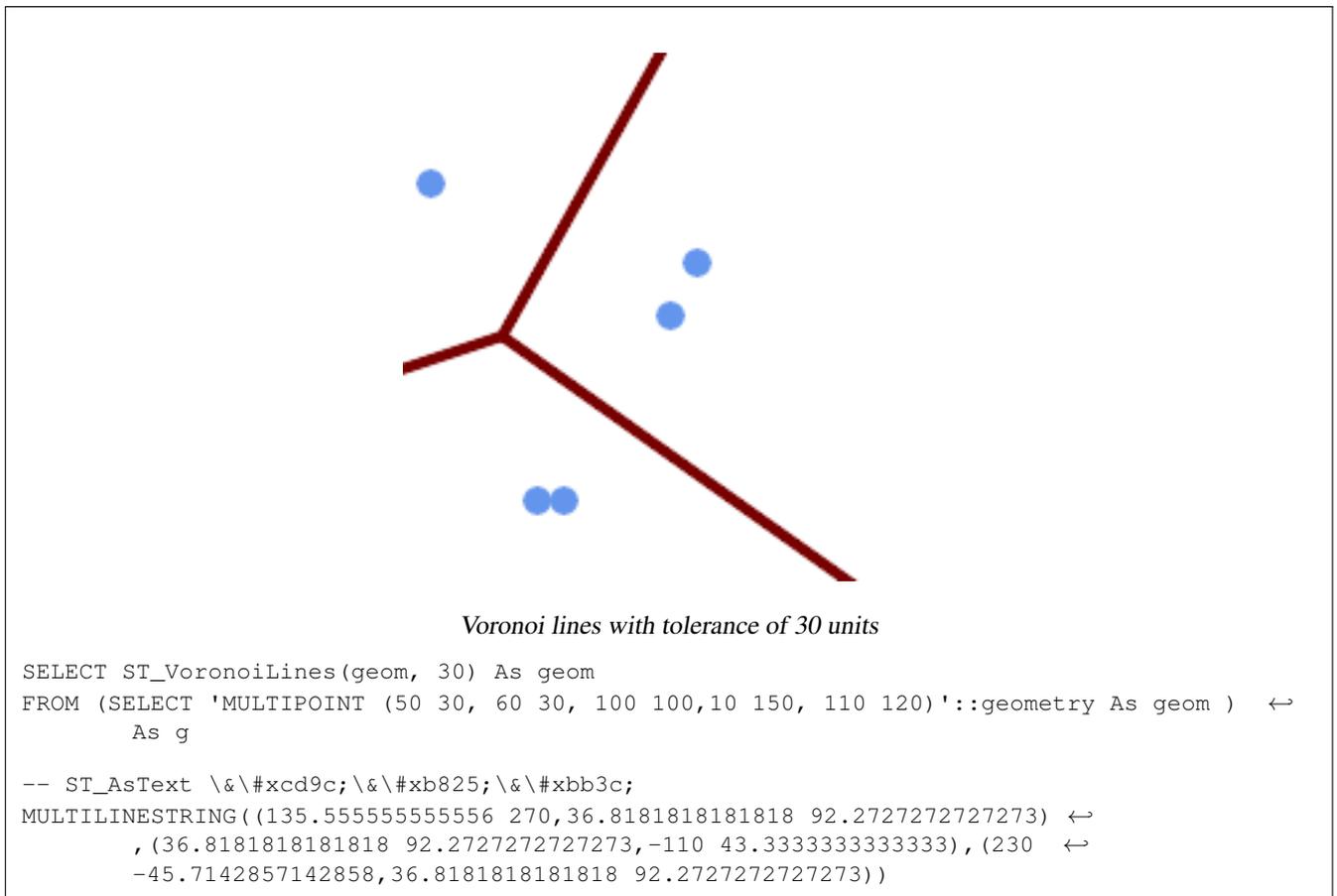
`ST_VoronoiLines`( geometry, tolerance, extend\_to )

- `'tolerance'` : A value that controls the precision of the Voronoi diagram. A value of 0.0 means that the diagram is computed with infinite precision. A non-zero value means that the diagram is computed with finite precision. The default value is 0.0.
- `'extend_to'` : If a geometry is supplied as the "extend\_to" parameter, the diagram will be extended to cover the envelope of the "extend\_to" geometry, unless that envelope is smaller than the default envelope (default = NULL, default envelope is boundingbox of input geometry extended by about 50% in each direction).

GEOS `ST_VoronoiLines`( geometry, tolerance, extend\_to )

2.3.0 `ST_VoronoiLines`( geometry, tolerance, extend\_to )

**ST\_VoronoiLines**



**&\#xc38;\#xace0;**

[ST\\_DelaunayTriangles](#), [ST\\_VoronoiPolygons](#), [ST\\_GeomCollFromText](#)

### 8.14.28 ST\_VoronoiPolygons

`ST_VoronoiPolygons` — Returns the cells of the Voronoi diagram of the vertices of a geometry.

#### Synopsis

geometry `ST_VoronoiPolygons`( g1 geometry , tolerance float8 , extend\_to geometry );

**&\#xc124;\#xba85;**

`ST_VoronoiPolygons` computes a two-dimensional **Voronoi diagram** from the vertices of the supplied geometry. The result is a `GeometryCollection` of Polygons that covers an envelope larger than the extent of the input vertices. Returns null if input geometry is null. Returns an empty geometry collection if the input geometry contains only one vertex. Returns an empty geometry collection if the `extend_to` envelope has zero area.

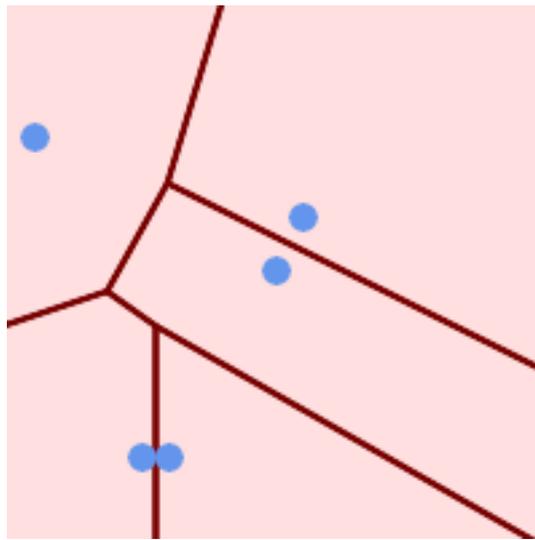
`&\#xc120;\#xd0dd;\#xd560; &\#xc218; &\#xc788;\#xb294; &\#xd30c;\#xb77c;\#xbbf8;\#xd130;`

- 'tolerance' : `&\#xadf8;\#xbcf4;\#xb2e4; &\#xac00;\#xae4c;\#xc6b4; &\#xaf2d;\#xc9d3;\#xc810;\#xb4e4;\#xc744; &\#xb3d9; &\#xac04;\#xc8fc;\#xd560; &\#xd5c8;\#xc6a9; &\#xc624;\#xcc28; &\#xac70;\#xb9ac;\#xc785;\#xb2c8;\#xb2e4;. 0&\#xc774; &\#xc544;\#xb2cc; &\#xd5c8;\#xc6a9; &\#xc624;\#xcc28; &\#xac70;\#xb9ac;\#xb97c; &\#xc124;\#xc815;\#xd558;\#ba74; &\#xc54c;\#xace0;\#xb9ac;\#xc998;\#xc758; &\#xac15;\#xb825;\#xd568;\#xc744; &\#xd5a5;\#xc0c1;\#xc2dc;\#xd0ac; &\#xc218; &\#xc788;\#xc2b5;\#xb2c8;\#xb2e4;. (&\#xae30;\#xbcf8;\#xac12; = 0.0)`

- 'extend\_to' : If a geometry is supplied as the "extend\_to" parameter, the diagram will be extended to cover the envelope of the "extend\_to" geometry, unless that envelope is smaller than the default envelope (default = NULL, default envelope is boundingbox of input geometry extended by about 50% in each direction).

GEOS

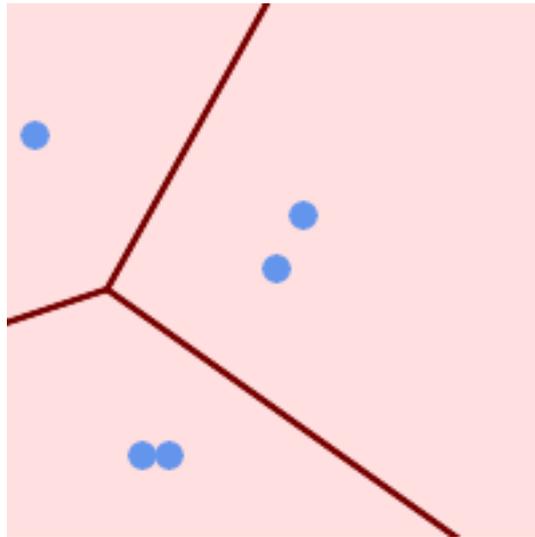
2.3.0



*Points overlaid on top of Voronoi diagram*

```
SELECT
  ST_VoronoiPolygons(geom) As geom
FROM (SELECT 'MULTIPOINT (50 30, 60 30, 100 100,10 150, 110 120)::geometry As geom ) ↔
      As g;

-- ST_AsText '\&\#xcd9c;\&\#xb825;\&\#xbb3c;
GEOMETRYCOLLECTION(POLYGON((-110 43.3333333333333, -110 270, 100.5 270, 59.3478260869565 ↔
  132.826086956522, 36.8181818181818 92.2727272727273, -110 43.3333333333333)),
POLYGON((55 -90, -110 -90, -110 43.3333333333333, 36.8181818181818 92.2727272727273, 55 ↔
  79.2857142857143, 55 -90)),
POLYGON((230 47.5, 230 -20.7142857142857, 55 79.2857142857143, 36.8181818181818 ↔
  92.2727272727273, 59.3478260869565 132.826086956522, 230 47.5)), POLYGON((230 ↔
  -20.7142857142857, 230 -90, 55 -90, 55 79.2857142857143, 230 -20.7142857142857)),
POLYGON((100.5 270, 230 270, 230 47.5, 59.3478260869565 132.826086956522, 100.5 270)))
```

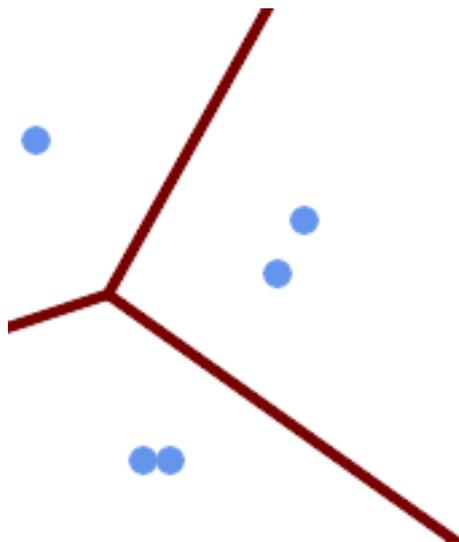


&#xd5c8;&#xc6a9; &#xc624;&#xcc28; &#xac70;&#xb9ac;&#xac00; 30 &#xb2e8;&#xc704;&#xc778;  
 &#xbcf4;&#xb85c;&#xb178;&#xc774; &#xb2e4;&#xc774;&#xc5b4;&#xadf8;&#xb7a8;

```

SELECT ST_VoronoiPolygons(geom, 30) As geom
FROM (SELECT 'MULTIPOINT (50 30, 60 30, 100 100,10 150, 110 120) '::geometry As geom ) ←
      As g;

-- ST_AsText \&#\xcd9c;\&#\xb825;\&#\xbb3c;
GEOMETRYCOLLECTION(POLYGON((-110 43.33333333333333,-110 270,100.5 270,59.3478260869565 ←
  132.826086956522,36.8181818181818 92.2727272727273,-110 43.3333333333333)), ←
POLYGON((230 47.5,230 -45.7142857142858,36.8181818181818 ←
  92.2727272727273,59.3478260869565 132.826086956522,230 47.5)),POLYGON((230 ←
  -45.7142857142858,230 -90,-110 -90,-110 43.3333333333333,36.8181818181818 ←
  92.2727272727273,230 -45.7142857142858)), ←
POLYGON((100.5 270,230 270,230 47.5,59.3478260869565 132.826086956522,100.5 270)))
  
```



Voronoi with tolerance of 30 units as MultiLineString

```
SELECT ST_VoronoiLines(geom, 30) As geom
FROM (SELECT 'MULTIPOINT (50 30, 60 30, 100 100,10 150,110 120) '::geometry As geom ) ↔
      As g

-- ST_AsText \&\#xcd9c;\&\#xb825;\&\#xbb3c;
MULTILINESTRING((135.555555555556 270,36.8181818181818 92.2727272727273) ↔
, (36.8181818181818 92.2727272727273,-110 43.3333333333333), (230 ↔
-45.7142857142858,36.8181818181818 92.2727272727273))
```

**&\#xcc38;\&\#xace0;**

[ST\\_DelaunayTriangles](#), [ST\\_VoronoiLines](#), [ST\\_GeomCollFromText](#)

## 8.15 Affine Transformations

### 8.15.1 ST\_Affine

**ST\_Affine** — Apply a 3D affine transformation to a geometry.

#### Synopsis

```
geometry ST_Affine(geometry geomA, float a, float b, float c, float d, float e, float f, float g, float h, float i, float xoff, float yoff,
float zoff);
geometry ST_Affine(geometry geomA, float a, float b, float d, float e, float xoff, float yoff);
```

#### Description

Applies a 3D affine transformation to the geometry to do things like translate, rotate, scale in one step.

Version 1: The call

```
ST_Affine(geom, a, b, c, d, e, f, g, h, i, xoff, yoff, zoff)
```

represents the transformation matrix

```

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

```

and the vertices are transformed as follows:

```

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

```

All of the translate / scale functions below are expressed via such an affine transformation.

Version 2: Applies a 2d affine transformation to the geometry. The call

```
ST_Affine(geom, a, b, d, e, xoff, yoff)
```

represents the 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 /

```

and the vertices are transformed as follows:

```

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

```

This method is a subcase of the 3D method above.

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

Availability: 1.1.2. Name changed from Affine to ST\_Affine in 1.2.2



#### Note

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



This function supports Polyhedral surfaces.



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



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



This method supports Circular Strings and Curves

#### Examples

```

--Rotate a 3d line 180 degrees about the z axis. Note this is long-hand for doing ↔
ST_Rotate();
SELECT ST_AseWKT(ST_Affine(geom, cos(pi()), -sin(pi()), 0, sin(pi()), cos(pi()), 0, 0, ↔
0, 1, 0, 0, 0)) As using_affine,
       ST_AseWKT(ST_Rotate(geom, pi())) As using_rotate
FROM (SELECT ST_GeomFromEWKT('LINESTRING(1 2 3, 1 4 3)') As geom) As foo;
using_affine | using_rotate

```

```

-----+-----
  LINESTRING(-1 -2 3,-1 -4 3) | LINESTRING(-1 -2 3,-1 -4 3)
(1 row)

--Rotate a 3d line 180 degrees in both the x and z axis
SELECT ST_AsEWKT(ST_Affine(geom, cos(pi()), -sin(pi()), 0, sin(pi()), cos(pi()), -sin(pi()) ←
  , 0, sin(pi()), cos(pi()), 0, 0, 0))
  FROM (SELECT ST_GeomFromEWKT('LINESTRING(1 2 3, 1 4 3)') As geom) As foo;
      st_asewkt
-----
  LINESTRING(-1 -2 -3,-1 -4 -3)
(1 row)

```

**See Also**

[ST\\_Rotate](#), [ST\\_Scale](#), [ST\\_Translate](#), [ST\\_TransScale](#)

**8.15.2 ST\_Rotate**

ST\_Rotate — Rotates a geometry about an origin point.

**Synopsis**

```

geometry ST_Rotate(geometry geomA, float rotRadians);
geometry ST_Rotate(geometry geomA, float rotRadians, float x0, float y0);
geometry ST_Rotate(geometry geomA, float rotRadians, geometry pointOrigin);

```

**Description**

Rotates geometry rotRadians counter-clockwise about the origin point. The rotation origin can be specified either as a POINT geometry, or as x and y coordinates. If the origin is not specified, the geometry is rotated about POINT(0 0).

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

Enhanced: 2.0.0 additional parameters for specifying the origin of rotation were added.

Availability: 1.1.2. Name changed from Rotate to ST\_Rotate in 1.2.2



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



This method supports Circular Strings and Curves



This function supports Polyhedral surfaces.



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

**Examples**

```

--Rotate 180 degrees
SELECT ST_AsEWKT(ST_Rotate('LINESTRING (50 160, 50 50, 100 50)', pi()));
      st_asewkt
-----
  LINESTRING(-50 -160,-50 -50,-100 -50)
(1 row)

```

```

--Rotate 30 degrees counter-clockwise at x=50, y=160
SELECT ST_AsEWKT(ST_Rotate('LINESTRING (50 160, 50 50, 100 50)', pi()/6, 50, 160));
      st_asewkt
-----
LINESTRING(50 160,105 64.7372055837117,148.301270189222 89.7372055837117)
(1 row)

--Rotate 60 degrees clockwise from centroid
SELECT ST_AsEWKT(ST_Rotate(geom, -pi()/3, ST_Centroid(geom)))
FROM (SELECT 'LINESTRING (50 160, 50 50, 100 50)::geometry AS geom) AS foo;
      st_asewkt
-----
LINESTRING(116.4225 130.6721,21.1597 75.6721,46.1597 32.3708)
(1 row)

```

**See Also**

[ST\\_Affine](#), [ST\\_RotateX](#), [ST\\_RotateY](#), [ST\\_RotateZ](#)

**8.15.3 ST\_RotateX**

ST\_RotateX — Rotates a geometry about the X axis.

**Synopsis**

geometry **ST\_RotateX**(geometry geomA, float rotRadians);

**Description**

Rotates a geometry geomA - rotRadians about the X axis.

**Note**

ST\_RotateX(geomA, rotRadians) is short-hand for ST\_Affine(geomA, 1, 0, 0, 0, cos(rotRadians), -sin(rotRadians), 0, sin(rotRadians), cos(rotRadians), 0, 0, 0).

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

Availability: 1.1.2. Name changed from RotateX to ST\_RotateX in 1.2.2



This function supports Polyhedral surfaces.



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



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

**Examples**

```

--Rotate a line 90 degrees along x-axis
SELECT ST_AsEWKT(ST_RotateX(ST_GeomFromEWKT('LINESTRING(1 2 3, 1 1 1)'), pi()/2));
      st_asewkt
-----
LINESTRING(1 -3 2,1 -1 1)

```

## See Also

[ST\\_Affine](#), [ST\\_RotateY](#), [ST\\_RotateZ](#)

### 8.15.4 ST\_RotateY

ST\_RotateY — Rotates a geometry about the Y axis.

#### Synopsis

geometry **ST\_RotateY**(geometry geomA, float rotRadians);

#### Description

Rotates a geometry geomA - rotRadians about the y axis.



#### Note

ST\_RotateY(geomA, rotRadians) is short-hand for ST\_Affine(geomA, cos(rotRadians), 0, sin(rotRadians), 0, 1, 0, -sin(rotRadians), 0, cos(rotRadians), 0, 0, 0).

Availability: 1.1.2. Name changed from RotateY to ST\_RotateY in 1.2.2

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



This function supports Polyhedral surfaces.



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



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

#### Examples

```
--Rotate a line 90 degrees along y-axis
SELECT ST_AsEWKT(ST_RotateY(ST_GeomFromEWKT('LINESTRING(1 2 3, 1 1 1)'), pi()/2));
           st_asewkt
-----
LINESTRING(3 2 -1,1 1 -1)
```

## See Also

[ST\\_Affine](#), [ST\\_RotateX](#), [ST\\_RotateZ](#)

### 8.15.5 ST\_RotateZ

ST\_RotateZ — Rotates a geometry about the Z axis.

#### Synopsis

geometry **ST\_RotateZ**(geometry geomA, float rotRadians);

## Description

Rotates a geometry geomA - rotRadians about the Z axis.



### Note

This is a synonym for ST\_Rotate



### Note

ST\_RotateZ(geomA, rotRadians) is short-hand for SELECT ST\_Affine(geomA, cos(rotRadians), -sin(rotRadians), 0, sin(rotRadians), cos(rotRadians), 0, 0, 0, 1, 0, 0, 0).

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

Availability: 1.1.2. Name changed from RotateZ to ST\_RotateZ in 1.2.2



### Note

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



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



This method supports Circular Strings and Curves



This function supports Polyhedral surfaces.



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

## Examples

```
--Rotate a line 90 degrees along z-axis
SELECT ST_AsEWKT(ST_RotateZ(ST_GeomFromEWKT('LINESTRING(1 2 3, 1 1 1)'), pi()/2));
      st_asewkt
-----
LINESTRING(-2 1 3,-1 1 1)

--Rotate a curved circle around z-axis
SELECT ST_AsEWKT(ST_RotateZ(geom, pi()/2))
FROM (SELECT ST_LineToCurve(ST_Buffer(ST_GeomFromText('POINT(234 567)'), 3)) As geom) As foo;

-----

CURVEPOLYGON(CIRCULARSTRING(-567 237,-564.87867965644 236.12132034356,-564 234,-569.12132034356 231.87867965644,-567 237))
```

## See Also

[ST\\_Affine](#), [ST\\_RotateX](#), [ST\\_RotateY](#)

## 8.15.6 ST\_Scale

`ST_Scale` — Scales a geometry by given factors.

### Synopsis

```
geometry ST_Scale(geometry geomA, float XFactor, float YFactor, float ZFactor);
geometry ST_Scale(geometry geomA, float XFactor, float YFactor);
geometry ST_Scale(geometry geom, geometry factor);
geometry ST_Scale(geometry geom, geometry factor, geometry origin);
```

### Description

Scales the geometry to a new size by multiplying the ordinates with the corresponding factor parameters.

The version taking a geometry as the `factor` parameter allows passing a 2d, 3dm, 3dz or 4d point to set scaling factor for all supported dimensions. Missing dimensions in the `factor` point are equivalent to no scaling the corresponding dimension.

The three-geometry variant allows a "false origin" for the scaling to be passed in. This allows "scaling in place", for example using the centroid of the geometry as the false origin. Without a false origin, scaling takes place relative to the actual origin, so all coordinates are just multiplied by the scale factor.



#### Note

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

---

Availability: 1.1.0.

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

Enhanced: 2.2.0 support for scaling all dimension (`factor` parameter) was introduced.

Enhanced: 2.5.0 support for scaling relative to a local origin (`origin` parameter) was introduced.



This function supports Polyhedral surfaces.



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



This method supports Circular Strings and Curves



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



This function supports M coordinates.

### Examples

---

```

--Version 1: scale X, Y, Z
SELECT ST_AsEWKT(ST_Scale(ST_GeomFromEWKT('LINESTRING(1 2 3, 1 1 1)'), 0.5, 0.75, 0.8));
          st_asewkt
-----
LINESTRING(0.5 1.5 2.4,0.5 0.75 0.8)

--Version 2: Scale X Y
SELECT ST_AsEWKT(ST_Scale(ST_GeomFromEWKT('LINESTRING(1 2 3, 1 1 1)'), 0.5, 0.75));
          st_asewkt
-----
LINESTRING(0.5 1.5 3,0.5 0.75 1)

--Version 3: Scale X Y Z M
SELECT ST_AsEWKT(ST_Scale(ST_GeomFromEWKT('LINESTRING(1 2 3 4, 1 1 1 1)'),
          ST_MakePoint(0.5, 0.75, 2, -1)));
          st_asewkt
-----
LINESTRING(0.5 1.5 6 -4,0.5 0.75 2 -1)

--Version 4: Scale X Y using false origin
SELECT ST_AsText(ST_Scale('LINESTRING(1 1, 2 2)', 'POINT(2 2)', 'POINT(1 1)::geometry));
          st_astext
-----
LINESTRING(1 1,3 3)

```

**See Also**

[ST\\_Affine](#), [ST\\_TransScale](#)

**8.15.7 ST\_Translate**

**ST\_Translate** — Translates a geometry by given offsets.

**Synopsis**

```

geometry ST_Translate(geometry g1, float deltax, float deltay);
geometry ST_Translate(geometry g1, float deltax, float deltay, float deltaxz);

```

**Description**

Returns a new geometry whose coordinates are translated delta x,delta y,delta z units. Units are based on the units defined in spatial reference (SRID) for this geometry.

**Note**

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

Availability: 1.2.2



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



This method supports Circular Strings and Curves

## Examples

### Move a point 1 degree longitude

```
SELECT ST_AsText(ST_Translate(ST_GeomFromText('POINT(-71.01 42.37)',4326),1,0)) As
wgs_transgeomtxt;

wgs_transgeomtxt
-----
POINT(-70.01 42.37)
```

### Move a linestring 1 degree longitude and 1/2 degree latitude

```
SELECT ST_AsText(ST_Translate(ST_GeomFromText('LINESTRING(-71.01 42.37,-71.11 42.38)',4326)
,1,0.5)) As wgs_transgeomtxt;
wgs_transgeomtxt
-----
LINESTRING(-70.01 42.87,-70.11 42.88)
```

### Move a 3d point

```
SELECT ST_AsEWKT(ST_Translate(CAST('POINT(0 0 0)' As geometry), 5, 12,3));
st_asewkt
-----
POINT(5 12 3)
```

### Move a curve and a point

```
SELECT ST_AsText(ST_Translate(ST_Collect('CURVEPOLYGON(CIRCULARSTRING(4 3,3.12 0.878,1
0,-1.121 5.1213,6 7, 8 9,4 3))','POINT(1 3)'),1,2));
-----
GEOMETRYCOLLECTION(CURVEPOLYGON(CIRCULARSTRING(5 5,4.12 2.878,2 2,-0.121 7.1213,7 9,9 11,5
5)),POINT(2 5))
```

## See Also

[ST\\_Affine](#), [ST\\_AsText](#), [ST\\_GeomFromText](#)

## 8.15.8 ST\_TransScale

**ST\_TransScale** — Translates and scales a geometry by given offsets and factors.

### Synopsis

geometry **ST\_TransScale**(geometry geomA, float deltaX, float deltaY, float XFactor, float YFactor);

### Description

Translates the geometry using the deltaX and deltaY args, then scales it using the XFactor, YFactor args, working in 2D only.



#### Note

**ST\_TransScale**(geomA, deltaX, deltaY, XFactor, YFactor) is short-hand for **ST\_Affine**(geomA, XFactor, 0, 0, 0, YFactor, 0, 0, 0, 1, deltaX\*XFactor, deltaY\*YFactor, 0).

**Note**

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

Availability: 1.1.0.



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



This method supports Circular Strings and Curves

**Examples**

```
SELECT ST_AsEWKT(ST_TransScale(ST_GeomFromEWKT('LINESTRING(1 2 3, 1 1 1)'), 0.5, 1, 1, 2));
           st_asewkt
```

```
-----
LINESTRING(1.5 6 3,1.5 4 1)
```

```
--Buffer a point to get an approximation of a circle, convert to curve and then translate ↵
  1,2 and scale it 3,4
```

```
SELECT ST_AsText(ST_Transscale(ST_LineToCurve(ST_Buffer('POINT(234 567)', 3)),1,2,3,4));
```

```
-----
CURVEPOLYGON(CIRCULARSTRING(714 2276,711.363961030679 2267.51471862576,705 ↵
  2264,698.636038969321 2284.48528137424,714 2276))
```

**See Also**

[ST\\_Affine](#), [ST\\_Translate](#)

## 8.16 Clustering Functions

### 8.16.1 ST\_ClusterDBSCAN

`ST_ClusterDBSCAN` — Window function that returns a cluster id for each input geometry using the DBSCAN algorithm.

**Synopsis**

```
integer ST_ClusterDBSCAN(geometry winset geom, float8 eps, integer minpoints);
```

**Description**

Returns cluster number for each input geometry, based on a 2D implementation of the [Density-based spatial clustering of applications with noise \(DBSCAN\)](#) algorithm. Unlike `ST_ClusterKMeans`, it does not require the number of clusters to be specified, but instead uses the desired **distance** (`eps`) and density (`minpoints`) parameters to construct each cluster.

An input geometry will be added to a cluster if it is either:

- A "core" geometry, that is within `eps distance` of at least `minpoints` input geometries (including itself) or
- A "border" geometry, that is within `eps distance` of a core geometry.

Note that border geometries may be within `eps` distance of core geometries in more than one cluster; in this case, either assignment would be correct, and the border geometry will be arbitrarily assigned to one of the available clusters. In these cases, it is possible for a correct cluster to be generated with fewer than `minpoints` geometries. When assignment of a border geometry is ambiguous, repeated calls to `ST_ClusterDBSCAN` will produce identical results if an `ORDER BY` clause is included in the window definition, but cluster assignments may differ from other implementations of the same algorithm.

**Note**

Input geometries that do not meet the criteria to join any other cluster will be assigned a cluster number of `NULL`.

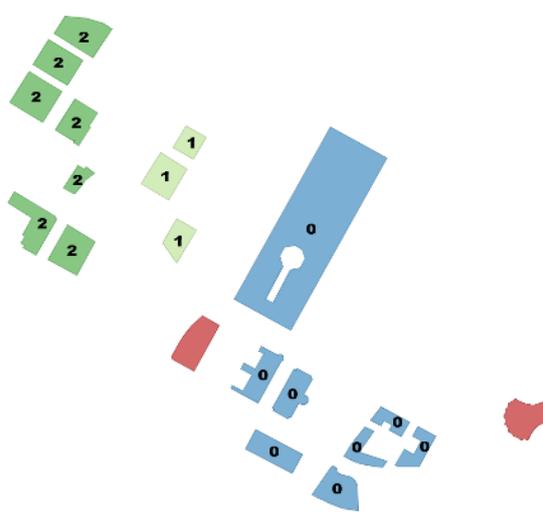
---

Availability: 2.3.0

**Examples**

Assigning a cluster number to each polygon within 50 meters of each other. Require at least 2 polygons per cluster

---



*within 50 meters at least 2 per cluster. singletons have NULL for cid*

```

SELECT name, ST_ClusterDBSCAN(geom, eps ←
:= 50, minpoints := 2) over () AS cid
FROM boston_polys
WHERE name > '' AND building > ''
AND ST_DWithin(geom,
ST_Transform(
ST_GeomFromText('POINT ←
(-71.04054 42.35141)', 4326), 26986),
500);
        
```

| bucket                              | name |  | ↔ |
|-------------------------------------|------|--|---|
| Manulife Tower                      |      |  | ↔ |
| 0                                   |      |  |   |
| Park Lane Seaport I                 |      |  | ↔ |
| 0                                   |      |  |   |
| Park Lane Seaport II                |      |  | ↔ |
| 0                                   |      |  |   |
| Renaissance Boston Waterfront Hotel |      |  | ↔ |
| 0                                   |      |  |   |
| Seaport Boston Hotel                |      |  | ↔ |
| 0                                   |      |  |   |
| Seaport Hotel & World Trade Center  |      |  | ↔ |
| 0                                   |      |  |   |
| Waterside Place                     |      |  | ↔ |
| 0                                   |      |  |   |
| World Trade Center East             |      |  | ↔ |
| 0                                   |      |  |   |
| 100 Northern Avenue                 |      |  | ↔ |
| 1                                   |      |  |   |
| 100 Pier 4                          |      |  | ↔ |
| 1                                   |      |  |   |
| The Institute of Contemporary Art   |      |  | ↔ |
| 1                                   |      |  |   |
| 101 Seaport                         |      |  | ↔ |
| 2                                   |      |  |   |
| District Hall                       |      |  | ↔ |
| 2                                   |      |  |   |
| One Marina Park Drive               |      |  | ↔ |
| 2                                   |      |  |   |
| Twenty Two Liberty                  |      |  | ↔ |
| 2                                   |      |  |   |
| Vertex                              |      |  | ↔ |
| 2                                   |      |  |   |
| Vertex                              |      |  | ↔ |
| 2                                   |      |  |   |
| Watermark Seaport                   |      |  | ↔ |
| 2                                   |      |  |   |
| Blue Hills Bank Pavilion            |      |  | ↔ |
| NULL                                |      |  |   |
| World Trade Center West             |      |  | ↔ |
| NULL                                |      |  |   |
| (20 rows)                           |      |  |   |

Combining parcels with the same cluster number into a single geometry. This uses named argument calling

```

SELECT cid, ST_Collect(geom) AS cluster_geom, array_agg(parcel_id) AS ids_in_cluster FROM (
SELECT parcel_id, ST_ClusterDBSCAN(geom, eps := 0.5, minpoints := 5) over () AS cid, ←
geom
FROM parcels) sq
GROUP BY cid;
        
```

**See Also**

[ST\\_DWithin](#), [ST\\_ClusterKMeans](#), [ST\\_ClusterIntersecting](#), [ST\\_ClusterWithin](#)

## 8.16.2 ST\_ClusterIntersecting

`ST_ClusterIntersecting` — Aggregate function that clusters the input geometries into connected sets.

### Synopsis

```
geometry[] ST_ClusterIntersecting(geometry set g);
```

### Description

`ST_ClusterIntersecting` is an aggregate function that returns an array of `GeometryCollections`, where each `GeometryCollection` represents an interconnected set of geometries.

Availability: 2.2.0

### Examples

```
WITH testdata AS
  (SELECT unnest(ARRAY['LINESTRING (0 0, 1 1)::geometry',
                      'LINESTRING (5 5, 4 4)::geometry',
                      'LINESTRING (6 6, 7 7)::geometry',
                      'LINESTRING (0 0, -1 -1)::geometry',
                      'POLYGON ((0 0, 4 0, 4 4, 0 4, 0 0))::geometry'] AS geom)

SELECT ST_AsText(unnest(ST_ClusterIntersecting(geom))) FROM testdata;

--result

st_astext
-----
GEOMETRYCOLLECTION(LINESTRING(0 0,1 1),LINESTRING(5 5,4 4),LINESTRING(0 0,-1 -1),POLYGON((0 ←
  0,4 0,4 4,0 4,0 0)))
GEOMETRYCOLLECTION(LINESTRING(6 6,7 7))
```

### See Also

[ST\\_ClusterDBSCAN](#), [ST\\_ClusterKMeans](#), [ST\\_ClusterWithin](#)

## 8.16.3 ST\_ClusterKMeans

`ST_ClusterKMeans` — Window function that returns a cluster id for each input geometry using the K-means algorithm.

### Synopsis

```
integer ST_ClusterKMeans(geometry winset geom, integer number_of_clusters, float max_radius);
```

### Description

Returns **K-means** cluster number for each input geometry. The distance used for clustering is the distance between the centroids for 2D geometries, and distance between bounding box centers for 3D geometries. For `POINT` inputs, `M` coordinate will be treated as weight of input and has to be larger than 0.

`max_radius`, if set, will cause `ST_ClusterKMeans` to generate more clusters than `k` ensuring that no cluster in output has radius larger than `max_radius`. This is useful in reachability analysis.

Enhanced: 3.2.0 Support for `max_radius`

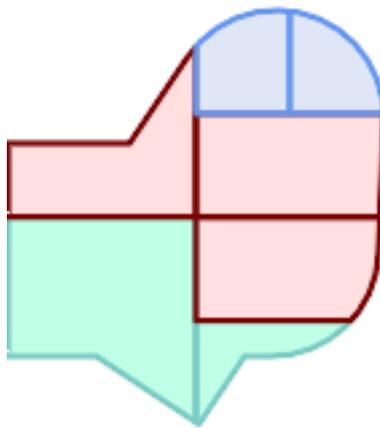
Enhanced: 3.1.0 Support for 3D geometries and weights

Availability: 2.3.0

## Examples

Generate dummy set of parcels for examples:

```
CREATE TABLE parcels AS
SELECT lpad((row_number() over())::text,3,'0') As parcel_id, geom,
('{residential, commercial}'::text[])[1 + mod(row_number()OVER(),2)] As type
FROM
  ST_Subdivide(ST_Buffer('SRID=3857;LINESTRING(40 100, 98 100, 100 150, 60 90)'::geometry ←
  40, 'endcap=square'),12) As geom;
```



*Parcels color-coded by cluster number (cid)*

```
SELECT ST_ClusterKMeans(geom, 3) OVER() AS cid, parcel_id, geom
FROM parcels;
```

| cid | parcel_id | geom          |
|-----|-----------|---------------|
| 0   | 001       | 0103000000... |
| 0   | 002       | 0103000000... |
| 1   | 003       | 0103000000... |
| 0   | 004       | 0103000000... |
| 1   | 005       | 0103000000... |
| 2   | 006       | 0103000000... |
| 2   | 007       | 0103000000... |

Partitioning parcel clusters by type:

```
SELECT ST_ClusterKMeans(geom, 3) over (PARTITION BY type) AS cid, parcel_id, type
FROM parcels;
```

| cid | parcel_id | type        |
|-----|-----------|-------------|
| 1   | 005       | commercial  |
| 1   | 003       | commercial  |
| 2   | 007       | commercial  |
| 0   | 001       | commercial  |
| 1   | 004       | residential |
| 0   | 002       | residential |
| 2   | 006       | residential |

**Example:** Clustering a preaggregated planetary-scale data population dataset using 3D clustering and weighting. Identify at least 20 regions based on **Kontur Population Data** that do not span more than 3000 km from their center:

```
create table kontur_population_3000km_clusters as
select
  geom,
  ST_ClusterKMeans(
    ST_Force4D(
      ST_Transform(ST_Force3D(geom), 4978), -- cluster in 3D XYZ CRS
      mvalue := population -- set clustering to be weighed by population
    ),
    20, -- aim to generate at least 20 clusters
    max_radius := 3000000 -- but generate more to make each under 3000 km radius
  ) over () as cid
from
  kontur_population;
```



*World population clustered to above specs produces 46 clusters. Clusters are centered at well-populated regions (New York, Moscow). Greenland is one cluster. There are island clusters that span across the antimeridian. Cluster edges follow Earth's curvature.*

### See Also

[ST\\_ClusterDBSCAN](#), [ST\\_ClusterIntersecting](#), [ST\\_ClusterWithin](#), [ST\\_Subdivide](#), [ST\\_Force3D](#), [ST\\_Force4D](#),

## 8.16.4 ST\_ClusterWithin

`ST_ClusterWithin` — Aggregate function that clusters the input geometries by separation distance.

### Synopsis

```
geometry[] ST_ClusterWithin(geometry set g, float8 distance);
```

## Description

`ST_ClusterWithin` is an aggregate function that returns an array of `GeometryCollections`, where each `GeometryCollection` represents a set of geometries separated by no more than the specified distance. (Distances are Cartesian distances in the units of the SRID.)

Availability: 2.2.0

## Examples

```
WITH testdata AS
  (SELECT unnest(ARRAY['LINESTRING (0 0, 1 1)::geometry,
                      'LINESTRING (5 5, 4 4)::geometry,
                      'LINESTRING (6 6, 7 7)::geometry,
                      'LINESTRING (0 0, -1 -1)::geometry,
                      'POLYGON ((0 0, 4 0, 4 4, 0 4, 0 0))::geometry']) AS geom)

SELECT ST_AsText(unnest(ST_ClusterWithin(geom, 1.4))) FROM testdata;

--result

st_astext
-----
GEOMETRYCOLLECTION(LINESTRING(0 0,1 1),LINESTRING(5 5,4 4),LINESTRING(0 0,-1 -1),POLYGON((0 0,4 0,4 4,0 4,0 0)))
GEOMETRYCOLLECTION(LINESTRING(6 6,7 7))
```

## See Also

[ST\\_ClusterDBSCAN](#), [ST\\_ClusterKMeans](#), [ST\\_ClusterIntersecting](#)

## 8.17 Bounding Box Functions

### 8.17.1 Box2D

`Box2D` — Returns a `BOX2D` representing the 2D extent of a geometry.

#### Synopsis

```
box2d Box2D(geometry geom);
```

#### Description

Returns a `box2d` representing the 2D extent of the geometry.

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



This method supports Circular Strings and Curves



This function supports Polyhedral surfaces.



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

## Examples

```
SELECT Box2D(ST_GeomFromText('LINESTRING(1 2, 3 4, 5 6)'));
```

```
box2d
```

```
-----
```

```
BOX(1 2,5 6)
```

```
SELECT Box2D(ST_GeomFromText('CIRCULARSTRING(220268 150415,220227 150505,220227 150406)'));
```

```
box2d
```

```
-----
```

```
BOX(220186.984375 150406,220288.25 150506.140625)
```

## See Also

[Box3D](#), [ST\\_GeomFromText](#)

### 8.17.2 Box3D

Box3D — Returns a BOX3D representing the 3D extent of a geometry.

#### Synopsis

```
box3d Box3D(geometry geom);
```

#### Description

Returns a **box3d** representing the 3D extent of the geometry.

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



This method supports Circular Strings and Curves



This function supports Polyhedral surfaces.



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



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

## Examples

```
SELECT Box3D(ST_GeomFromEWKT('LINESTRING(1 2 3, 3 4 5, 5 6 5)'));
```

```
Box3d
```

```
-----
```

```
BOX3D(1 2 3,5 6 5)
```

```
SELECT Box3D(ST_GeomFromEWKT('CIRCULARSTRING(220268 150415 1,220227 150505 1,220227 150406 ←
1)'));
```

```
Box3d
```

```
-----
```

```
BOX3D(220227 150406 1,220268 150415 1)
```

## See Also

[Box2D](#), [ST\\_GeomFromEWKT](#)

### 8.17.3 ST\_EstimatedExtent

`ST_EstimatedExtent` — Returns the estimated extent of a spatial table.

#### Synopsis

```
box2d ST_EstimatedExtent(text schema_name, text table_name, text geocolumn_name, boolean parent_only);
box2d ST_EstimatedExtent(text schema_name, text table_name, text geocolumn_name);
box2d ST_EstimatedExtent(text table_name, text geocolumn_name);
```

#### Description

Returns the estimated extent of a spatial table as a [box2d](#). The current schema is used if not specified. The estimated extent is taken from the geometry column's statistics. This is usually much faster than computing the exact extent of the table using [ST\\_Extent](#) or [ST\\_3DExtent](#).

The default behavior is to also use statistics collected from child tables (tables with `INHERITS`) if available. If `parent_only` is set to `TRUE`, only statistics for the given table are used and child tables are ignored.

For PostgreSQL  $\geq 8.0.0$  statistics are gathered by `VACUUM ANALYZE` and the result extent will be about 95% of the actual one. For PostgreSQL  $< 8.0.0$  statistics are gathered by running `update_geometry_stats()` and the result extent is exact.



#### Note

In the absence of statistics (empty table or no `ANALYZE` called) this function returns `NULL`. Prior to version 1.5.4 an exception was thrown instead.

Availability: 1.0.0

Changed: 2.1.0. Up to 2.0.x this was called `ST_Estimated_Extent`.



This method supports Circular Strings and Curves

#### Examples

```
SELECT ST_EstimatedExtent('ny', 'edges', 'geom');
--result--
BOX(-8877653 4912316,-8010225.5 5589284)

SELECT ST_EstimatedExtent('feature_poly', 'geom');
--result--
BOX(-124.659652709961 24.6830825805664,-67.7798080444336 49.0012092590332)
```

## See Also

[ST\\_Extent](#), [ST\\_3DExtent](#)

### 8.17.4 ST\_Expand

`ST_Expand` — Returns a bounding box expanded from another bounding box or a geometry.

## Synopsis

```
geometry ST_Expand(geometry geom, float units_to_expand);
geometry ST_Expand(geometry geom, float dx, float dy, float dz=0, float dm=0);
box2d ST_Expand(box2d box, float units_to_expand);
box2d ST_Expand(box2d box, float dx, float dy);
box3d ST_Expand(box3d box, float units_to_expand);
box3d ST_Expand(box3d box, float dx, float dy, float dz=0);
```

## Description

Returns a bounding box expanded from the bounding box of the input, either by specifying a single distance with which the box should be expanded on both axes, or by specifying an expansion distance for each axis. Uses double-precision. Can be used for distance queries, or to add a bounding box filter to a query to take advantage of a spatial index.

In addition to the version of `ST_Expand` accepting and returning a geometry, variants are provided that accept and return `box2d` and `box3d` data types.

Distances are in the units of the spatial reference system of the input.

`ST_Expand` is similar to `ST_Buffer`, except while buffering expands a geometry in all directions, `ST_Expand` expands the bounding box along each axis.



### Note

Pre version 1.3, `ST_Expand` was used in conjunction with `ST_Distance` to do indexable distance queries. For example, `geom && ST_Expand('POINT(10 20)', 10) AND ST_Distance(geom, 'POINT(10 20)') < 10`. This has been replaced by the simpler and more efficient `ST_DWithin` function.

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

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

Enhanced: 2.3.0 support was added to expand a box by different amounts in different dimensions.



This function supports Polyhedral surfaces.



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

## Examples



### Note

Examples below use US National Atlas Equal Area (SRID=2163) which is a meter projection

```
--10 meter expanded box around bbox of a linestring
SELECT CAST(ST_Expand(ST_GeomFromText('LINESTRING(2312980 110676,2312923 110701,2312892  ←
  110714)', 2163),10) As box2d);
                                st_expand
-----
BOX(2312882 110666,2312990 110724)

--10 meter expanded 3D box of a 3D box
SELECT ST_Expand(CAST('BOX3D(778783 2951741 1,794875 2970042.61545891 10)' As box3d),10)
                                st_expand
-----
```

```

BOX3D(778773 2951731 -9,794885 2970052.61545891 20)

--10 meter geometry as text rep of a expand box around a point geometry
SELECT ST_AsEWKT(ST_Expand(ST_GeomFromEWKT('SRID=2163;POINT(2312980 110676)'),10));
-----
SRID=2163;POLYGON((2312970 110666,2312970 110686,2312990 110686,2312990 110666,2312970
110666))

```

## See Also

[ST\\_Buffer](#), [ST\\_DWithin](#), [ST\\_SRID](#)

## 8.17.5 ST\_Extent

`ST_Extent` — Aggregate function that returns the bounding box of geometries.

### Synopsis

```
box2d ST_Extent(geometry set geomfield);
```

### Description

An aggregate function that returns a **box2d** bounding box that bounds a set of geometries.

The bounding box coordinates are in the spatial reference system of the input geometries.

`ST_Extent` is similar in concept to Oracle Spatial/Locator's `SDO_AGGR_MBR`.



#### Note

`ST_Extent` returns boxes with only X and Y ordinates even with 3D geometries. To return XYZ ordinates use [ST\\_3DExtent](#).



#### Note

The returned `box3d` value does not include a SRID. Use [ST\\_SetSRID](#) to convert it into a geometry with SRID metadata. The SRID is the same as the input geometries.

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



This function supports Polyhedral surfaces.



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

## Examples



### Note

Examples below use Massachusetts State Plane ft (SRID=2249)

```
SELECT ST_Extent(geom) as bextent FROM sometable;
                                st_bextent
-----
BOX(739651.875 2908247.25,794875.8125 2970042.75)

--Return extent of each category of geometries
SELECT ST_Extent(geom) as bextent
FROM sometable
GROUP BY category ORDER BY category;

                                bextent                                |          name
-----+-----
BOX(778783.5625 2951741.25,794875.8125 2970042.75) | A
BOX(751315.8125 2919164.75,765202.6875 2935417.25) | B
BOX(739651.875 2917394.75,756688.375 2935866)      | C

--Force back into a geometry
-- and render the extended text representation of that geometry
SELECT ST_SetSRID(ST_Extent(geom),2249) as bextent FROM sometable;

                                bextent
-----
SRID=2249;POLYGON((739651.875 2908247.25,739651.875 2970042.75,794875.8125 2970042.75,
794875.8125 2908247.25,739651.875 2908247.25))
```

## See Also

[ST\\_EstimatedExtent](#), [ST\\_3DExtent](#), [ST\\_SetSRID](#)

## 8.17.6 ST\_3DExtent

**ST\_3DExtent** — Aggregate function that returns the 3D bounding box of geometries.

### Synopsis

```
box3d ST_3DExtent(geometry set geomfield);
```

### Description

An aggregate function that returns a **box3d** (includes Z ordinate) bounding box that bounds a set of geometries.

The bounding box coordinates are in the spatial reference system of the input geometries.



### Note

The returned **box3d** value does not include a SRID. Use [ST\\_SetSRID](#) to convert it into a geometry with SRID meta-data. The SRID is the same as the input geometries.

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

Changed: 2.0.0 In prior versions this used to be called ST\_Extent3D

-  This function supports 3d and will not drop the z-index.
-  This method supports Circular Strings and Curves
-  This function supports Polyhedral surfaces.
-  This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).

## Examples

```
SELECT ST_3DExtent(foo.geom) As b3extent
FROM (SELECT ST_MakePoint(x,y,z) As geom
      FROM generate_series(1,3) As x
           CROSS JOIN generate_series(1,2) As y
           CROSS JOIN generate_series(0,2) As Z) As foo;
      b3extent
-----
BOX3D(1 1 0,3 2 2)

--Get the extent of various elevated circular strings
SELECT ST_3DExtent(foo.geom) As b3extent
FROM (SELECT ST_Translate(ST_Force_3DZ(ST_LineToCurve(ST_Buffer(ST_Point(x,y),1))),0,0,z) ←
      As geom
      FROM generate_series(1,3) As x
           CROSS JOIN generate_series(1,2) As y
           CROSS JOIN generate_series(0,2) As Z) As foo;
      b3extent
-----
BOX3D(1 0 0,4 2 2)
```

## See Also

[ST\\_Extent](#), [ST\\_Force3DZ](#), [ST\\_SetSRID](#)

### 8.17.7 ST\_MakeBox2D

ST\_MakeBox2D — Creates a BOX2D defined by two 2D point geometries.

#### Synopsis

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

#### Description

Creates a [box2d](#) defined by two Point geometries. This is useful for doing range queries.

## Examples

```
--Return all features that fall reside or partly reside in a US national atlas coordinate
-- bounding box
--It is assumed here that the geometries are stored with SRID = 2163 (US National atlas
-- equal area)
SELECT feature_id, feature_name, geom
FROM features
WHERE geom && ST_SetSRID(ST_MakeBox2D(ST_Point(-989502.1875, 528439.5625),
    ST_Point(-987121.375 , 529933.1875)),2163)
```

## See Also

[ST\\_Point](#), [ST\\_SetSRID](#), [ST\\_SRID](#)

## 8.17.8 ST\_3DMakeBox

**ST\_3DMakeBox** — Creates a BOX3D defined by two 3D point geometries.

### Synopsis

```
box3d ST_3DMakeBox(geometry point3DLeftBottom, geometry point3DUpRightTop);
```

### Description

Creates a **box3d** defined by two 3D Point geometries.



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

Changed: 2.0.0 In prior versions this used to be called `ST_MakeBox3D`

## Examples

```
SELECT ST_3DMakeBox(ST_MakePoint(-989502.1875, 528439.5625, 10),
    ST_MakePoint(-987121.375 , 529933.1875, 10)) As abb3d

--bb3d--
-----
BOX3D(-989502.1875 528439.5625 10,-987121.375 529933.1875 10)
```

## See Also

[ST\\_MakePoint](#), [ST\\_SetSRID](#), [ST\\_SRID](#)

## 8.17.9 ST\_XMax

**ST\_XMax** — Returns the X maxima of a 2D or 3D bounding box or a geometry.

### Synopsis

```
float ST_XMax(box3d aGeomorBox2DorBox3D);
```

## Description

Returns the X maxima of a 2D or 3D bounding box or a geometry.



### Note

Although this function is only defined for `box3d`, it also works for `box2d` and geometry values due to automatic casting. However, it will not accept a geometry or `box2d` text representation, since those do not auto-cast.



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



This method supports Circular Strings and Curves

## Examples

```
SELECT ST_XMax('BOX3D(1 2 3, 4 5 6)');
st_xmax
-----
4

SELECT ST_XMax(ST_GeomFromText('LINESTRING(1 3 4, 5 6 7)'));
st_xmax
-----
5

SELECT ST_XMax(CAST('BOX(-3 2, 3 4)' As box2d));
st_xmax
-----
3
--Observe THIS DOES NOT WORK because it will try to auto-cast the string representation to
a BOX3D
SELECT ST_XMax('LINESTRING(1 3, 5 6)');

--ERROR: BOX3D parser - doesn't start with BOX3D(

SELECT ST_XMax(ST_GeomFromEWKT('CIRCULARSTRING(220268 150415 1,220227 150505 2,220227
150406 3)'));
st_xmax
-----
220288.248780547
```

## See Also

[ST\\_XMin](#), [ST\\_YMax](#), [ST\\_YMin](#), [ST\\_ZMax](#), [ST\\_ZMin](#)

### 8.17.10 ST\_XMin

`ST_XMin` — Returns the X minima of a 2D or 3D bounding box or a geometry.

## Synopsis

```
float ST_XMin(box3d aGeomorBox2DorBox3D);
```

## Description

Returns the X minima of a 2D or 3D bounding box or a geometry.



### Note

Although this function is only defined for box3d, it also works for box2d and geometry values due to automatic casting. However it will not accept a geometry or box2d text representation, since those do not auto-cast.



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



This method supports Circular Strings and Curves

## Examples

```
SELECT ST_XMin('BOX3D(1 2 3, 4 5 6)');
st_xmin
-----
1

SELECT ST_XMin(ST_GeomFromText('LINESTRING(1 3 4, 5 6 7)'));
st_xmin
-----
1

SELECT ST_XMin(CAST('BOX(-3 2, 3 4)' As box2d));
st_xmin
-----
-3
--Observe THIS DOES NOT WORK because it will try to auto-cast the string representation to ←
a BOX3D
SELECT ST_XMin('LINESTRING(1 3, 5 6)');

--ERROR: BOX3D parser - doesn't start with BOX3D(

SELECT ST_XMin(ST_GeomFromEWKT('CIRCULARSTRING(220268 150415 1,220227 150505 2,220227 ←
150406 3)'));
st_xmin
-----
220186.995121892
```

## See Also

[ST\\_XMax](#), [ST\\_YMax](#), [ST\\_YMin](#), [ST\\_ZMax](#), [ST\\_ZMin](#)

### 8.17.11 ST\_YMax

ST\_YMax — Returns the Y maxima of a 2D or 3D bounding box or a geometry.

## Synopsis

```
float ST_YMax(box3d aGeomorBox2DorBox3D);
```

## Description

Returns the Y maxima of a 2D or 3D bounding box or a geometry.



### Note

Although this function is only defined for box3d, it also works for box2d and geometry values due to automatic casting. However it will not accept a geometry or box2d text representation, since those do not auto-cast.



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



This method supports Circular Strings and Curves

## Examples

```

SELECT ST_YMax('BOX3D(1 2 3, 4 5 6)');
st_ymax
-----
5

SELECT ST_YMax(ST_GeomFromText('LINESTRING(1 3 4, 5 6 7)'));
st_ymax
-----
6

SELECT ST_YMax(CAST('BOX(-3 2, 3 4)' As box2d));
st_ymax
-----
4
--Observe THIS DOES NOT WORK because it will try to auto-cast the string representation to
a BOX3D
SELECT ST_YMax('LINESTRING(1 3, 5 6)');

--ERROR: BOX3D parser - doesn't start with BOX3D(

SELECT ST_YMax(ST_GeomFromEWKT('CIRCULARSTRING(220268 150415 1,220227 150505 2,220227
150406 3)'));
st_ymax
-----
150506.126829327

```

## See Also

[ST\\_XMin](#), [ST\\_XMax](#), [ST\\_YMin](#), [ST\\_ZMax](#), [ST\\_ZMin](#)

### 8.17.12 ST\_YMin

**ST\_YMin** — Returns the Y minima of a 2D or 3D bounding box or a geometry.

## Synopsis

```
float ST_YMin(box3d aGeomorBox2DorBox3D);
```

## Description

Returns the Y minima of a 2D or 3D bounding box or a geometry.



### Note

Although this function is only defined for box3d, it also works for box2d and geometry values due to automatic casting. However it will not accept a geometry or box2d text representation, since those do not auto-cast.



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



This method supports Circular Strings and Curves

## Examples

```
SELECT ST_YMin('BOX3D(1 2 3, 4 5 6)');
st_ymin
-----
2

SELECT ST_YMin(ST_GeomFromText('LINESTRING(1 3 4, 5 6 7)'));
st_ymin
-----
3

SELECT ST_YMin(CAST('BOX(-3 2, 3 4)' As box2d));
st_ymin
-----
2
--Observe THIS DOES NOT WORK because it will try to auto-cast the string representation to ←
a BOX3D
SELECT ST_YMin('LINESTRING(1 3, 5 6)');

--ERROR: BOX3D parser - doesn't start with BOX3D(

SELECT ST_YMin(ST_GeomFromEWKT('CIRCULARSTRING(220268 150415 1,220227 150505 2,220227 ←
150406 3)'));
st_ymin
-----
150406
```

## See Also

[ST\\_GeomFromEWKT](#), [ST\\_XMin](#), [ST\\_XMax](#), [ST\\_YMax](#), [ST\\_ZMax](#), [ST\\_ZMin](#)

### 8.17.13 ST\_ZMax

**ST\_ZMax** — Returns the Z maxima of a 2D or 3D bounding box or a geometry.

## Synopsis

```
float ST_ZMax(box3d aGeomorBox2DorBox3D);
```

## Description

Returns the Z maxima of a 2D or 3D bounding box or a geometry.



### Note

Although this function is only defined for `box3d`, it also works for `box2d` and geometry values due to automatic casting. However it will not accept a geometry or `box2d` text representation, since those do not auto-cast.



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



This method supports Circular Strings and Curves

## Examples

```
SELECT ST_ZMax('BOX3D(1 2 3, 4 5 6)');
st_zmax
-----
6

SELECT ST_ZMax(ST_GeomFromEWKT('LINESTRING(1 3 4, 5 6 7)'));
st_zmax
-----
7

SELECT ST_ZMax('BOX3D(-3 2 1, 3 4 1) ');
st_zmax
-----
1
--Observe THIS DOES NOT WORK because it will try to auto-cast the string representation to ←
a BOX3D
SELECT ST_ZMax('LINESTRING(1 3 4, 5 6 7)');

--ERROR: BOX3D parser - doesn't start with BOX3D(

SELECT ST_ZMax(ST_GeomFromEWKT('CIRCULARSTRING(220268 150415 1,220227 150505 2,220227 ←
150406 3)'));
st_zmax
-----
3
```

## See Also

[ST\\_GeomFromEWKT](#), [ST\\_XMin](#), [ST\\_XMax](#), [ST\\_YMax](#), [ST\\_YMin](#), [ST\\_ZMax](#)

### 8.17.14 ST\_ZMin

`ST_ZMin` — Returns the Z minima of a 2D or 3D bounding box or a geometry.

## Synopsis

```
float ST_ZMin(box3d aGeomorBox2DorBox3D);
```

## Description

Returns the Z minima of a 2D or 3D bounding box or a geometry.



### Note

Although this function is only defined for box3d, it also works for box2d and geometry values due to automatic casting. However it will not accept a geometry or box2d text representation, since those do not auto-cast.



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



This method supports Circular Strings and Curves

## Examples

```

SELECT ST_ZMin('BOX3D(1 2 3, 4 5 6)');
st_zmin
-----
3

SELECT ST_ZMin(ST_GeomFromEWKT('LINESTRING(1 3 4, 5 6 7)'));
st_zmin
-----
4

SELECT ST_ZMin('BOX3D(-3 2 1, 3 4 1)');
st_zmin
-----
1
--Observe THIS DOES NOT WORK because it will try to auto-cast the string representation to ←
  a BOX3D
SELECT ST_ZMin('LINESTRING(1 3 4, 5 6 7)');

--ERROR: BOX3D parser - doesn't start with BOX3D(

SELECT ST_ZMin(ST_GeomFromEWKT('CIRCULARSTRING(220268 150415 1,220227 150505 2,220227 ←
  150406 3)'));
st_zmin
-----
1

```

## See Also

[ST\\_GeomFromEWKT](#), [ST\\_GeomFromText](#), [ST\\_XMin](#), [ST\\_XMax](#), [ST\\_YMax](#), [ST\\_YMin](#), [ST\\_ZMax](#)

## 8.18 Linear Referencing

### 8.18.1 ST\_LineInterpolatePoint

`ST_LineInterpolatePoint` — Returns a point interpolated along a line at a fractional location.

#### Synopsis

```
geometry ST_LineInterpolatePoint(geometry a_linestring, float8 a_fraction);
```



```
-- The point 20% along a line

SELECT ST_AsEWKT( ST_LineInterpolatePoint (
    'LINESTRING(25 50, 100 125, 150 190)',
    0.2 ));
-----
POINT(51.5974135047432 76.5974135047432)
```

The mid-point of a 3D line:

```
SELECT ST_AsEWKT( ST_LineInterpolatePoint ('
    LINESTRING(1 2 3, 4 5 6, 6 7 8)',
    0.5 ));
-----
POINT(3.5 4.5 5.5)
```

The closest point on a line to a point:

```
SELECT ST_AsText( ST_LineInterpolatePoint( line.geom,
    ST_LineLocatePoint( line.geom, 'POINT(4 3)'))
FROM (SELECT ST_GeomFromText('LINESTRING(1 2, 4 5, 6 7)') As geom) AS line;
-----
POINT(3 4)
```

**See also:**

[ST\\_LineInterpolatePoints](#), [ST\\_LineInterpolatePoint](#), [ST\\_LineMerge](#)

## 8.18.2 ST\_LineInterpolatePoint

`ST_LineInterpolatePoint` — Returns a point interpolated along a 3D line at a fractional location.

### Synopsis

geometry `ST_LineInterpolatePoint`(geometry a\_linestring, float8 a\_fraction);

**Parameters:**

`a_linestring`: A 2D or 3D linestring geometry.  
`a_fraction`: A float8 value representing the fraction of the line to interpolate to, ranging from 0 to 1. A value of 0 returns the first point, and a value of 1 returns the last point.



#### Note

`ST_LineInterpolatePoint` computes points in 2D and then interpolates the values for Z and M, while this function computes points in 3D and only interpolates the M value.

2.0.0 [See also:](#) [ST\\_LineInterpolatePoint](#), [ST\\_LineMerge](#), [ST\\_LineInterpolatePoints](#)



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

## Return point 20% along 3D line

Return point 20% along 3D line

```
-- 2D line: (25 50, 100 125, 150 190)
-- 3D line: (25 50 76.5974135047432, 100 125 76.5974135047432, 150 190 76.5974135047432)
SELECT ST_AsEWKT(ST_LineInterpolatePoint(the_line, 0.20))
FROM (SELECT ST_GeomFromEWKT('LINESTRING(25 50, 100 125, 150 190)') as the_line) As
foo;
-----
POINT(51.5974135047432 76.5974135047432)
```

## ST\_LineInterpolatePoint

[ST\\_LineInterpolatePoint](#), [ST\\_LineInterpolatePoint](#), [ST\\_LineMerge](#)

### 8.18.3 ST\_LineInterpolatePoints

`ST_LineInterpolatePoints` — Returns points interpolated along a line at a fractional interval.

#### Synopsis

geometry `ST_LineInterpolatePoints`(geometry a\_linestring, float8 a\_fraction, boolean repeat);

#### Returns

Returns one or more points interpolated along a line at a fractional interval. The first argument must be a `LINestring`. The second argument is a float8 between 0 and 1 representing the spacing between the points as a fraction of line length. If the third argument is false, at most one point will be constructed (which is equivalent to [ST\\_LineInterpolatePoint](#).)

If the result has zero or one points, it is returned as a `POINT`. If it has two or more points, it is returned as a `MULTIPOINT`.

Availability: 2.5.0



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



This function supports M coordinates.



**ST\_LineSubstring**

```
-- ST_LineSubstring: Returns the part of a line between two fractional locations.
-- &#xc70;&#xb9ac;&#xb97c; &#xb530;&#xb77c; &#xc788;&#xb294; &#xc7d3ec;&#xc778;&#xd2b8;&#xc758; &#xb88;&#xc9c0;&#xc218;&#xb97c; &#xb300;&#xcda9; &#xc5b4;&#xb9bc;&#xc9d0;&#xc791;&#xd558;&#xae30;
-- foo &#xc804;&#xc9cb4;&#xcac00; &#xc9d1; &#xc911;&#xc2ec;&#xc810; &#xbcf4; &#xc70;&#xb9ac;&#xc98;&#xb7fc; &#xbcf4;&#xc774;&#xb294; &#xcac00;&#xc9dc; &#xb370;&#xc774;&#xd130;&#xb97c; &#xc0dd;&#xc131;&#xd558;&#xae30; &#xc704;&#xd55c; &#xcac83;&#xc774;&#xb77c;&#xb294; &#xc810;&#xc5d0; &#xc8fc;&#xc758;
-- ST_DWithin(&#xc744; &#xc774;&#xc6a9;&#xd574;&#xc11c; &#xc70;&#xb9ac;&#xc5d0;&#xc11c; &#xb108;&#xbb34; &#xba40;&#xb9ac; &#xb5a8;&#xc5b4;&#xc9c4; &#xc9d1;&#xb4e4;&#xc744; &#xc81c;&#xc678;
SELECT ST_AsText(house_loc) As as_text_house_loc,
       startstreet_num +
       CAST( (endstreet_num - startstreet_num)
            * ST_LineLocatePoint(street_line, house_loc) As integer) As
       street_num
FROM
(SELECT ST_GeomFromText('LINESTRING(1 2, 3 4)') As street_line,
      ST_MakePoint(x*1.01,y*1.03) As house_loc, 10 As startstreet_num,
      20 As endstreet_num
FROM generate_series(1,3) x CROSS JOIN generate_series(2,4) As y)
As foo
WHERE ST_DWithin(street_line, house_loc, 0.2);
```

| as_text_house_loc | street_num |
|-------------------|------------|
| POINT(1.01 2.06)  | 10         |
| POINT(2.02 3.09)  | 15         |
| POINT(3.03 4.12)  | 20         |

```
-- &#xd3ec;&#xc778;&#xd2b8; &#xb610;&#xb294; &#xb2e4;&#xb978; &#xb3c4;&#xd615;&#xc5d0; &#xcac00;&#xc7a5; &#xcac00;&#xae4c;&#xc6b4; &#xb77c;&#xc778; &#xc0c1;&#xc758; &#xd3ec;&#xc778;&#xd2b8;&#xb97c; &#xc3e;&#xae30;
SELECT ST_AsText(ST_LineInterpolatePoint(foo.the_line, ST_LineLocatePoint(foo.the_line,
ST_GeomFromText('POINT(4 3)'))))
FROM (SELECT ST_GeomFromText('LINESTRING(1 2, 4 5, 6 7)') As the_line) As foo;
st_astext
-----
POINT(3 4)
```

**ST\_LineSubstring**

**ST\_DWithin, ST\_Length2D, ST\_LineInterpolatePoint, ST\_LineSubstring**

**8.18.5 ST\_LineSubstring**

**ST\_LineSubstring** — Returns the part of a line between two fractional locations.

**Synopsis**

geometry **ST\_LineSubstring**(geometry a\_linestring, float8 startfraction, float8 endfraction);



If start and end locations are the same, the result is a POINT.

```
SELECT ST_AsText(ST_LineSubstring( 'LINESTRING(25 50, 100 125, 150 190)', 0.333, 0.333));
-----
POINT(69.2846934853974 94.2846934853974)
```

A query to cut a LineString into sections of length 100 or shorter. It uses `generate_series()` with a `CROSS JOIN LATERAL` to produce the equivalent of a FOR loop.

```
WITH data(id, geom) AS (VALUES
  ( 'A', 'LINESTRING( 0 0, 200 0)::geometry ),
  ( 'B', 'LINESTRING( 0 100, 350 100)::geometry ),
  ( 'C', 'LINESTRING( 0 200, 50 200)::geometry )
)
SELECT id, i,
       ST_AsText( ST_LineSubstring( geom, startfrac, LEAST( endfrac, 1 ) ) ) AS geom
FROM (
  SELECT id, geom, ST_Length(geom) len, 100 sublen FROM data
) AS d
CROSS JOIN LATERAL (
  SELECT i, (sublen * i) / len AS startfrac,
         (sublen * (i+1)) / len AS endfrac
  FROM generate_series(0, floor( len / sublen )::integer ) AS t(i)
  -- skip last i if line length is exact multiple of sublen
  WHERE (sublen * i) / len <> 1.0
) AS d2;
```

| id | i | geom                        |
|----|---|-----------------------------|
| A  | 0 | LINESTRING(0 0,100 0)       |
| A  | 1 | LINESTRING(100 0,200 0)     |
| B  | 0 | LINESTRING(0 100,100 100)   |
| B  | 1 | LINESTRING(100 100,200 100) |
| B  | 2 | LINESTRING(200 100,300 100) |
| B  | 3 | LINESTRING(300 100,350 100) |
| C  | 0 | LINESTRING(0 200,50 200)    |

**ST\_LineSubstring**

[ST\\_Length](#), [ST\\_LineInterpolatePoint](#), [ST\\_LineMerge](#)

### 8.18.6 ST\_LocateAlong

`ST_LocateAlong` — Returns the point(s) on a geometry that match a measure value.

#### Synopsis

geometry **ST\_LocateAlong**(geometry ageom\_with\_measure, float8 a\_measure, float8 offset);

**ST\_LocateAlong**

Returns the location(s) along a measured geometry that have the given measure values. The result is a Point or MultiPoint. Polygonal inputs are not supported.

If `offset` is provided, the result is offset to the left or right of the input line by the specified distance. A positive offset will be to the left, and a negative one to the right.

**Note**

Use this function only for linear geometries with an M component

The semantic is specified by the *ISO/IEC 13249-3 SQL/MM Spatial* standard.

1.1.0 `ST_LocateAlong`(`geometry geomA`, `float8 measure_start`, `float8 measure_end`, `float8 offset`);

2.0.0 `ST_LocateAlong`(`geometry geomA`, `float8 measure_start`, `float8 measure_end`, `float8 offset`, `boolean use_m`);

✓ This function supports M coordinates.

✓ This method implements the SQL/MM specification. SQL-MM IEC 13249-3: 5.1.13

**Example**

```
SELECT ST_AsText (
  ST_LocateAlong (
    'MULTILINESTRINGM((1 2 3, 3 4 2, 9 4 3), (1 2 3, 5 4 5))'::geometry,
    3 ) );
```

```
-----
MULTIPOINT M ((1 2 3), (9 4 3), (1 2 3))
```

**See Also**

[ST\\_LocateBetween](#), [ST\\_LocateBetweenElevations](#), [ST\\_InterpolatePoint](#)

## 8.18.7 ST\_LocateBetween

`ST_LocateBetween` — Returns the portions of a geometry that match a measure range.

**Synopsis**

`geometry` `ST_LocateBetween`(`geometry geomA`, `float8 measure_start`, `float8 measure_end`, `float8 offset`);

**Example**

```
SELECT ST_AsText (
  ST_LocateBetween (
    'MULTILINESTRINGM((1 2 3, 3 4 2, 9 4 3), (1 2 3, 5 4 5))'::geometry,
    3, 5, 0 );
```





## Synopsis

```
float8 ST_InterpolatePoint(geometry linear_geom_with_measure, geometry point);
```

### Notes

Returns an interpolated measure value of a linear measured geometry at the location closest to the given point.



#### Note

Use this function only for linear geometries with an M component

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



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

### Example

```
SELECT ST_InterpolatePoint('LINESTRING M (0 0 0, 10 0 20)', 'POINT(5 5)');
-----
10
```

### See Also

[ST\\_AddMeasure](#), [ST\\_LocateAlong](#), [ST\\_LocateBetween](#)

## 8.18.10 ST\_AddMeasure

`ST_AddMeasure` — Interpolates measures along a linear geometry.

### Synopsis

```
geometry ST_AddMeasure(geometry geom_mline, float8 measure_start, float8 measure_end);
```

### Notes

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

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



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

**SQL Examples**

```

SELECT ST_AsText(ST_AddMeasure(
ST_GeomFromEWKT('LINESTRING(1 0, 2 0, 4 0)'),1,4)) As ewelev;
-----
LINESTRINGM(1 0 1,2 0 2,4 0 4)

SELECT ST_AsText(ST_AddMeasure(
ST_GeomFromEWKT('LINESTRING(1 0 4, 2 0 4, 4 0 4)'),10,40)) As ewelev;
-----
LINESTRING(1 0 4 10,2 0 4 20,4 0 4 40)

SELECT ST_AsText(ST_AddMeasure(
ST_GeomFromEWKT('LINESTRINGM(1 0 4, 2 0 4, 4 0 4)'),10,40)) As ewelev;
-----
LINESTRINGM(1 0 10,2 0 20,4 0 40)

SELECT ST_AsText(ST_AddMeasure(
ST_GeomFromEWKT('MULTILINESTRINGM((1 0 4, 2 0 4, 4 0 4),(1 0 4, 2 0 4, 4 0 4)'),10,70)) As ←
    ewelev;
-----
MULTILINESTRINGM((1 0 10,2 0 20,4 0 40),(1 0 40,2 0 50,4 0 70))

```

## 8.19 Trajectory Functions

### 8.19.1 ST\_IsValidTrajectory

`ST_IsValidTrajectory` — Tests if the geometry is a valid trajectory.

#### Synopsis

boolean `ST_IsValidTrajectory`(geometry line);

#### Description

Tests if a geometry encodes a valid trajectory. A valid trajectory is represented as a `LINESTRING` with measures (M values). The measure values must increase from each vertex to the next.

Valid trajectories are expected as input to spatio-temporal functions like [ST\\_ClosestPointOfApproach](#)

Availability: 2.2.0



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

#### Examples

```

-- A valid trajectory
SELECT ST_IsValidTrajectory(ST_MakeLine(
    ST_MakePointM(0,0,1),
    ST_MakePointM(0,1,2))
);

```

```

t

-- An invalid trajectory
SELECT ST_IsValidTrajectory(ST_MakeLine(ST_MakePointM(0,0,1), ST_MakePointM(0,1,0)));
NOTICE: Measure of vertex 1 (0) not bigger than measure of vertex 0 (1)
  st_isvalidtrajectory
-----
f

```

## See Also

[ST\\_ClosestPointOfApproach](#)

## 8.19.2 ST\_ClosestPointOfApproach

`ST_ClosestPointOfApproach` — Returns a measure at the closest point of approach of two trajectories.

### Synopsis

```
float8 ST_ClosestPointOfApproach(geometry track1, geometry track2);
```

### Description

Returns the smallest measure at which points interpolated along the given trajectories are at the smallest distance.

Inputs must be valid trajectories as checked by [ST\\_IsValidTrajectory](#). Null is returned if the trajectories do not overlap in their M ranges.

See [ST\\_LocateAlong](#) for getting the actual points at the given measure.

Availability: 2.2.0



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

### Examples

```

-- Return the time in which two objects moving between 10:00 and 11:00
-- are closest to each other and their distance at that point
WITH inp AS ( SELECT
  ST_AddMeasure('LINESTRING Z (0 0 0, 10 0 5)::geometry,
    extract(epoch from '2015-05-26 10:00'::timestampz),
    extract(epoch from '2015-05-26 11:00'::timestampz)
  ) a,
  ST_AddMeasure('LINESTRING Z (0 2 10, 12 1 2)::geometry,
    extract(epoch from '2015-05-26 10:00'::timestampz),
    extract(epoch from '2015-05-26 11:00'::timestampz)
  ) b
), cpa AS (
  SELECT ST_ClosestPointOfApproach(a,b) m FROM inp
), points AS (
  SELECT ST_Force3DZ(ST_GeometryN(ST_LocateAlong(a,m),1)) pa,
    ST_Force3DZ(ST_GeometryN(ST_LocateAlong(b,m),1)) pb
  FROM inp, cpa
)
SELECT to_timestamp(m) t,
  ST_Distance(pa,pb) distance

```

```
FROM points, cpa;

          t          | distance
-----+-----
2015-05-26 10:45:31.034483+02 | 1.96036833151395
```

**See Also**

[ST\\_IsValidTrajectory](#), [ST\\_DistanceCPA](#), [ST\\_LocateAlong](#), [ST\\_AddMeasure](#)

**8.19.3 ST\_DistanceCPA**

`ST_DistanceCPA` — Returns the distance between the closest point of approach of two trajectories.

**Synopsis**

```
float8 ST_DistanceCPA(geometry track1, geometry track2);
```

**Description**

Returns the minimum distance two moving objects have ever been each other.

Inputs must be valid trajectories as checked by [ST\\_IsValidTrajectory](#). Null is returned if the trajectories do not overlap in their M ranges.

Availability: 2.2.0



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

**Examples**

```
-- Return the minimum distance of two objects moving between 10:00 and 11:00
WITH inp AS ( SELECT
  ST_AddMeasure('LINESTRING Z (0 0 0, 10 0 5)::geometry',
    extract(epoch from '2015-05-26 10:00'::timestampz),
    extract(epoch from '2015-05-26 11:00'::timestampz)
  ) a,
  ST_AddMeasure('LINESTRING Z (0 2 10, 12 1 2)::geometry',
    extract(epoch from '2015-05-26 10:00'::timestampz),
    extract(epoch from '2015-05-26 11:00'::timestampz)
  ) b
)
SELECT ST_DistanceCPA(a,b) distance FROM inp;

          distance
-----
1.96036833151395
```

**See Also**

[ST\\_IsValidTrajectory](#), [ST\\_ClosestPointOfApproach](#), [ST\\_AddMeasure](#), [|](#)

## 8.19.4 ST\_CPAWithin

ST\_CPAWithin — Tests if the closest point of approach of two trajectories is within the specified distance.

### Synopsis

boolean **ST\_CPAWithin**(geometry track1, geometry track2, float8 dist);

### Description

Tests whether two moving objects have ever been closer than the specified distance.

Inputs must be valid trajectories as checked by [ST\\_IsValidTrajectory](#). False is returned if the trajectories do not overlap in their M ranges.

Availability: 2.2.0



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

### Examples

```
WITH inp AS ( SELECT
  ST_AddMeasure('LINESTRING Z (0 0 0, 10 0 5)::geometry,
    extract(epoch from '2015-05-26 10:00'::timestampz),
    extract(epoch from '2015-05-26 11:00'::timestampz)
  ) a,
  ST_AddMeasure('LINESTRING Z (0 2 10, 12 1 2)::geometry,
    extract(epoch from '2015-05-26 10:00'::timestampz),
    extract(epoch from '2015-05-26 11:00'::timestampz)
  ) b
)
SELECT ST_CPAWithin(a,b,2), ST_DistanceCPA(a,b) distance FROM inp;
```

| st_cpawithin | distance         |
|--------------|------------------|
| t            | 1.96521473776207 |

### See Also

[ST\\_IsValidTrajectory](#), [ST\\_ClosestPointOfApproach](#), [ST\\_DistanceCPA](#), [|](#)

## 8.20 SFCGAL

### 8.20.1 postgis\_sfcgal\_version

postgis\_sfcgal\_version — SFCGAL

### Synopsis

text **postgis\_sfcgal\_version**(void);



**8.20.3 ST\_3DArea**

2.1.0 This method needs SFCGAL backend.



This method needs SFCGAL backend.



This method implements the SQL/MM specification. SQL-MM IEC 13249-3: 8.1, 10.5



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



This function supports Polyhedral surfaces.



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

**8.20.4 ST\_3DConvexHull**

This method needs SFCGAL backend. This method implements the SQL/MM specification. SQL-MM IEC 13249-3: 8.1, 10.5. This function supports 3d and will not drop the z-index. This function supports Polyhedral surfaces. This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).

```
SELECT ST_3DArea(geom) As cube_surface_area,
       ST_3DArea(ST_MakeSolid(geom)) As solid_surface_area
FROM (SELECT 'POLYHEDRALSURFACE( ((0 0 0, 0 0 1, 0 1 1, 0 1 0, 0 0 0)),
  ((0 0 0, 0 1 0, 1 1 0, 1 0 0, 0 0 0)),
  ((0 0 0, 1 0 0, 1 0 1, 0 0 1, 0 0 0)),
  ((1 1 0, 1 1 1, 1 0 1, 1 0 0, 1 1 0)),
  ((0 1 0, 0 1 1, 1 1 1, 1 1 0, 0 1 0)),
  ((0 0 1, 1 0 1, 1 1 1, 0 1 1, 0 0 1)) )'::geometry) As f(geom);
```

| cube_surface_area | solid_surface_area |
|-------------------|--------------------|
| 6                 | 0                  |

**8.20.5 ST\_3DConvexHull**

[ST\\_Area](#), [ST\\_MakeSolid](#), [ST\\_IsSolid](#), [ST\\_Area](#)

**8.20.4 ST\_3DConvexHull**

ST\_3DConvexHull — This method needs SFCGAL backend. This method implements the SQL/MM specification. SQL-MM IEC 13249-3: 8.1, 10.5. This function supports 3d and will not drop the z-index. This function supports Polyhedral surfaces. This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).

**Synopsis**

geometry **ST\_3DConvexHull**(geometry geom1);

**8.20.5 ST\_3DConvexHull**

2.1.0 This method needs SFCGAL backend.



This method needs SFCGAL backend.



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

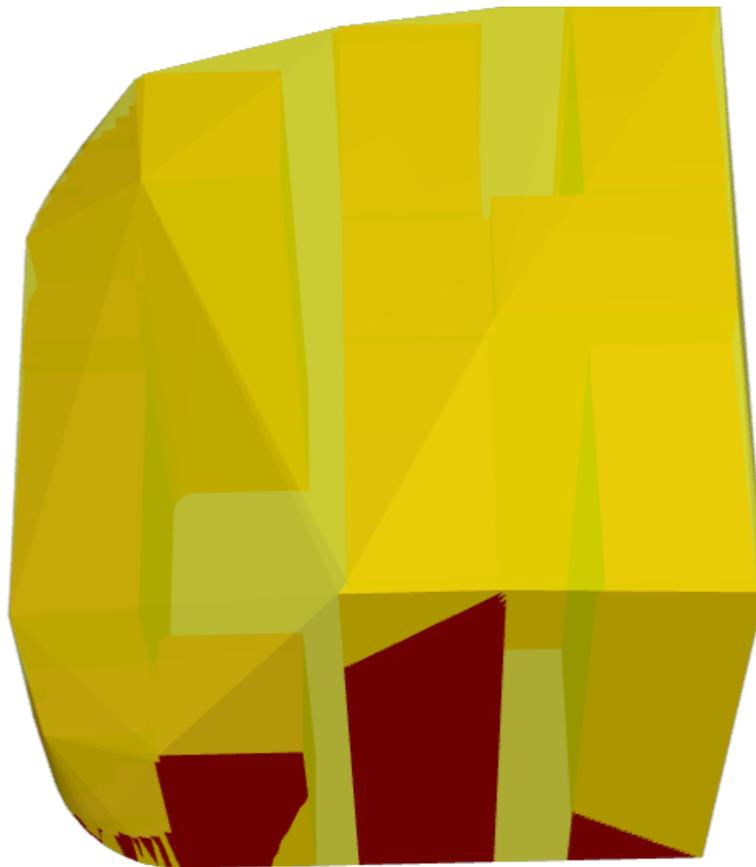
- ✔ This function supports Polyhedral surfaces.
- ✔ This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).

### Example

```
SELECT ST_AsText(ST_3DConvexHull('LINESTRING Z(0 0 5, 1 5 3, 5 7 6, 9 5 3, 5 7 5, 6 3 5) ←
  '::geometry));
```

```
POLYHEDRALSURFACE Z (((1 5 3,9 5 3,0 0 5,1 5 3)),((1 5 3,0 0 5,5 7 6,1 5 3)),((5 7 6,5 7 ←
  5,1 5 3,5 7 6)),((0 0 5,6 3 5,5 7 6,0 0 5)),((6 3 5,9 5 3,5 7 6,6 3 5)),((0 0 5,9 5 3,6 ←
  3 5,0 0 5)),((9 5 3,5 7 5,5 7 6,9 5 3)),((1 5 3,5 7 5,9 5 3,1 5 3)))
```

```
WITH f AS (SELECT i, ST_Extrude(geom, 0,0, i ) AS geom
FROM ST_Subdivide(ST_Letters('CH'),5) WITH ORDINALITY AS sd(geom,i)
)
SELECT ST_3DConvexHull(ST_Collect(f.geom) )
FROM f;
```

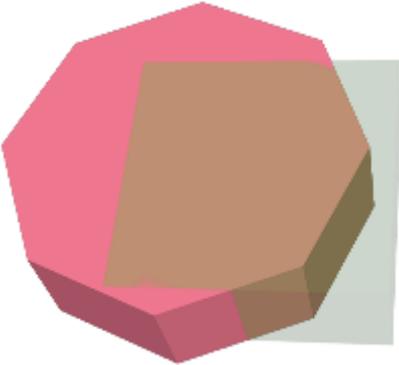
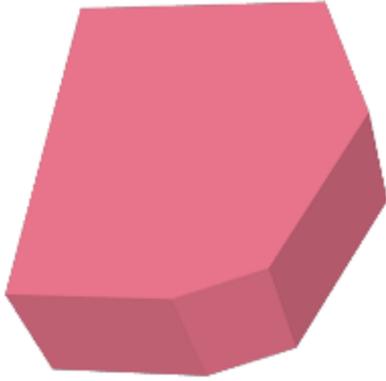


Original geometry overlaid with 3D convex hull

### See also

[ST\\_Letters](#), [ST\\_AsX3D](#)



|   |  |
|---|--|
| <pre>SELECT ST_Extrude(ST_Buffer( ←     ST_GeomFromText('POINT(100 90)'),     50, 'quad_segs=2'),0,0,30) AS geom1,     ST_Extrude(ST_Buffer( ←     ST_GeomFromText('POINT(80 80)'),     50, 'quad_segs=1'),0,0,30) AS geom2;</pre>  <p>geom1 and geom2</p> | <pre>SELECT ST_3DIntersection(geom1,geom2) FROM ( SELECT ST_Extrude(ST_Buffer( ←     ST_GeomFromText('POINT(100 90)'),     50, 'quad_segs=2'),0,0,30) AS geom1,     ST_Extrude(ST_Buffer( ←     ST_GeomFromText('POINT(80 80)'),     50, 'quad_segs=1'),0,0,30) AS geom2 ) As ← t;</pre>  <p>geom1 and geom2</p> |
|---|--|

3 and geom1 and geom2

```
SELECT ST_AsText(ST_3DIntersection(linestring, polygon)) As wkt
FROM ST_GeomFromText('LINESTRING Z (2 2 6,1.5 1.5 7,1 1 8,0.5 0.5 8,0 0 10)') AS ←
linestring
CROSS JOIN ST_GeomFromText('POLYGON((0 0 8, 0 1 8, 1 1 8, 1 0 8, 0 0 8))') AS polygon;
```

wkt

```
-----
LINESTRING Z (1 1 8,0.5 0.5 8)
```

and Z

```
SELECT ST_AsText(ST_3DIntersection(
    ST_GeomFromText('POLYHEDRALSURFACE Z( ((0 0 0, 0 0 1, 0 1 1, 0 1 0, 0 0 0)) ←
    ((0 0 0, 0 1 0, 1 1 0, 1 0 0, 0 0 0)), ((0 0 0, 1 0 0, 1 0 1, 0 0 1, 0 0 0)),
    ((1 1 0, 1 1 1, 1 0 1, 1 0 0, 1 1 0)),
    ((0 1 0, 0 1 1, 1 1 1, 1 1 0, 0 1 0)), ((0 0 1, 1 0 1, 1 1 1, 0 1 1, 0 0 1)) )'),
    'POLYGON Z ((0 0 0, 0 0 0.5, 0 0.5 0.5, 0 0.5 0, 0 0 0))'::geometry))
```

```
TIN Z (((0 0 0,0 0 0.5,0 0.5 0.5,0 0 0)),((0 0.5 0,0 0 0,0 0.5 0.5,0 0.5 0)))
```

and (ST\_Dimension and 3 and geom1 and geom2)

```
SELECT ST_AsText(ST_3DIntersection( ST_Extrude(ST_Buffer('POINT(10 20)::geometry,10,1) ←
,0,0,30),
ST_Extrude(ST_Buffer('POINT(10 20)::geometry,10,1),2,0,10) ));
```







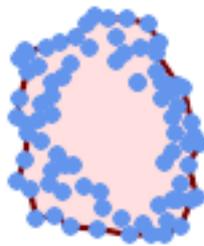
**ST\_ConcaveHull**

Assume we are given a set  $S$  of points in 2D [...] and we would like to have something like "the shape formed by these points". This is quite a vague notion and there are probably many possible interpretations, the  $\alpha$ -shape being one of them. Alpha shapes can be used for shape reconstruction from a dense unorganized set of data points. Indeed, an  $\alpha$ -shape is demarcated by a frontier, which is a linear approximation of the original shape [1]. [1] F. Bernardini and C. Bajaj. Sampling and reconstructing manifolds using alpha-shapes. Technical Report CSD-TR-97-013, Dept. Comput. Sci., Purdue Univ., West Lafayette, IN, 1997. Source: [CGAL ALpha Shapes](#) This function compute the concave hull of a set of geometry, but using CGAL and a different algorithm than ST\_ConcaveHull performed by the GEOS module. See : [Concave Hulls in JTS](#)

Availability: 3.3.0 - requires SFCGAL >= 1.4.1.



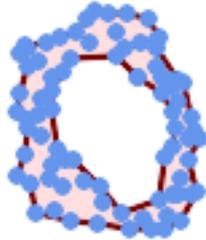
This method needs SFCGAL backend.

**ST\_ConcaveHull**

*Concave Hull of a MultiPoint (same example As ST\_OptimalAlphaShape)*

```
SELECT ST_AsText(ST_AlphaShape('MULTIPOINT((63 84),(76 88),(68 73),(53 18),(91 50),(81 70),
(88 29),(24 82),(32 51),(37 23),(27 54),(84 19),(75 87),(44 42),(77 67),(90 ←
30),(36 61),(32 65),
(81 47),(88 58),(68 73),(49 95),(81 60),(87 50),
(78 16),(79 21),(30 22),(78 43),(26 85),(48 34),(35 35),(36 40),(31 79),(83 ←
29),(27 84),(52 98),(72 95),(85 71),
(75 84),(75 77),(81 29),(77 73),(41 42),(83 72),(23 36),(89 53),(27 57),(57 ←
97),(27 77),(39 88),(60 81),
(80 72),(54 32),(55 26),(62 22),(70 20),(76 27),(84 35),(87 42),(82 54),(83 ←
64),(69 86),(60 90),(50 86),(43 80),(36 73),
(36 68),(40 75),(24 67),(23 60),(26 44),(28 33),(40 32),(43 19),(65 16),(73 ←
16),(38 46),(31 59),(34 86),(45 90),(64 97))'::geometry,80.2));
```

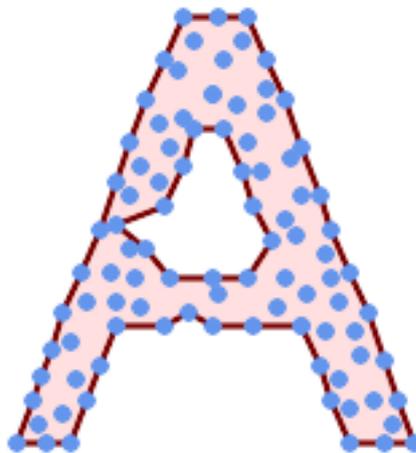
```
POLYGON((89 53,91 50,87 42,90 30,88 29,84 19,78 16,73 16,65 16,53 18,43 19,37 23,30 22,28 ←
33,23 36,26 44,27 54,23 60,24 67,
27 77,24 82,26 85,34 86,39 88,45 90,49 95,52 98,57 97,64 97,72 95,76 88,75 84,83 ←
72,85 71,88 58,89 53))
```



*Concave Hull of a MultiPoint, allowing holes (same example as ST\_OptimalAlphaShape)*

```
SELECT ST_AsText(ST_AlphaShape('MULTIPOINT((63 84),(76 88),(68 73),(53 18),(91 50),(81 70) ←
, (88 29),(24 82),(32 51),(37 23),(27 54),(84 19),(75 87),(44 42),(77 67),(90 30),(36 61) ←
, (32 65),(81 47),(88 58),(68 73),(49 95),(81 60),(87 50),
(78 16),(79 21),(30 22),(78 43),(26 85),(48 34),(35 35),(36 40),(31 79),(83 ←
29),(27 84),(52 98),(72 95),(85 71),
(75 84),(75 77),(81 29),(77 73),(41 42),(83 72),(23 36),(89 53),(27 57),(57 ←
97),(27 77),(39 88),(60 81),
(80 72),(54 32),(55 26),(62 22),(70 20),(76 27),(84 35),(87 42),(82 54),(83 ←
64),(69 86),(60 90),(50 86),(43 80),(36 73),
(36 68),(40 75),(24 67),(23 60),(26 44),(28 33),(40 32),(43 19),(65 16),(73 ←
16),(38 46),(31 59),(34 86),(45 90),(64 97))'::geometry, 100.1,true))

POLYGON((89 53,91 50,87 42,90 30,88 29,84 19,78 16,73 16,65 16,53 18,43 19,37 23,30 22,28 ←
33,23 36,26 44,27 54,23 60,24 67,27 77,24 82,26 85,34 86,39 88,45 90,49 95,52 98,57 ←
97,64 97,72 95,76 88,75 84,83 72,85 71,88 58,89 53),
(36 61,36 68,40 75,43 80,50 86,60 81,68 73,77 67,81 60,82 54,81 47,78 43,76 ←
27,62 22,54 32,48 34,44 42,38 46,36 61))
```



*Concave Hull of a MultiPoint, allowing holes (same example as ST\_ConcaveHull)*

```
SELECT ST_AlphaShape (
  'MULTIPOINT ((132 64), (114 64), (99 64), (81 64), (63 64), (57 49), (52 36), (
    46 20), (37 20), (26 20), (32 36), (39 55), (43 69), (50 84), (57 100), (63 (
    118), (68 133), (74 149), (81 164), (88 180), (101 180), (112 180), (119 (
    164), (126 149), (132 131), (139 113), (143 100), (150 84), (157 69), (163 (
    51), (168 36), (174 20), (163 20), (150 20), (143 36), (139 49), (132 64), (
    99 151), (92 138), (88 124), (81 109), (74 93), (70 82), (83 82), (99 82), (
    112 82), (126 82), (121 96), (114 109), (110 122), (103 138), (99 151), (34 (
    27), (43 31), (48 44), (46 58), (52 73), (63 73), (61 84), (72 71), (90 69) (
    , (101 76), (123 71), (141 62), (166 27), (150 33), (159 36), (146 44), (154 (
    53), (152 62), (146 73), (134 76), (143 82), (141 91), (130 98), (126 104), (
    132 113), (128 127), (117 122), (112 133), (119 144), (108 147), (119 153) (
    , (110 171), (103 164), (92 171), (86 160), (88 142), (79 140), (72 124), (
    83 131), (79 118), (68 113), (63 102), (68 93), (35 45))'::geometry,102.2, (
    true);
```

```
POLYGON((134 80,136 75,130 63,135 45,132 44,126 28,117 24,110 24,98 24,80 27,82 39,72 51,60 (
    48,56 34,52 52,42 50,
    34 54,39 66,40 81,34 90,36 100,40 116,36 123,39 128,51 129,58 132,68 135,74 (
    142,78 147,86 146,96 146,
    108 142,114 132,112 126,112 116,116 110,120 108,125 108,128 106,125 96,132 (
    87,134 80))
```

**ST\_ConcaveHull**, **ST\_OptimalAlphaShape**

**ST\_ConcaveHull**, **ST\_OptimalAlphaShape**

## 8.20.9 ST\_ApproximateMedialAxis

**ST\_ApproximateMedialAxis** — (approximate medial axis)

### Synopsis

geometry **ST\_ApproximateMedialAxis**(geometry geom);

**ST\_ApproximateMedialAxis**

(approximate medial axis)

(1.2.0)

SFCGAL; API; ST\_StraightSkeleton; (wrapper)

2.2.0



This method needs SFCGAL backend.



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



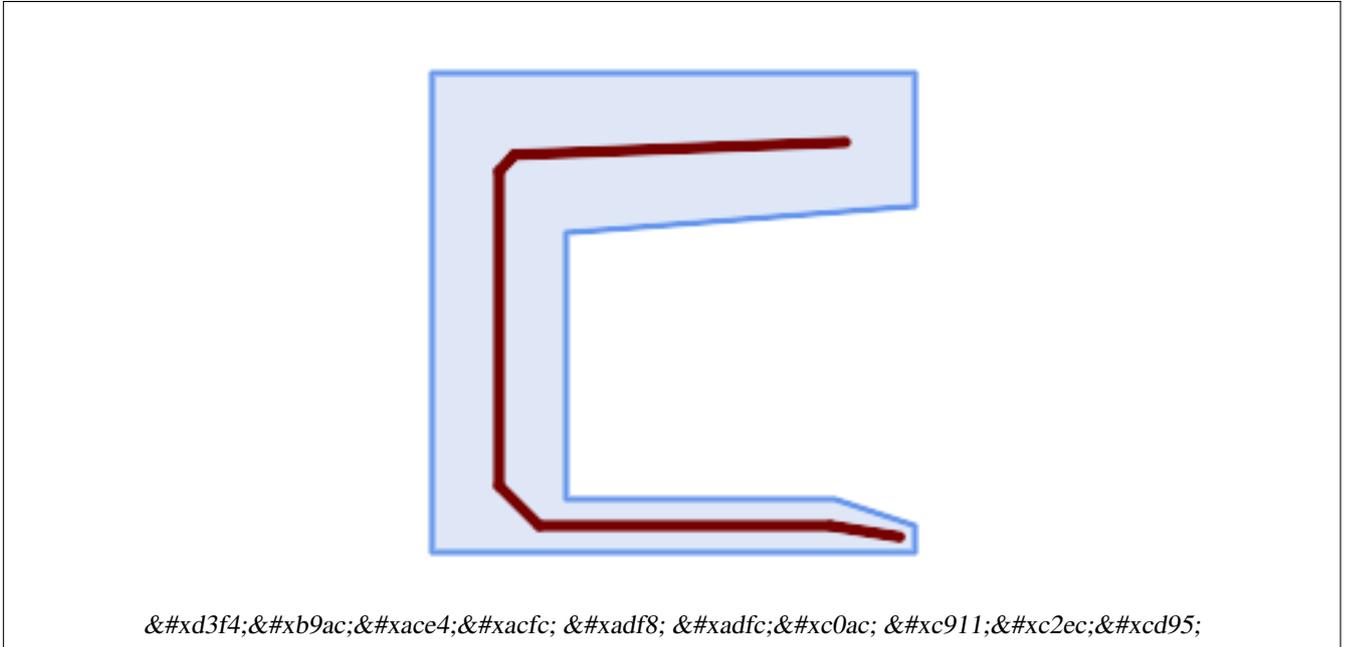
This function supports Polyhedral surfaces.



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

**Example:**

```
SELECT ST_ApproximateMedialAxis(ST_GeomFromText('POLYGON (( 190 190, 10 190, 10 10, 190 10, ↵
190 20, 160 30, 60 30, 60 130, 190 140, 190 190 ))'));
```



*Figure 8.20.10: Medial axis of a polygon.*

**Example:**

`ST_StraightSkeleton`

### 8.20.10 ST\_ConstrainedDelaunayTriangles

`ST_ConstrainedDelaunayTriangles` — Return a constrained Delaunay triangulation around the given input geometry.

#### Synopsis

geometry `ST_Tessellate`(geometry geom);

**Example:**

Return a **Constrained Delaunay triangulation** around the vertices of the input geometry. Output is a TIN.



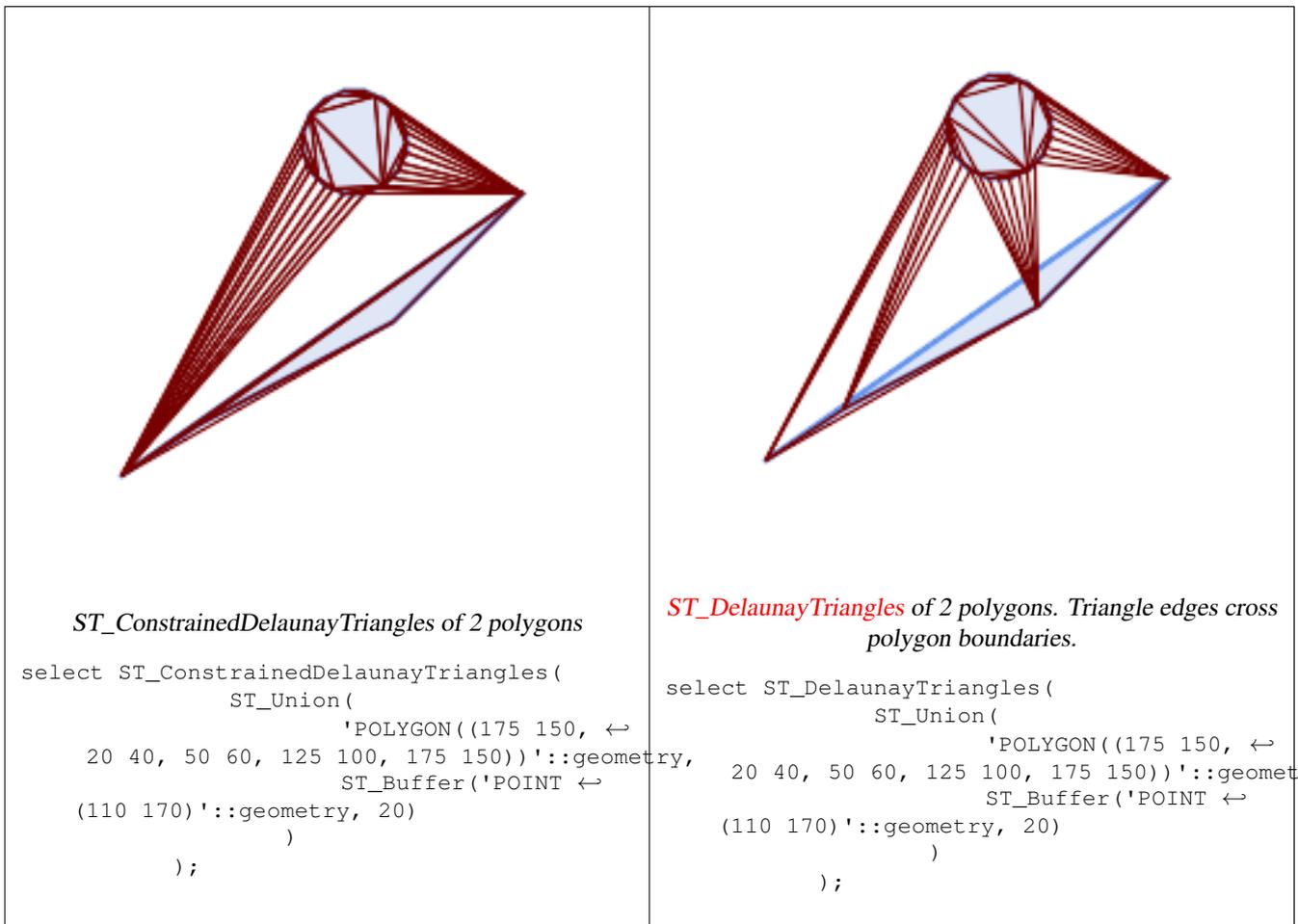
This method needs SFCGAL backend.

2.1.0 **Example:**



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

**Example:**



**ST\_DelaunayTriangles**

[ST\\_DelaunayTriangles](#), [ST\\_MakeSolid](#), [ST\\_IsSolid](#), [ST\\_Area](#)

### 8.20.11 ST\_Extrude

**ST\_Extrude** — Extrude a 2D geometry into 3D space.

#### Synopsis

geometry **ST\_Extrude**(geometry geom, float x, float y, float z);

**ST\_Extrude**

2.1.0 &#x2013; This function supports Polyhedral surfaces.

- ✔ This method needs SFCGAL backend.
- ✔ This function supports 3d and will not drop the z-index.
- ✔ This function supports Polyhedral surfaces.
- ✔ This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).

**ST\_Buffer**

PostGIS **ST\_AsX3D** **X3Dom HTML** **HTML**

|   |  |
|---|--|
| <pre>SELECT ST_Buffer(ST_GeomFromText ('POINT ↵ (100 90)'), 50, 'quad_segs=2'),0,0,30);</pre>  <p> <b>ST_Buffer</b><br/>         (ST_GeomFromText('POINT (100 90)'),<br/>         50, 'quad_segs=2'),0,0,30);     </p> | <pre>ST_Extrude(ST_Buffer(ST_GeomFromText (' ↵ POINT(100 90)'), 50, 'quad_segs=2'),0,0,30);</pre>  <p> <b>ST_Extrude</b><br/>         (ST_Buffer(ST_GeomFromText('POINT (100 90)'),<br/>         50, 'quad_segs=2'),0,0,30);     </p> |
|---|--|



## Synopsis

boolean **ST\_IsPlanar**(geometry geom);

### Notes

2.2.0: Added support for 3D geometries. 2.1.0: Added support for Polyhedral surfaces. 2.1: Added support for Triangles and Triangulated Irregular Network Surfaces (TIN).



This method needs SFCGAL backend.



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



This function supports Polyhedral surfaces.



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

## 8.20.14 ST\_IsSolid

**ST\_IsSolid** — Returns true if the geometry is a solid. Returns false if the geometry is a surface or a volume.

## Synopsis

boolean **ST\_IsSolid**(geometry geom1);

### Notes

2.2.0: Added support for 3D geometries. 2.1.0: Added support for Polyhedral surfaces. 2.1: Added support for Triangles and Triangulated Irregular Network Surfaces (TIN).



This method needs SFCGAL backend.



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



This function supports Polyhedral surfaces.



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

## 8.20.15 ST\_MakeSolid

**ST\_MakeSolid** — Returns the solid geometry. Returns null if the geometry is not a solid. Returns the geometry if it is already a solid.

## Synopsis

geometry **ST\_MakeSolid**(geometry geom1);

**Notes:**

2.2.0 This method needs SFCGAL backend.



This method needs SFCGAL backend.



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



This function supports Polyhedral surfaces.



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

**8.20.16 ST\_MinkowskiSum**

ST\_MinkowskiSum — Returns the Minkowski sum of two geometries.

**Synopsis**

geometry ST\_MinkowskiSum(geometry geom1, geometry geom2);

**Notes:**

2.2.0 This method needs SFCGAL backend.

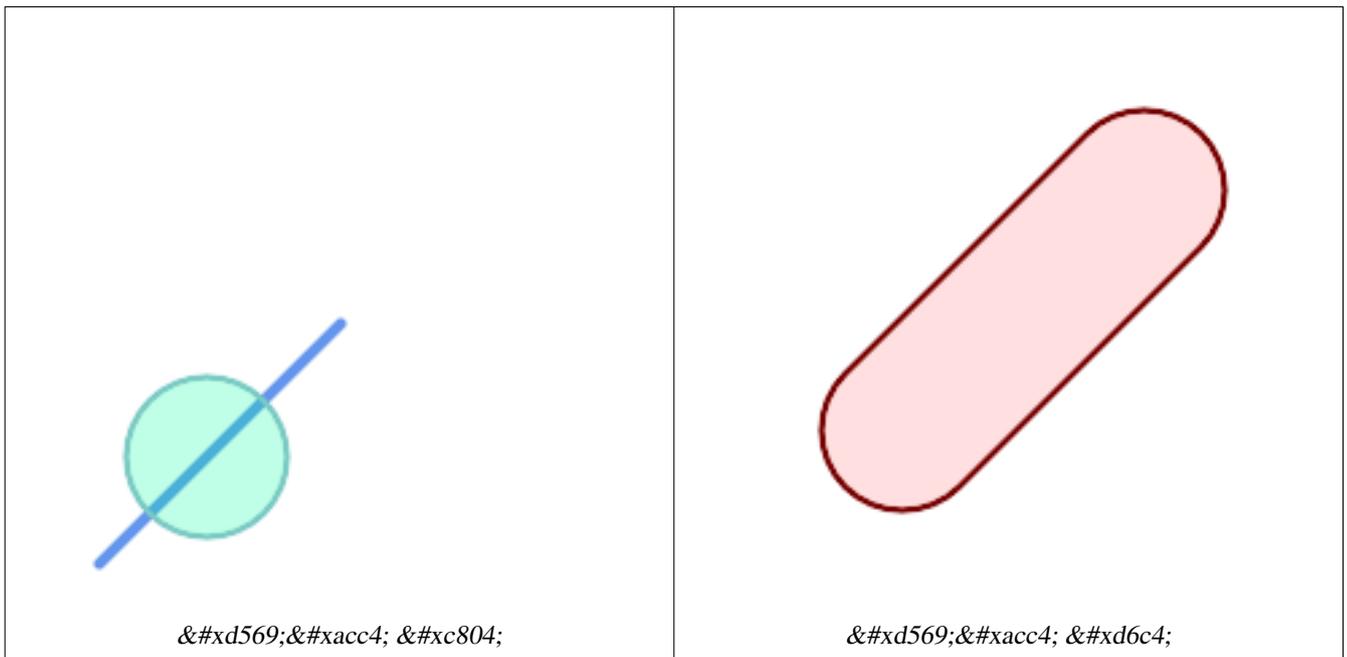
2.1.0 This method needs SFCGAL backend.



This method needs SFCGAL backend.

**Notes:**

2.1.0 This method needs SFCGAL backend.

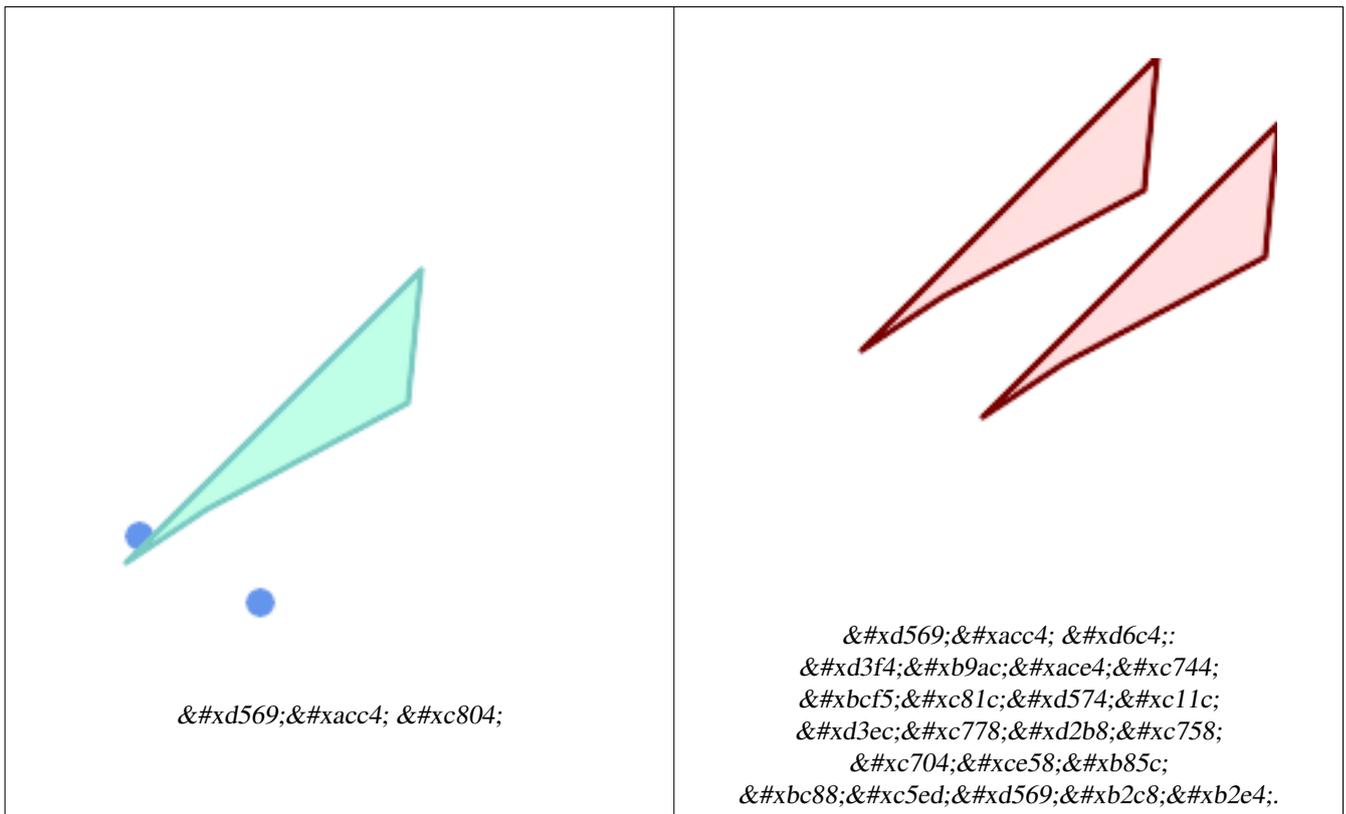


```

SELECT ST_MinkowskiSum(line, circle)
FROM (SELECT
  ST_MakeLine(ST_MakePoint(10, 10),ST_MakePoint(100, 100)) As line,
  ST_Buffer(ST_GeomFromText('POINT(50 50)'), 30) As circle) As foo;

-- WKT --
MULTIPOLYGON(((30 59.9999999999999,30.5764415879031 54.1472903395161,32.2836140246614
48.5194970290472,35.0559116309237 43.3328930094119,38.7867965644036
38.7867965644035,43.332893009412 35.0559116309236,48.5194970290474
32.2836140246614,54.1472903395162 30.5764415879031,60.0000000000001 30,65.8527096604839
30.5764415879031,71.4805029709527 32.2836140246614,76.6671069905881
35.0559116309237,81.2132034355964 38.7867965644036,171.213203435596
128.786796564404,174.944088369076 133.332893009412,177.716385975339
138.519497029047,179.423558412097 144.147290339516,180 150,179.423558412097
155.852709660484,177.716385975339 161.480502970953,174.944088369076
166.667106990588,171.213203435596 171.213203435596,166.667106990588 174.944088369076,
161.480502970953 177.716385975339,155.852709660484 179.423558412097,150
180,144.147290339516 179.423558412097,138.519497029047 177.716385975339,133.332893009412
174.944088369076,128.786796564403 171.213203435596,38.7867965644035
81.2132034355963,35.0559116309236 76.667106990588,32.2836140246614
71.4805029709526,30.5764415879031 65.8527096604838,30 59.9999999999999)))
    
```

The Minkowski sum of a line and a circle is a rounded rectangle.



```
SELECT ST_MinkowskiSum(mp, poly)
FROM (SELECT 'MULTIPOINT(25 50,70 25)::geometry As mp,
  'POLYGON((130 150, 20 40, 50 60, 125 100, 130 150))::geometry As poly
  ) As foo

-- WKT --
MULTIPOLYGON(
  ((70 115,100 135,175 175,225 225,70 115)),
  ((120 65,150 85,225 125,275 175,120 65))
)
```

### 8.20.17 ST\_OptimalAlphaShape

`ST_OptimalAlphaShape` — Computes a possible concave geometry using the CGAL Alpha Shapes algorithm after have computed the "optimal" alpha value.

#### Synopsis

```
geometry ST_OptimalAlphaShape(geometry param_geom, boolean allow_holes = false, integer nb_components);
```

#### **&#xc124;&#xba85;**

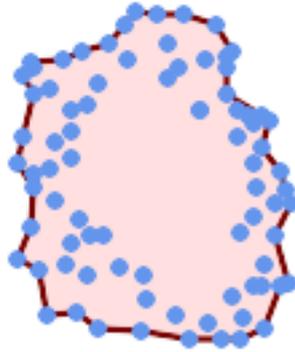
Computes the "optimal" alpha-shapes of the set of geometry. CGAL can automatically find the optimal value of alpha. This version uses it to find an "optimal" alpha-shape. The result is a single polygon. It will not contain holes unless the optional `param_allow_holes` argument is specified as true. The result will be generated such that the number of solid component of the alpha shape is equal to or smaller than `param_nb_components`.

Availability: 3.3.0 - requires SFCGAL >= 1.4.1.



This method needs SFCGAL backend.

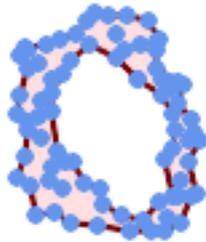
**Concave Hull of a MultiPoint**



*Concave Hull of a MultiPoint (same example as ST\_AlphaShape)*

```
SELECT ST_AsText(ST_OptimalAlphaShape('MULTIPOINT((63 84), (76 88), (68 73), (53 18), (91 50) ←
, (81 70),
(88 29), (24 82), (32 51), (37 23), (27 54), (84 19), (75 87), (44 42), (77 67), (90 ←
30), (36 61), (32 65),
(81 47), (88 58), (68 73), (49 95), (81 60), (87 50),
(78 16), (79 21), (30 22), (78 43), (26 85), (48 34), (35 35), (36 40), (31 79), (83 ←
29), (27 84), (52 98), (72 95), (85 71),
(75 84), (75 77), (81 29), (77 73), (41 42), (83 72), (23 36), (89 53), (27 57), (57 ←
97), (27 77), (39 88), (60 81),
(80 72), (54 32), (55 26), (62 22), (70 20), (76 27), (84 35), (87 42), (82 54), (83 ←
64), (69 86), (60 90), (50 86), (43 80), (36 73),
(36 68), (40 75), (24 67), (23 60), (26 44), (28 33), (40 32), (43 19), (65 16), (73 ←
16), (38 46), (31 59), (34 86), (45 90), (64 97))'::geometry));
```

```
POLYGON((89 53,91 50,87 42,90 30,88 29,84 19,78 16,73 16,65 16,53 18,43 19,37 23,30 22,28 ←
33,23 36,
26 44,27 54,23 60,24 67,27 77,24 82,26 85,34 86,39 88,45 90,49 95,52 98,57 ←
97,64 97,72 95,76 88,75 84,75 77,83 72,85 71,83 64,88 58,89 53))
```



*Concave Hull of a MultiPoint, allowing holes (same example as ST\_AlphaShape)*

```
SELECT ST_AsText(ST_OptimalAlphaShape('MULTIPOINT((63 84),(76 88),(68 73),(53 18),(91 50) ←
, (81 70),(88 29),(24 82),(32 51),(37 23),(27 54),(84 19),(75 87),(44 42),(77 67),(90 30) ←
, (36 61),(32 65),(81 47),(88 58),(68 73),(49 95),(81 60),(87 50),
(78 16),(79 21),(30 22),(78 43),(26 85),(48 34),(35 35),(36 40),(31 79),(83 ←
29),(27 84),(52 98),(72 95),(85 71),
(75 84),(75 77),(81 29),(77 73),(41 42),(83 72),(23 36),(89 53),(27 57),(57 ←
97),(27 77),(39 88),(60 81),
(80 72),(54 32),(55 26),(62 22),(70 20),(76 27),(84 35),(87 42),(82 54),(83 ←
64),(69 86),(60 90),(50 86),(43 80),(36 73),
(36 68),(40 75),(24 67),(23 60),(26 44),(28 33),(40 32),(43 19),(65 16),(73 ←
16),(38 46),(31 59),(34 86),(45 90),(64 97))'::geometry, allow_holes => ←
true));
```

```
POLYGON((89 53,91 50,87 42,90 30,88 29,84 19,78 16,73 16,65 16,53 18,43 19,37 23,30 22,28 ←
33,23 36,26 44,27 54,23 60,24 67,27 77,24 82,26 85,34 86,39 88,45 90,49 95,52 98,57 ←
97,64 97,72 95,76 88,75 84,75 77,83 72,85 71,83 64,88 58,89 53),(36 61,36 68,40 75,43 ←
80,50 86,60 81,68 73,77 67,81 60,82 54,81 47,78 43,81 29,76 27,70 20,62 22,55 26,54 ←
32,48 34,44 42,38 46,36 61))
```

**&#xc38;&#xace0;**

[ST\\_ConcaveHull](#), [ST\\_AlphaShape](#)

## 8.20.18 ST\_Orientation

ST\_Orientation — [&#xd45c;&#xba74;&#xc758;&#xbc29;&#xd5a5;\(orientation\)&#xc744;&#xacb0;&#xc815;&#xd569;&#xb2c8;&#xc744;](#)

### Synopsis

integer **ST\_Orientation**(geometry geom);

**&#xc124;&#xba85;**

[&#xc774;&#xd568;&#xc218;&#xb294;&#xd3f4;&#xb9ac;&#xace4;&#xc5d0;&#xb9cc;&#xc801;&#xc6a9;&#xb429;&#xb2c8;&#xc744;](#)  
[&#xd3f4;&#xb9ac;&#xace4;&#xc774;&#xbc18;&#xc2dc;&#xacc4;&#xbc29;&#xd5a5;&#xc774;&#xba74;&#xc744;](#), [-1&#xc744;](#), [&#xc2dc;&#xbc29;&#xd5a5;&#xc774;&#xba74;](#) [1&#xc744;](#) [&#xbc18;&#xd658;&#xd569;&#xb2c8;&#xb2e4;](#).

2.1.0 This method needs SFCGAL backend.



This method needs SFCGAL backend.



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

## 8.20.19 ST\_StraightSkeleton

`ST_StraightSkeleton` — (straight skeleton)

### Synopsis

geometry `ST_StraightSkeleton`(geometry geom);

### 2.1.0

2.1.0 This method needs SFCGAL backend.



This method needs SFCGAL backend.



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



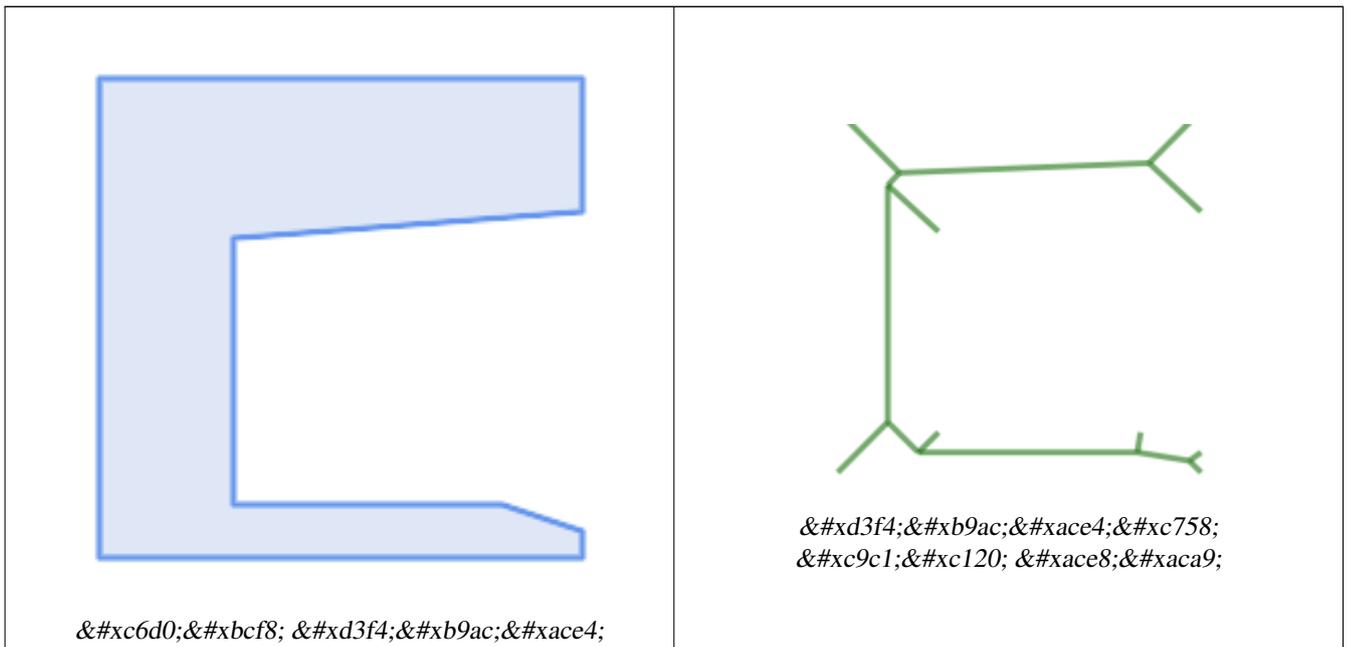
This function supports Polyhedral surfaces.



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

### Example

```
SELECT ST_StraightSkeleton(ST_GeomFromText('POLYGON (( 190 190, 10 190, 10 10, 190 10, 190 20, 160 30, 60 30, 60 130, 190 140, 190 190 ))')); ←
```



Original polygon

Resulting straight skeleton

## 8.20.20 ST\_Tesselate

ST\_Tesselate — tessellate a geometry into a set of triangles. The tessellation is done using a Delaunay triangulation algorithm. The function returns a geometry of type TIN (Triangulated Irregular Network).

### Synopsis

geometry ST\_Tesselate(geometry geom);

### Parameters

`geom` geometry: The geometry to be tessellated. The geometry must be a closed polygon or a set of closed polygons. The function will return a geometry of type TIN (Triangulated Irregular Network).

2.1.0: This function was added in PostGIS 2.1.0.



This method needs SFCGAL backend.



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



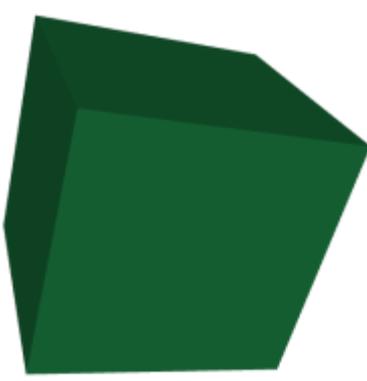
This function supports Polyhedral surfaces.



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

### See also

```
SELECT ST_GeomFromText('POLYHEDRALSURFACE
Z( ((0 0 0, 0 0 1, 0 1 1, 0 1 0, 0 0 0)),
((0 0 0, 0 1 0, 1 1 0, 1 0 0, 0 0 0)),
((0 0 0, 1 0 0, 1 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))
```

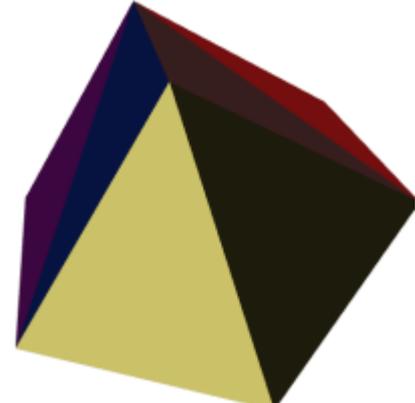


A 3D visualization of a green polyhedron. It has a rectangular base and a slanted top surface. The polyhedron is oriented such that one of its rectangular faces is visible on the left side.

The caption below the image is:

A 3D visualization of a green polyhedron.

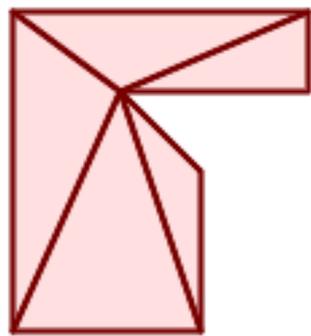
```
SELECT ST_Tessellate(ST_GeomFromText('
POLYHEDRALSURFACE Z( ((0 0 0, 0 0 1, 0 1 1, 0 1
((0 0 0, 0 1 0, 1 1 0, 1 0 0, 0 0
0)), ((0 0 0, 1 0 0, 1 0 1, 0 0 1, 0 0 0)),
((1 1 0, 1 1 1, 1 0 1, 1 0 0, 1 1
0)),
((0 1 0, 0 1 1, 1 1 1, 1 1 0, 0 1
0)), ((0 0 1, 1 0 1, 1 1 1, 0 1 1, 0 0 1)) )')
)
ST_AsText
TIN Z ((0 0 0,0 0 1,0 1 1,0 0 0)),((0 1
0,0,0)0 0,0 1 1,0 1 0)),
((0 0 0,0 1 0,1 1 0,0 0 0)),
((1 0 0,0 0 0,1 1 0,1 0 0)),((0 0
1,1 0 0,1 0 1,0 0 1)),
((0,0 1,0 0 0,1 0 0,0 0 1)),
((1 1 0,1 1 1,1 0 1,1 1 0)),((1 0
0,1 1 0,1 0 1,1 0 0)),
((0 1 0,0 1 1,1 1 1,0 1 0)),((1 1
0,0 1 0,1 1 1,1 1 0)),
((0 1 1,1 0 1,1 1 1,0 1 1)),((0 1
1,0 0 1,1 0 1,0 1 1))
```



A 3D visualization of a multi-colored polyhedron. The faces are colored in various shades: purple, blue, yellow, black, and red. The polyhedron is oriented similarly to the one in the left panel, showing a slanted top surface.

The caption below the image is:

A 3D visualization of a multi-colored polyhedron.

|  |  |
|--|--|
| <pre>SELECT 'POLYGON (( 10 190, 10 70, 80 70, ↵       80 130, 50 160, 120 160, 120 190, 10 190))'</pre>  <p style="text-align: center;">&amp;#xc6d0;&amp;#xbcf8; &amp;#xd3f4;&amp;#xb9ac;&amp;#xace4;</p> | <pre>SELECT   ST_Tessellate('POLYGON (( 10 190, ↵     10 70, 80 70, 80 130, 50 160, 120 160, 120 190, ↵     10 190))', 10)   ST_AsText &amp;#xcd9c;&amp;#xb825;&amp;#xbb3c;::   TIN(6680,130, 50 160,80 70,80 130)),((50 ↵     160,10 190,10 70,50 160)),     ((80 70,50 160,10 70,80 70)) ↵     ,((120 160,120 190,50 160,120 160)),     ((120 190,10 190,50 160,120 190)))</pre>  <p style="text-align: center;">&amp;#xb2e4;&amp;#xb4ec;&amp;#xc5b4;&amp;#xc9c4; &amp;#xd3f4;&amp;#xb9ac;&amp;#xace4;</p> |
|--|--|

**&#xcc38;&#xace0;**

[ST\\_ConstrainedDelaunayTriangles](#), [ST\\_DelaunayTriangles](#)

### 8.20.21 ST\_Volume

**ST\_Volume** — 3&#xcc28;&#xc6d0; &#xc785;&#xccb4;&#xc758; &#xbd80;&#xd53c;&#xb97c; &#xacc4;&#xc0b0;&#xd569;&#xb2 &#xd45c;&#xba74; &#xb3c4;&#xd615;&#xc744; &#xc785;&#xb825;&#xd558;&#xba74; (&#xb2eb;&#xd78c; &#xb3c4;&#xd615;&#xc744; &#xbc18;&#xd658;&#xd560; &#xac83;&#xc785;&#xb2c8;&#xb2e4;.

#### Synopsis

```
float ST_Volume(geometry geom1);
```

**&#xc124;&#xba85;**

2.2.0 &#xbc84;&#xc804;&#xbd80;&#xd130; &#xc0ac;&#xc6a9;&#xd560; &#xc218; &#xc788;&#xc2b5;&#xb2c8;&#xb2e4;.



This method needs SFCGAL backend.



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



**8.21.1 LockRow**

LockRow — Locks a row in a table. The lock is held until the end of the transaction. The lock is held on the row identified by the schema name, table name, and primary key value.

Adds the current transaction identifier and authorization token to a temporary table called temp\_lock\_have\_table.

1.1.3 LockRow: Locks a row in a table. The lock is held until the end of the transaction.

**8.21.2 CheckAuth**

```
SELECT LockRow('towns', '353', 'priscilla');
BEGIN TRANSACTION;
SELECT AddAuth('joey');
UPDATE towns SET the_geom = ST_Translate(the_geom,2,2) WHERE gid = 353;
COMMIT;

-- LockRow: ERROR: UPDATE where "gid" = '353' requires authorization 'priscilla'
```

**8.21.3 LockRow****LockRow****8.21.2 CheckAuth**

CheckAuth — Checks if a user has the right to update a row in a table. The user is identified by the schema name, table name, and primary key value. The user is identified by the schema name, table name, and primary key value.

**Synopsis**

integer **CheckAuth**(text a\_schema\_name, text a\_table\_name, text a\_key\_column\_name);  
integer **CheckAuth**(text a\_table\_name, text a\_key\_column\_name);

**8.21.3 LockRow**

LockRow — Locks a row in a table. The lock is held until the end of the transaction. The lock is held on the row identified by the schema name, table name, and primary key value. The lock is held on the row identified by the schema name, table name, and primary key value.

**Note**

LockRow: Locks a row in a table. The lock is held until the end of the transaction. The lock is held on the row identified by the schema name, table name, and primary key value. The lock is held on the row identified by the schema name, table name, and primary key value.

1.1.3 LockRow: Locks a row in a table. The lock is held until the end of the transaction.



**EnableLongTransactions**

```

ALTER TABLE authorization_table
  ADD COLUMN authorized LONGTEXT;

```

```

ALTER TABLE authorization_table
  ADD COLUMN authorized LONGTEXT;

```

1.1.3 `ALTER TABLE authorization_table ADD COLUMN authorized LONGTEXT;`

**DisableLongTransactions**

```

SELECT EnableLongTransactions();
-- EnableLongTransactions; --
Long transactions support enabled

```

**DisableLongTransactions****DisableLongTransactions****8.21.5 LockRow**

**LockRow** — `ALTER TABLE table_name ADD COLUMN lock_row LONGTEXT;`

**Synopsis**

```

integer LockRow(text a_schema_name, text a_table_name, text a_row_key, text an_auth_token, timestamp expire_dt);
integer LockRow(text a_table_name, text a_row_key, text an_auth_token, timestamp expire_dt);
integer LockRow(text a_table_name, text a_row_key, text an_auth_token);

```

**EnableLongTransactions**

```

ALTER TABLE table_name
  ADD COLUMN lock_row LONGTEXT;

```

1.1.3 `ALTER TABLE table_name ADD COLUMN lock_row LONGTEXT;`

**DisableLongTransactions**

```

SELECT LockRow('public', 'towns', '2', 'joey');
LockRow
-----
1

```



## Synopsis

```
text PostGIS_Extensions_Upgrade();
```

## Description

Packages and upgrades PostGIS extensions to latest version. Only extensions you have installed in the database will be packaged and upgraded if needed. Reports full PostGIS version and build configuration infos after. This is short-hand for doing multiple CREATE EXTENSION .. FROM unpackaged and ALTER EXTENSION .. UPDATE for each PostGIS extension. Currently only tries to upgrade extensions postgis, postgis\_raster, postgis\_sfcgal, postgis\_topology, and postgis\_tiger\_geocoder.

Availability: 2.5.0



### Note

Changed: 3.3.0 support for upgrades from any PostGIS version. Does not work on all systems.  
Changed: 3.0.0 to repack loose extensions and support postgis\_raster.

## Examples

```
SELECT PostGIS_Extensions_Upgrade();
```

```
NOTICE: Packaging extension postgis
NOTICE: Packaging extension postgis_raster
NOTICE: Packaging extension postgis_sfcgal
NOTICE: Extension postgis_topology is not available or not packagable for some reason
NOTICE: Extension postgis_tiger_geocoder is not available or not packagable for some reason
      reason
      postgis_extensions_upgrade
-----
Upgrade completed, run SELECT postgis_full_version(); for details
(1 row)
```

## See Also

Section 3.4, [PostGIS\\_GEOS\\_Version](#), [PostGIS\\_Lib\\_Version](#), [PostGIS\\_LibXML\\_Version](#), [PostGIS\\_PROJ\\_Version](#), [PostGIS\\_Version](#)

### 8.22.2 PostGIS\_Full\_Version

PostGIS\_Full\_Version — Reports full PostGIS version and build configuration infos.

## Synopsis

```
text PostGIS_Full_Version();
```

## Description

Reports full PostGIS version and build configuration infos. Also informs about synchronization between libraries and scripts suggesting upgrades as needed.

## Examples

```
SELECT PostGIS_Full_Version();
                                     postgis_full_version
-----
POSTGIS="3.0.0dev r17211" [EXTENSION] PGSQL="110" GEOS="3.8.0dev-CAPI-1.11.0 df24b6bb" ↔
SFCGAL="1.3.6" PROJ="Rel. 5.2.0, September 15th, 2018"
GDAL="GDAL 2.3.2, released 2018/09/21" LIBXML="2.9.9" LIBJSON="0.13.1" LIBPROTOBUF="1.3.1" ↔
WAGYU="0.4.3 (Internal)" TOPOLOGY RASTER
(1 row)
```

## See Also

Section 3.4, [PostGIS\\_GEOS\\_Version](#), [PostGIS\\_Lib\\_Version](#), [PostGIS\\_LibXML\\_Version](#), [PostGIS\\_PROJ\\_Version](#), [PostGIS\\_Wagyu\\_Version](#), [PostGIS\\_Version](#)

### 8.22.3 PostGIS\_GEOS\_Version

`PostGIS_GEOS_Version` — Returns the version number of the GEOS library.

#### Synopsis

```
text PostGIS_GEOS_Version();
```

#### Description

Returns the version number of the GEOS library, or NULL if GEOS support is not enabled.

#### Examples

```
SELECT PostGIS_GEOS_Version();
       postgis_geos_version
-----
3.11.0-CAPI-1.5.0
(1 row)
```

## See Also

[PostGIS\\_Full\\_Version](#), [PostGIS\\_Lib\\_Version](#), [PostGIS\\_LibXML\\_Version](#), [PostGIS\\_PROJ\\_Version](#), [PostGIS\\_Version](#)

### 8.22.4 PostGIS\_Liblwgeom\_Version

`PostGIS_Liblwgeom_Version` — Returns the version number of the liblwgeom library. This should match the version of PostGIS.

#### Synopsis

```
text PostGIS_Liblwgeom_Version();
```

#### Description

Returns the version number of the liblwgeom library/

## Examples

```
SELECT PostGIS_Liblwgeom_Version();
postgis_liblwgeom_version
-----
2.3.3 r15473
(1 row)
```

## See Also

[PostGIS\\_Full\\_Version](#), [PostGIS\\_Lib\\_Version](#), [PostGIS\\_LibXML\\_Version](#), [PostGIS\\_PROJ\\_Version](#), [PostGIS\\_Version](#)

### 8.22.5 PostGIS\_LibXML\_Version

`PostGIS_LibXML_Version` — Returns the version number of the libxml2 library.

#### Synopsis

text `PostGIS_LibXML_Version()`;

#### Description

Returns the version number of the LibXML2 library.

Availability: 1.5

## Examples

```
SELECT PostGIS_LibXML_Version();
postgis_libxml_version
-----
2.7.6
(1 row)
```

## See Also

[PostGIS\\_Full\\_Version](#), [PostGIS\\_Lib\\_Version](#), [PostGIS\\_PROJ\\_Version](#), [PostGIS\\_GEOS\\_Version](#), [PostGIS\\_Version](#)

### 8.22.6 PostGIS\_Lib\_Build\_Date

`PostGIS_Lib_Build_Date` — Returns build date of the PostGIS library.

#### Synopsis

text `PostGIS_Lib_Build_Date()`;

#### Description

Returns build date of the PostGIS library.

---

## Examples

```
SELECT PostGIS_Lib_Build_Date();
 postgis_lib_build_date
-----
2008-06-21 17:53:21
(1 row)
```

### 8.22.7 PostGIS\_Lib\_Version

PostGIS\_Lib\_Version — Returns the version number of the PostGIS library.

#### Synopsis

```
text PostGIS_Lib_Version();
```

#### Description

Returns the version number of the PostGIS library.

#### Examples

```
SELECT PostGIS_Lib_Version();
 postgis_lib_version
-----
1.3.3
(1 row)
```

#### See Also

[PostGIS\\_Full\\_Version](#), [PostGIS\\_GEOS\\_Version](#), [PostGIS\\_LibXML\\_Version](#), [PostGIS\\_PROJ\\_Version](#), [PostGIS\\_Version](#)

### 8.22.8 PostGIS\_PROJ\_Version

PostGIS\_PROJ\_Version — Returns the version number of the PROJ4 library.

#### Synopsis

```
text PostGIS_PROJ_Version();
```

#### Description

Returns the version number of the PROJ4 library, or NULL if PROJ4 support is not enabled.

#### Examples

```
SELECT PostGIS_PROJ_Version();
 postgis_proj_version
-----
Rel. 4.4.9, 29 Oct 2004
(1 row)
```

**See Also**

[PostGIS\\_Full\\_Version](#), [PostGIS\\_GEOS\\_Version](#), [PostGIS\\_Lib\\_Version](#), [PostGIS\\_LibXML\\_Version](#), [PostGIS\\_Version](#)

**8.22.9 PostGIS\_Wagyu\_Version**

`PostGIS_Wagyu_Version` — Returns the version number of the internal Wagyu library.

**Synopsis**

```
text PostGIS_Wagyu_Version();
```

**Description**

Returns the version number of the internal Wagyu library, or `NULL` if Wagyu support is not enabled.

**Examples**

```
SELECT PostGIS_Wagyu_Version() ;
 postgis_wagyu_version
-----
 0.4.3 (Internal)
(1 row)
```

**See Also**

[PostGIS\\_Full\\_Version](#), [PostGIS\\_GEOS\\_Version](#), [PostGIS\\_PROJ\\_Version](#), [PostGIS\\_Lib\\_Version](#), [PostGIS\\_LibXML\\_Version](#), [PostGIS\\_Version](#)

**8.22.10 PostGIS\_Scripts\_Build\_Date**

`PostGIS_Scripts_Build_Date` — Returns build date of the PostGIS scripts.

**Synopsis**

```
text PostGIS_Scripts_Build_Date();
```

**Description**

Returns build date of the PostGIS scripts.

Availability: 1.0.0RC1

**Examples**

```
SELECT PostGIS_Scripts_Build_Date() ;
 postgis_scripts_build_date
-----
 2007-08-18 09:09:26
(1 row)
```

## See Also

[PostGIS\\_Full\\_Version](#), [PostGIS\\_GEOS\\_Version](#), [PostGIS\\_Lib\\_Version](#), [PostGIS\\_LibXML\\_Version](#), [PostGIS\\_Version](#)

### 8.22.11 PostGIS\_Scripts\_Installed

`PostGIS_Scripts_Installed` — Returns version of the PostGIS scripts installed in this database.

#### Synopsis

```
text PostGIS_Scripts_Installed();
```

#### Description

Returns version of the PostGIS scripts installed in this database.



#### Note

If the output of this function doesn't match the output of [PostGIS\\_Scripts\\_Released](#) you probably missed to properly upgrade an existing database. See the [Upgrading](#) section for more info.

Availability: 0.9.0

#### Examples

```
SELECT PostGIS_Scripts_Installed();
 postgis_scripts_installed
-----
 1.5.0SVN
(1 row)
```

## See Also

[PostGIS\\_Full\\_Version](#), [PostGIS\\_Scripts\\_Released](#), [PostGIS\\_Version](#)

### 8.22.12 PostGIS\_Scripts\_Released

`PostGIS_Scripts_Released` — Returns the version number of the `postgis.sql` script released with the installed PostGIS lib.

#### Synopsis

```
text PostGIS_Scripts_Released();
```

#### Description

Returns the version number of the `postgis.sql` script released with the installed PostGIS lib.



#### Note

Starting with version 1.1.0 this function returns the same value of [PostGIS\\_Lib\\_Version](#). Kept for backward compatibility.

Availability: 0.9.0





2.2.0

**Note**

GDAL  
GDAL\_DATA  
&#xc218; &#xc788;&#xc2b5;&#xb2c8;&#xb2e4;.

**&#xc608;&#xc2dc;**

```
postgis.gdal_datapath
```

```
SET postgis.gdal_datapath TO '/usr/local/share/gdal.hidden';
SET postgis.gdal_datapath TO default;
```

```
&#xd2b9;&#xc815; &#xb370;&#xc774;&#xd130;&#xbca0;&#xc774;&#xc2a4;&#xb97c; &#xb300;&#xc0c1;&#xc73c;&#xb85c;
&#xc708;&#xb3c4;&#xc6b0; &#xc0c1;&#xc5d0;&#xc11c; &#xc124;&#xc815;&#xd574;&#xbcf4;&#xc2ed;&#xc2dc;&#xc624;
```

```
ALTER DATABASE gisdb
SET postgis.gdal_datapath = 'C:/Program Files/PostgreSQL/9.3/gdal-data';
```

**&#xcc38;&#xace0;****PostGIS\_GDAL\_Version, ST\_Transform****8.23.3 postgis.gdal\_enabled\_drivers**

postgis.gdal\_enabled\_drivers — PostGIS  
GDAL  
GDAL

**&#xc124;&#xba85;**

PostGIS  
&#xc124;&#xc815;&#xd558;&#xb294; &#xc124;&#xc815; &#xc635;&#xc158;&#xc785;&#xb2c8;&#xb2e4;. GDAL  
&#xbcc0;&#xc218; GDAL\_SKIP  
&#xc124;&#xc815; &#xd30c;&#xc77c; postgresql.conf  
&#xc124;&#xc815;&#xd560; &#xc218; &#xc788;&#xc2b5;&#xb2c8;&#xb2e4;. &#xb610; &#xc5f0;&#xcacb0;&#xc774;&#xb098;  
&#xc0c1;&#xd638;&#xc98;&#xb9ac; &#xb2e8;&#xacc4;&#xc5d0;&#xc11c;&#xb3c4; &#xc124;&#xc815;&#xd560; &#xc218;  
&#xc788;&#xc2b5;&#xb2c8;&#xb2e4;.

PostgreSQL  
&#xbcc0;&#xc218; POSTGIS\_GDAL\_ENABLED\_DRIVERS  
&#xc758; &#xcd08;&#xae30;&#xac12;&#xc744; &#xc124;&#xc815;&#xd560; &#xc218;&#xb3c4; &#xc788;&#xc2b5;&#xb2c8;&#xb2e4;&#xb4dc;&#xb77c;&#xc774;&#xb84;&#xc758; &#xcd95;&#xc57d;&#xba85; &#xb610;&#xb294; &#xcf54;&#xb4dc;&#xb97c;  
&#xd1b5;&#xd574; &#xd65c;&#xc131;&#xd654;&#xb41c; GDAL &#xd2b9;&#xd654; &#xb4dc;&#xb77c;&#xc774;&#xb84;&#xb9c0;&#xc815;&#xd560; &#xc218; &#xc788;&#xc2b5;&#xb2c8;&#xb2e4;. &#xb4dc;&#xb77c;&#xc774;&#xb84;&#xc758;  
&#xcd95;&#xc57d;&#xba85; &#xb610;&#xb294; &#xcf54;&#xb4dc;&#xb294; GDAL &#xb798;&#xc2a4;&#xd130; &#xd615;&#xc5d0;&#xc11c; &#xc3e;&#xc744; &#xc218; &#xc788;&#xc2b5;&#xb2c8;&#xb2e4;. &#xac01; &#xb4dc;&#xb77c;&#xc774;&#xc0ac;&#xc774;&#xc5d0; &#xacf5;&#xb31;&#xc744; &#xc0bd;&#xc785;&#xd558;&#xba74; &#xbcf5;&#xc218;&#xc758;  
&#xb4dc;&#xb77c;&#xc774;&#xb84;&#xb97c; &#xc9c0;&#xc815;&#xd560; &#xc218; &#xc788;&#xc2b5;&#xb2c8;&#xb2e4;.





```
ALTER DATABASE mygisdb SET postgis.backend = sfcgal;
```

Setting for whole database cluster. You need to reconnect to the database for changes to take effect.

```
--writes to postgres.auto.conf
ALTER SYSTEM postgis.enable_outdb_rasters = true;
--Reloads postgres conf
SELECT pg_reload_conf();
```

[postgis.gdal\\_enabled\\_drivers](#) [postgis.gdal\\_vsi\\_options](#)

### 8.23.5 postgis.gdal\_vsi\_options

`postgis.gdal_vsi_options` — DB configuration parameter

A string configuration to set options used when working with an out-db raster. [Configuration options](#) control things like how much space GDAL allocates to local data cache, whether to read overviews, and what access keys to use for remote out-db data sources.

2.2.0 configuration value

```
postgis.enable_outdb_rasters configuration parameter
SET postgis.gdal_vsi_options = 'AWS_ACCESS_KEY_ID=xxxxxxxxxxxxxxxxx AWS_SECRET_ACCESS_KEY=
YYYYYYYYYYYYYYYYYYYYYYYYYYYYYYY';
```

Set `postgis.gdal_vsi_options` just for the *current transaction* using the `LOCAL` keyword:

```
SET LOCAL postgis.gdal_vsi_options = 'AWS_ACCESS_KEY_ID=xxxxxxxxxxxxxxxxx
AWS_SECRET_ACCESS_KEY=YYYYYYYYYYYYYYYYYYYYYYYYYYYYYYY';
```

[postgis.enable\\_outdb\\_rasters](#) [postgis.gdal\\_enabled\\_drivers](#)

## 8.24 Troubleshooting Functions

### 8.24.1 PostGIS\_AddBBBox

`PostGIS_AddBBBox` — Add bounding box to the geometry.

#### Synopsis

```
geometry PostGIS_AddBBBox(geometry geomA);
```

## Description

Add bounding box to the geometry. This would make bounding box based queries faster, but will increase the size of the geometry.



### Note

Bounding boxes are automatically added to geometries so in general this is not needed unless the generated bounding box somehow becomes corrupted or you have an old install that is lacking bounding boxes. Then you need to drop the old and readd.



This method supports Circular Strings and Curves

## Examples

```
UPDATE sometable
SET geom = PostGIS_AddBBox(geom)
WHERE PostGIS_HasBBox(geom) = false;
```

## See Also

[PostGIS\\_DropBBox](#), [PostGIS\\_HasBBox](#)

### 8.24.2 PostGIS\_DropBBox

PostGIS\_DropBBox — Drop the bounding box cache from the geometry.

## Synopsis

geometry **PostGIS\_DropBBox**(geometry geomA);

## Description

Drop the bounding box cache from the geometry. This reduces geometry size, but makes bounding-box based queries slower. It is also used to drop a corrupt bounding box. A tale-tell sign of a corrupt cached bounding box is when your ST\_Intersects and other relation queries leave out geometries that rightfully should return true.



### Note

Bounding boxes are automatically added to geometries and improve speed of queries so in general this is not needed unless the generated bounding box somehow becomes corrupted or you have an old install that is lacking bounding boxes. Then you need to drop the old and readd. This kind of corruption has been observed in 8.3-8.3.6 series whereby cached bboxes were not always recalculated when a geometry changed and upgrading to a newer version without a dump reload will not correct already corrupted boxes. So one can manually correct using below and readd the bbox or do a dump reload.



This method supports Circular Strings and Curves

## Examples

```
--This example drops bounding boxes where the cached box is not correct
--The force to ST_AsBinary before applying Box2D forces a ↔
  recalculation of the box, and Box2D applied to the table ↔
  geometry always
-- returns the cached bounding box.
UPDATE sometable
SET geom = PostGIS_DropBBox(geom)
WHERE Not (Box2D(ST_AsBinary(geom)) = Box2D(geom));

UPDATE sometable
SET geom = PostGIS_AddBBox(geom)
WHERE Not PostGIS_HasBBOX(geom);
```

## See Also

[PostGIS\\_AddBBox](#), [PostGIS\\_HasBBox](#), [Box2D](#)

### 8.24.3 PostGIS\_HasBBox

`PostGIS_HasBBox` — Returns TRUE if the bbox of this geometry is cached, FALSE otherwise.

## Synopsis

boolean `PostGIS_HasBBox`(geometry geomA);

## Description

Returns TRUE if the bbox of this geometry is cached, FALSE otherwise. Use [PostGIS\\_AddBBox](#) and [PostGIS\\_DropBBox](#) to control caching.



This method supports Circular Strings and Curves

## Examples

```
SELECT geom
FROM sometable WHERE PostGIS_HasBBox(geom) = false;
```

## See Also

[PostGIS\\_AddBBox](#), [PostGIS\\_DropBBox](#)



PostGIS 2 &#xc804;&#xbd80;&#xd130; &#xae30;&#xbcfc8; &#xb3c4;&#xd615; &#xc5f0;&#xc0b0;&#xc790; &#xd074;&#xb798;&#xc2a4;&#xac00; gist\_geometry\_ops&#xc5d0;&#xc11c; gist\_geometry\_ops\_2d&#xb85c; &#xbcc0;&#xc000; gist\_geometry\_ops&#xb294; &#xc644;&#xc804;&#xd788; &#xc0ad;&#xc81c;&#xb418;&#xc5c8;&#xc2b5;&#xb2c8;&#xb2e4; PostGIS 2 &#xc804;&#xbd80;&#xd130; 3D&#xb97c; &#xc9c0;&#xc6d0;&#xd558;&#xae30; &#xc704;&#xd574; N-D &#xacf5;&#xac04; &#xc778;&#xb371;&#xc2a4;&#xb97c; &#xb3c4;&#xc785;&#xd588;&#xb294;&#xb370;., &#xad6c; &#xba85;&#xce6d; gist\_geometry\_ops&#xc000; &#xbd80;&#xc815;&#xd655;&#xd558;&#xae0; &#xd63c;&#xb3d9;&#xb41c; &#xc5ec;&#xaca8;&#xc84c;&#xae30; &#xb54c;&#xbb38;&#xc785;&#xb2c8;&#xb2e4;.&#xd14c;&#xc774;&#xbe14; &#xbcc0; &#xc778;&#xb371;&#xc2a4;&#xb97c; &#xc0dd;&#xc131;&#xd558;&#xb294; &#xacfc;&#xc815;&#xc758; &#xc77c;&#xbd80; &#xba87;&#xba87; &#xad6c; &#xc84;&#xc804; &#xc751;&#xc6a9; &#xd504;&#xb85c;&#xadf8;&#xb7a8;&#xc740; &#xc5f0;&#xc0b0;&#xc790; &#xd074;&#xb798;&#xc2a4; &#xba85;&#xc744; &#xc815;&#xd655;&#xd788; &#xcc38;&#xc800; &#xae30;&#xbcfc8; 2D &#xc778;&#xb371;&#xc2a4;&#xb97c; &#xc0ac;&#xc6a9;&#xd558;&#xae0;&#xc790; &#xd558;&#xc000; &#xacbd;&#xc6b0; &#xc774;&#xb807;&#xac8c; &#xc815;&#xd655;&#xd788; &#xcc38;&#xc870;&#xd560; &#xd544;&#xc600; &#xc5c6;&#xc2b5;&#xb2c8;&#xb2e4;.&#xb530;&#xb77c;&#xc11c; &#xc774;&#xb7f0; &#xacbd;&#xc6b0;&#xb77c;&#xb800; &#xc778;&#xb371;&#xc2a4; &#xc0dd;&#xc131; &#xba85;&#xb839;&#xc5b4;&#xb97c; &#xb2e4;&#xc74c;&#xacfc; &#xc19;&#xc774; &#xbcc0;&#xacbd;&#xd558;&#xc2ed;&#xc2dc;&#xc624;.&#xb098;&#xc05c; &#xc608;&#xc5d0;&#xc11c;

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

&#xc88b;&#xc740; &#xc608;&#xb85c;:

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

&#xc0ac;&#xc6a9;&#xc790;&#xc000; &#xc5f0;&#xc0b0;&#xc790; &#xd074;&#xb798;&#xc2a4;&#xb97c; &#xc9c0;&#xc81c; &#xd558;&#xb294; &#xc720;&#xc77c;&#xd55c; &#xacbd;&#xc6b0;&#xb294; &#xb2e4;&#xc74c;&#xacfc; &#xc19;&#xc77c; 3D &#xacf5;&#xac04; &#xc778;&#xb371;&#xc2a4;&#xb97c; &#xc0dd;&#xc131;&#xd558;&#xae0;&#xc790; &#xd560; &#xb54c;&#xbfd0;&#xc785;&#xb2c8;&#xb2e4;:

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

&#xc548;&#xd0c0;&#xae5d;&#xac8c;&#xb3c4; &#xc0ac;&#xc6a9;&#xc790;&#xc000; &#xbcc0;&#xacbd;&#xd560; &#xc218; &#xc5c6;&#xb294;., &#xad6c; &#xc84;&#xc804; gist\_geometry\_ops&#xc000; &#xd558;&#xb4dc; &#xcf54;&#xb529;&#xb41c; &#xc788;&#xb294; &#xcfef4;&#xd30c;&#xc77c;&#xb41c; &#xcf54;&#xb4dc;&#xb97c; &#xc368;&#xc57c;&#xb9cc; &#xd560; &#xacbd;&#xc6b0;., PostGIS 2.0.2 &#xc774;&#xc0c1; &#xc84;&#xc804;&#xc5d0; &#xd328;&#xd0a4;&#xc9d5;&#xc000; legacy\_gist.sql &#xc744; &#xc774;&#xc6a9;&#xd574;&#xc11c; &#xad6c; &#xc84;&#xc804; &#xd074;&#xb798;&#xc0dd;&#xc131;&#xd560; &#xc218; &#xc788;&#xc2b5;&#xb2c8;&#xb2e4;.&#xd558;&#xc9c0;&#xb9cc; &#xc774; &#xd574;&#xacb0; &#xc29;&#xc95;&#xc744; &#xc4f8; &#xacbd;&#xc6b0;., &#xc774;&#xd6c4; &#xc5b4;&#xb5a4; &#xc2dc;&#xc810;&#xc5d0;&#xc11c; &#xd574;&#xb2f9; &#xc778;&#xb371;&#xc2a4;&#xb97c; &#xc0ad;&#xc81c;&#xd558; &#xc5f0;&#xc0b0;&#xc790; &#xd074;&#xb798;&#xc2a4; &#xc5c6;&#xc774; &#xc7ac;&#xc0dd;&#xc131;&#xd558;&#xb294; &#xc83;&#xc774; &#xc88b;&#xc2b5;&#xb2c8;&#xb2e4;.&#xc774;&#xb807;&#xac8c; &#xd558;&#xba74; &#xd5a5;&#xd074; &#xb2e4;&#xc2dc; &#xc5c5;&#xadf8;&#xb808;&#xc774;&#xb4dc;&#xb97c; &#xd574;&#xc57c; &#xd560; &#xb54c; &#xc218;&#xae0;&#xb97c; &#xb35c;&#xac8c; &#xb420; &#xc83;&#xc785;&#xb2c8;&#xb2e4;.

4. *PostgreSQL 9.0* &#xc744; &#xc6b4;&#xc6a9; &#xc911;&#xc774; &#xba70; *OpenJump, SafeFME* &#xc640; &#xadf8; &#xb9ac; &#xba87; &#xb2e4;&#xb978; &#xd234;&#xb4e4;&#xc5d0;&#xc11c; &#xc9c0;&#xc624; &#xba54;&#xd2b8; &#xb97c; &#xb354;&#xb294; &#xc77d; &#xc70; &#xb098; &#xbcfc; &#xc218; &#xc5c6;&#xc2b5;&#xb2c8;&#xb2e4;.

PostgreSQL 9.0+&#xc5d0;&#xc11c;., *bytea* &#xb370;&#xc774;&#xd130;&#xb97c; &#xc704;&#xd55c; &#xb514;&#xd3f4;&#xc778;&#xcf54;&#xb529;&#xc740; *hex* &#xb85c; &#xb14;&#xb00c;&#xc5c8;&#xae0; &#xc608;&#xc804; *JDBC* &#xb4dc;&#xb77c;&#xc774;&#xbe0c;&#xb294; &#xc5ec;&#xc804;&#xd788; *escape format* &#xc744; &#xcde8;&#xd569;&#xc774;&#xc83;&#xc740; &#xc608;&#xc804; *JDBC* &#xb4dc;&#xb77c;&#xc774;&#xc84;&#xb97c; &#xc0ac;&#xc6a9;&#xc5b4;&#xd50c;&#xb9ac;&#xcf00;&#xc774;&#xc158;&#xc774;&#xb098; &#xc624;&#xb798;&#xb41c; *ST\_AsBinary* &#xc791;&#xb3d9;&#xc744; &#xc694;&#xd558;&#xb294; &#xc608;&#xc804; *npgsql* &#xb4dc;&#xb77c;&#xc774;&#xc84; &#xc0ac;&#xc6a9;&#xd558;&#xb294; *.NET* &#xc5b4;&#xd50c;&#xb9ac;&#xcf00;&#xc774;&#xc158; &#xc19;&#xc740; &#xba87;&#xba87; &#xc5b4;&#xd50c;&#xb9ac;&#xcf00;&#xc774;&#xc158;&#xc5d0; &#xc601;&#xd5a5;&#xc744; &#xb07c;&#xce69;&#xb2c8;&#xb2e4;.&#xc774;&#xc83;&#xc744; &#xb2e4;&#xc2dc; &#xc791;&#xb3d9;&#xc2dc;&#xd074; &#xc704;&#xd55c; &#xb450; &#xc000;&#xc9c0; &#xc811;&#xadfc; &#xc29;&#xc95;&#xc774; &#xc788;&#xc2b5;&#xb2c8; &#xb97c; &#xcd5c;&#xc2e0; PostgreSQL 9.0 &#xc84;&#xc804;&#xc73c;&#xb85c; &#xc5c5;&#xadf8;&#xb808;&#xc77c; &#xc2dc;&#xd0ac; &#xc218; &#xc788;&#xc2b5;&#xb2c8;&#xb2e4;.&#xcd5c;&#xc2e0; PostgreSQL &#xc84;&#xc804;&#xc11c; &#xb2e4;&#xc6b4;&#xb1b;&#xc73c;&#xc2e4; &#xc218;

<http://jdbc.postgresql.org/download.html> &#xc5d0;&#xc11c; &#xb2e4;&#xc6b4;&#xb1b;&#xc73c;&#xc2e4; &#xc218;

&#xc788;&#xc2b5;&#xb2c8;&#xb2e4;&#xb9cc;&#xc57d;.NET app&#xc744; &#xc2e4;&#xd589;&#xc911;&#xc774;&#xb77d; Npgsql 2.0.11&#xb610;&#xb294; &#xadf8; &#xc774;&#xc0c1;&#xc758; &#xbc84;&#xc804;&#xc744; &#xc0ac;&#xc6a9;&#xc218; &#xc788;&#xc2b5;&#xb2c8;&#xb2e4;. &#xc774;&#xb294; [Francisco Figueiredo's Npgsql 2.0.11 released blog entry](#)&#xc5d0; &#xc124;&#xba85;&#xb418;&#xc5b4; &#xc788;&#xb294; &#xac83;&#xacfc; &#xac19;&#xc774; [http://pgfoundry.org/frs/?group\\_id=1000140](http://pgfoundry.org/frs/?group_id=1000140)&#xc5d0;&#xc11c; &#xb2e4;&#xc6b4;&#xb1b;&#xc73c;&#xc2e4; &#xc218; &#xc788;&#xc2b5;&#xb2c8;&#xb2e4;. &#xb9cc;&#xc57d; PostgreSQL driver&#xb97c; &#xc5c5;&#xadf8;&#xb808;&#xc774; &#xd558;&#xb294; &#xac83;&#xc774; &#xc635;&#xc158;&#xc0ac;&#xd56d;&#xc774; &#xc544;&#xb2c8;&#xb77c;&#xba &#xc544;&#xb798;&#xb97c; &#xc785;&#xb825;&#xd568;&#xc73c;&#xb85c;&#xc368; &#xc774;&#xc804; &#bc29;&#xc2 &#xb514;&#xd3f4;&#xd2b8; &#xc124;&#xc815;&#xc744; &#xd560; &#xc218; &#xc788;&#xc2b5;&#xb2c8;&#xb2e4.;

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

5. &#xc9c0;&#xc624;&#xba54;&#xd2b8;&#xb9ac; &#xcceec;&#xb7fc;&#xc744; &#xbcf4;&#xae30; &#xc704;&#xd574; PgAdmin&#xc744; &#xc0ac;&#xc6a9;&#xd558;&#xb824;&#xcace0; &#xd588;&#xc73c;&#xb098; &#xbe44;&#xc5b4;&#xc78 &#xc65c; &#xadf8;&#xb7f0;&#xc9c0;&#xc694;?

PgAdmin&#xc740; &#xd070; &#xc9c0;&#xc624;&#xba54;&#xd2b8;&#xb9ac;&#xc5d0; &#xb300;&#xd574; &#xc544;&#xbba &#xbcf4;&#xc5ec;&#xc8fc;&#xc9c0; &#xc54a;&#xc2b5;&#xb2c8;&#xb2e4;. &#xc9c0;&#xc624;&#xba54;&#xd2b8;&#xb9ac &#xcceec;&#xb7fc;&#xc5d0; &#xc788;&#xb294; &#xb370;&#xc774;&#xd130;&#xb97c; &#xac80;&#xc99d;&#xd558;&#xb29 &#xac00;&#xc7a5; &#xc88b;&#xc740; &#xbc29;&#xc95;&#xc740;?

```
-- &#xc774; &#xc2a4; &#xd06c; &#xb9bd; &#xd2b8; &#xb294; &#xc0ac; &#xc6a9; &#xc790; &#xc758; &#xcbaa8; &#xb4e0; geom &#xd56d; &#xbaa9; &#xc774; &#xcc28; &#xc788; &#xb2e4; &#xba74; &#xc5b4; &#xb5a4; &#xb808; &#xcf54; &#xb4dc; &#xb3c4; &#xbc18; &#xd658; &#xd558; &#xc9c0; &#xc54a; &#xc2b5; &#xb2c8; &#xb2e4;.
SELECT somefield FROM mytable WHERE geom IS NULL;
```

```
-- &#xc5bc; &#xb9c8; &#xb098; &#xd070; &#xc9c0; &#xc624; &#xba54; &#xd2b8; &#xb9ac; &#xac00; &#xcffc; &#xb9ac; &#xb418; &#xb294; &#xc9c0; &#xc54c; &#xcace0; &#xc2f6; &#xb2e4; &#xba74;
-- &#xc9c0; &#xc624; &#xba54; &#xd2b8; &#xb9ac; &#xcceec; &#xb7fc; &#xc5d0; &#xb4e4; &#xc5b4; &#xc788; &#xb294; &#xc5b4; &#xb5a4; &#xb3c4; &#xd615; &#xc758; &#xc810; &#xc218; &#xc911; &#xac00; &#xc7a5; &#xd070; &#xac83; &#xc744; &#xb9d0; &#xd574; &#xc90c;
SELECT MAX(ST_NPoints(geom)) FROM sometable;
```

6. &#xc5b4;&#xb5a0;&#xd55c; &#xc885;&#xb958;&#xc758; &#xc9c0;&#xc624;&#xba54;&#xd2b8;&#xb9ac; &#xc624;&#xbbe &#xc800;&#xc7a5;&#xd560; &#xc218; &#xc788;&#xc2b5;&#xb2c8;&#xae4c;?

&#xd3ec;&#xc778;&#xd2b8; &#xb77c;&#xc778;&#xc2a4;&#xd2b8;&#xb9c1; &#xd3f4;&#xb9ac;&#xcace4; &#xba40;&#xd &#xba40;&#xd2f0;&#xb77c;&#xc778;&#xc2a4;&#xd2b8;&#xb9c1; &#xba40;&#xd2f0;&#xd3f4;&#xb9ac;&#xcace4; &#xadf &#xc9c0;&#xc624;&#xba54;&#xd2b8;&#xb9ac;&#xcceec;&#xb809;&#xc158;(GeometryCollection) &#xb3c4;&#xd615;&#xc7 &#xc800;&#xc7a5;&#xd560; &#xc218; &#xc788;&#xc2b5;&#xb2c8;&#xb2e4;. PostGIS 2.0 &#xc774;&#xc0c1; &#bc84;&# TINS &#bc0f; &#xb2e4;&#ba74;&#ccb4; &#xd45c;&#ba74;(Polyhedral Surface) &#xb3c4; &#xae30;&#bcf8; &#xb3c4;& &#xc720;&#xd615;&#xc73c;&#xb85c; &#xc800;&#xc7a5;&#xd560; &#xc218; &#xc788;&#xc2b5;&#xb2c8;&#xb2e4;. Z, M, ZM &#xd655;&#xc7a5;&#xc790;&#xb97c; &#xac00;&#xc9c0;&#xb294; &#xc624;&#xd508;GIS WKT &#xd615;&#xc2 GIS Well Known Text Format)&#xc774; &#xc774;&#xb7f0; &#xb3c4;&#xd615;&#xb4e4;&#xc744; &#xc9c0;&#xc815;&#xd5 &#xc788;&#xc2b5;&#xb2c8;&#xb2e4;. &#xd604;&#xc7ac; &#xb2e4;&#xc74c; &#xc138; &#xac00;&#xc9c0; &#xb370;&#xc &#xc720;&#xd615;&#xc744; &#xc9c0;&#xc6d0;&#xd569;&#xb2c8;&#xb2e4;. &#xadf8; &#xc138; &#xac00;&#xc9c0; &#xc720;&#xd615;&#xc740; &#xc21;&#xc815;&#xc2dc; &#xd3c9;&#ba74;&#xc88c;&#xd45c;&#xacc4;&#xb97c; &#xc774;&#xc6a9;&#xd558;&#xb294; &#xd45c;&#xc900; OGC &#xb3c4;&#xd615; &#xb370;&#xc774;&#xd130; &#xc720; &#xad6c;&#xc4b4; &#xb610;&#xb294; &#xd3b8;&#xad6c;&#ccb4; &#xc0c1;&#xc5d0;&#xc11c; &#xacc4;&#xc0b0;&#xd5 &#xc21;&#xc9c0;&#xc88c;&#xd45c;&#xacc4;&#xb97c; &#xc774;&#xc6a9;&#xd558;&#xb294; &#xc9c0;&#xb9ac; &#xb370;&#xc774;&#xd130; &#xc720;&#xd615; &#xadf8;&#xb9ac;&#xcace0; PostGIS &#xcacf5;&#xc04; &#xc720;&#xd61 &#xacc4;&#bcf4;&#xc5d0; &#xc0c8;&#xb86d;&#xc8c; &#xcd94;&#xc00;&#xb41c; &#xb798;&#xc2a4;&#xd130; &#xb370;&#xc774;&#xd130; &#xbd84;&#xc11d; &#bc0f;&#xc800;&#xc7a5;&#xc744; &#xc704;&#xd55c; &#xb798;&#xc2 &#xb798;&#xc2a4;&#xd130; &#xc804;&#xc6a9; FAQ&#xb3c4; &#xc788;&#xc2b5;&#xb2c8;&#xb2e4;. &#xb354; &#xc790;&#xc124;&#xba85;&#xc740; Chapter 13 &#xcacf; Chapter 12 &#xb97c; &#xc38;&#xc870;&#xd558;&#xc2ed;&#xc2dc;&#xc

7. &#xd63c;&#xb3d9;&#xb418;&#xb124;&#xc694;. &#xc81c;&#xc00; &#xc9c0;&#xc624; &#xba54;&#xd2b8;&#xb9ac;- geometry- &#xb610;&#xb294; &#xc9c0;&#xd615;-geography- &#xc911; &#xc5b4;&#xb5a4; &#xb370;&#xc774;&#xd130;&#xc0ac;&#xc6a9;&#xd558;&#xc5ec; &#xc800;&#xc7a5;&#xd574;&#xc57c; &#xd569;&#xb2c8;&#xae4c;?

geography (long range distance)
   
 (SRID)
   
 Section 4.3.3 and function type matrix

8. geography (geographic region)
   
 geography (geography)
   
 Section 4.3.4
9. GIS
   
 Section 4.3.4

First, you need to create a table with a column of type "geometry" or "geography" to hold your GIS data. Storing geography type data is a little different than storing geometry. Refer to Section 4.3 for details on storing geography.

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

PostGIS
   
 Section 2.2

geometry; SQL insert geometry GIS "well-knowns text"

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

object reference GIS

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

geom

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

- SQL

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

11. `CREATE INDEX [indexname] ON [tablename] USING GIST ( [geometrycolumn] );`

"USING GIST" GiST(Generalized Search Tree)

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

"USING GIST" GiST(Generalized Search Tree)

**Note**



GiST

PostgreSQL

12. PostgreSQL R-Tree indexes

PostgreSQL R-Tree

- PostgreSQL R-Tree
- PostgreSQL R-Tree

13. `AddGeometryColumn()` ?

`OpenGIS` meta-data  
`geometry_columns` meta-data  
`MapServer` on-the-fly reprojection  
`Geometry` SRID

14. `ST_DWithin()` ?

`ST_DWithin(geometry, geometry, distance)`  
`POINT(1000 1000)`  
`ST_AsEWKT()`  
`ST_AsText()`

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

15. `ST_Transform()` ?

To perform a reprojection, both the source and destination coordinate systems must be defined in the `SPATIAL_REF_SYS` table, and the geometries being reprojected must already have an SRID set on them. Once that is done, a reprojection is as simple as referring to the desired destination SRID. The below projects a geometry to NAD 83 long lat. The below will only work if the `sr_id` of `geom` is not -1 (not undefined spatial ref)

```
SELECT ST_Transform(geom, 4269) FROM geotable;
```

16. `ST_AsEWKT()` / `ST_AsText()` ?

```
&#xe48; &#xd544; &#xb4dc; &#xb97c; &#xbc18; &#xd658; &#xd588; &#xc2b5; &#xb2c8; &#xb2e4;. &#xc65c; &#xc774; &#xb
&#xac83; &#xc778; &#xac00; &#xc694; ?
```

```
&#xc544; &#xb9c8; &#xb3c4; &#xd070; &#xd14d; &#xc2a4; &#xd2b8; &#xb97c; &#xbcf4; &#xc5ec; &#xc8fc; &#xc9c0;
&#xbabb; &#xd558; &#xb294; PgAdmin &#xc774; &#xb098; &#xb2e4; &#xb978; &#xd234; &#xc744; &#xc4f0; &#xace0;
&#xc788; &#xae30; &#xb54c; &#xbb38; &#xc77c; &#xac83; &#xc785; &#xb2c8; &#xb2e4;. &#xb9cc; &#xc57d; &#xc0ac; &#xc6
&#xc9c0; &#xc624; &#xba54; &#xd2b8; &#xb9ac; &#xac00; &#xcda9; &#bd84; &#xd788; &#xd06c; &#xb2e4; &#xba74;.
&#xc774; &#xb7f0; &#xd234; &#xc5d0; &#xc11c; &#xb294; &#xacf5; &#bc31; &#xc73c; &#xb85c; &#xb098; &#xd0c0; &#xb0a
&#xac83; &#xc785; &#xb2c8; &#xb2e4;. PSQL &#xc744; &#xc0ac; &#xc6a9; &#xd558; &#xc2dc; &#ba74; WKT &#xb85c;
&#xbcf4; &#xac70; &#xb098; &#xcd9c; &#xb825; &#xd558; &#xc2e4; &#xc218; &#xc788; &#xc2b5; &#xb2c8; &#xb2e4;.
```

```
--To check number of geometries are really blank
SELECT count(gid) FROM geotable WHERE geom IS NULL;
```

17. *ST\_Intersects* &#xb97c; &#xd588; &#xb294; &#xb370; &#xb450; &#xc9c0; &#xc624; &#ba54; &#xd2b8; &#b9ac; &#xac00; &#xad50; &#xcc28; &#xd558; &#xc9c0; &#xc54a; &#xb294; &#xb2e4; &#xb294; &#ba54; &#xc2dc; &#xc9c0; &#xb97c; &#bc1b; &#xc558; &#xc2b5; &#xb2c8; &#xb2e4;. &#xad50; &#xcc28; &#xd558; &#xace0; &#xc788; &#xc74c; &#xc744; &#xb0b4; &#xac00; &#xc544; &#xb294; &#xb370; &#xb9d0; &#xc774; &#xc8e0; !!! &#xc65c; &#xc774; &#xb7f4; &#xae4c; &#xc77c; &#bc18; &#xc801; &#xc73c; &#xb85c; &#xb450; &#xac00; &#xc9c0; &#xacbd; &#xc6b0; &#xc5d0; &#xb1c; &#xc0ac; &#xc6a9; &#xc790; &#xc758; &#xc9c0; &#xc624; &#ba54; &#xd2b8; &#b9ac; &#xac00; &#xc720; &#xd6a8; &#xd558; &#xc54a; &#xac70; &#xb098; - **ST\_IsValid** &#xd655; &#xc778;. &#xc0ac; &#xc6a9; &#xc790; &#xac00; ST\_AsText &#xac00; &#xc798; &#xb77c; &#xb0b8; &#xc720; &#xd6a8; &#xc22b; &#xc790; &#xb97c; &#xac00; &#xc9c0; &#xace0; &#xad50; &#xcc28; &#xd310; &#xb2e8; &#xd558; &#xb294; &#xacbd; &#xc6b0; &#xc785; &#xb2c8; &#xb2e4;. &#xc18c; &#xc22b; &#xc810; &#xc774; &#xd558; &#xc758; &#xb9cc; &#xc740; &#xc22b; &#xc790; &#xb4e4; &#xc774; &#xc0ac; &#xc6a9; &#xc790; &#xc5d0; &#bcf4; &#xc774; &#xc9c0; &#xc54a; &#xc2b5; &#xb2c8; &#xb2e4;.
18. &#xc800; &#xb294; PostGIS &#xb97c; &#xc0ac; &#xc6a9; &#xd558; &#xc5ec; &#xc18c; &#xd504; &#xd2b8; &#xc6e8; &#xc5b4; &#xac1c; &#bc1c; &#xd574; &#xcd9c; &#xc2dc; &#xd558; &#xc600; &#xc2b5; &#xb2c8; &#xb2e4;. &#xc81c; &#xc18c; &#xd504; PostGIS &#xcc98; &#xb7fc; GPL &#xc744; &#xb77c; &#xc774; &#xc13c; &#xc2a4; &#xb85c; &#xac00; &#xc838; &#xc57c; &#xd558; &#xb098; &#xc694; ? &#xb9cc; &#xc57d; PostGIS &#xb97c; &#xc0ac; &#xc6a9; &#xd560; &#xacbd; &#xc6b0; &#xc81c; &#xcf54; &#xb4dc; &#xb97c; &#xacf5; &#xac1c; &#xd574; &#xc57c; &#xb9cc; &#xd558; &#xb098; &#xc694; ?

Almost certainly not. As an example, consider Oracle database running on Linux. Linux is GPL, Oracle is not: does Oracle running on Linux have to be distributed using the GPL? No. Similarly your software can use a PostgreSQL/PostGIS database as much as it wants and be under any license you like. The only exception would be if you made changes to the PostGIS source code, and *distributed your changed version* of PostGIS. In that case you would have to share the code of your changed PostGIS (but not the code of applications running on top of it). Even in this limited case, you would still only have to distribute source code to people you distributed binaries to. The GPL does not require that you *publish* your source code, only that you share it with people you give binaries to. The above remains true even if you use PostGIS in conjunction with the optional CGAL-enabled functions. Portions of CGAL are GPL, but so is all of PostGIS already: using CGAL does not make PostGIS any more GPL than it was to start with.

19. *Why are the results of overlay operations and spatial predicates sometimes inconsistent?*

This is usually presented as a specific case, such as

- Why is `ST_Contains ( A, ST_Intersection ( A, B ) )` false ?
- Why is `ST_Contains ( ST_Union ( A, B ), A )` false ?
- Why is `ST_Union ( A, ST_Difference ( A, B ) )` not equal to A ?
- Why does `ST_Difference ( A, B )` intersect the interior of B ?

The reason is that PostGIS represents geometry and performs operations using finite-precision floating-point numbers. This provides the illusion of computing using real numbers - but it's only an illusion. Inevitably, small inaccuracies occur, which cause results of different operations to be slightly inconsistent. Furthermore, PostGIS operations contain error-prevention code which may perturb input geometries by tiny amounts in order to prevent robustness errors from occurring. These minor alterations also may produce computed results which are not fully consistent. The discrepancy between results should always be very small. But queries should not rely on exact consistency when comparing overlay results. Instead, consider using an area or distance-based tolerance in geometric comparisons.

## Chapter 10

# Topology

PostGIS Topology; (face), (node) and (edge) are the basic concepts of Topology. Topology is a set of rules that define the relationship between the geometric objects in a database. Topology is used to ensure the integrity of the data and to provide a framework for spatial analysis.

Sandro Santilli's presentation at PostGIS Day Paris 2011 conference gives a good synopsis of PostGIS Topology and where it is headed [Topology with PostGIS 2.0 slide deck](#).

Vincent Picavet provides a good synopsis and overview of what is Topology, how is it used, and various FOSS4G tools that support it in [PostGIS Topology PGConf EU 2012](#).

PostGIS Topology; GIS Census Topologically Integrated Geographic Encoding and Referencing System (TIGER) and Topology Load Tiger are the basic concepts of Topology. Topology is used to ensure the integrity of the data and to provide a framework for spatial analysis.

PostGIS Topology; PostGIS Topology; SQL-MM and Topology are the basic concepts of Topology. Topology is used to ensure the integrity of the data and to provide a framework for spatial analysis.

PostGIS Topology Wiki and Topology are the basic concepts of Topology. Topology is used to ensure the integrity of the data and to provide a framework for spatial analysis.

Topology and Topology are the basic concepts of Topology. Topology is used to ensure the integrity of the data and to provide a framework for spatial analysis.

SQL/MM Topology and Topology are the basic concepts of Topology. Topology is used to ensure the integrity of the data and to provide a framework for spatial analysis.

Topology support is build by default starting with PostGIS 2.0, and can be disabled specifying `--without-topology` configure option at build time as described in [Chapter 2](#)

## 10.1 `getfaceedges_returntype`

### 10.1.1 `getfaceedges_returntype`

`getfaceedges_returntype` — A composite type that consists of a sequence number and an edge number.

#### `getfaceedges_returntype`

A composite type that consists of a sequence number and an edge number. This is the return type for `ST_GetFaceEdges` and `GetNodeEdges` functions.

- `sequence`: SRID, topology, topology\_id, topology.layer\_id, topology.layer\_id, topology\_id, layer\_id, id, type
- `edge`: topology\_id, layer\_id, id, type

### 10.1.2 `TopoGeometry`

`TopoGeometry` — A composite type representing a topologically defined geometry.

#### `TopoGeometry`

`TopoGeometry` consists of the following fields:

- `topology_id`: SRID, topology, topology\_id, topology.layer\_id, topology\_id, layer\_id, id, type
- `layer_id`: TopoGeometry, topology\_id, layer\_id, topology\_id, layer\_id, id, type
- `id`: TopoGeometry, topology\_id, layer\_id, topology\_id, layer\_id, id, type
- `type`: 1: TopoGeometry, 2: TopoFace, 3: TopoEdge, 4: TopoNode

#### `TopoGeometry`

`TopoGeometry` consists of the following fields:

|                          |                          |
|--------------------------|--------------------------|
| <code>topology_id</code> | <code>topology_id</code> |
| <code>layer_id</code>    | <code>layer_id</code>    |

**error**;

**CreateTopoGeom**

### 10.1.3 validate\_topology\_return\_type

`validate_topology_return_type` — A composite type that consists of an error message and `id1` and `id2` to denote location of error. This is the return type for `ValidateTopology`.

**error**;

`error`; `id1`; `id2`; `varchar`(`varchar`); `coincident nodes`; `edge crosses node`; `edge not simple`; `edge end node geometry mis-match`; `edge start node geometry mismatch`; `face overlaps face`; `face within face`;

- `error`; `varchar`(`varchar`); `coincident nodes`; `edge crosses node`; `edge not simple`; `edge end node geometry mis-match`; `edge start node geometry mismatch`; `face overlaps face`; `face within face`;
- `id1`; `id2`; `varchar`(`varchar`); `coincident nodes`; `edge crosses node`; `edge not simple`; `edge end node geometry mis-match`; `edge start node geometry mismatch`; `face overlaps face`; `face within face`;
- `id2`; `varchar`(`varchar`); `coincident nodes`; `edge crosses node`; `edge not simple`; `edge end node geometry mis-match`; `edge start node geometry mismatch`; `face overlaps face`; `face within face`;

**error**;

**ValidateTopology**

## 10.2 error

### 10.2.1 TopoElement

`TopoElement` — `varchar`(`varchar`); `TopoGeometry`; `id1`; `id2`; `varchar`(`varchar`); `coincident nodes`; `edge crosses node`; `edge not simple`; `edge end node geometry mis-match`; `edge start node geometry mismatch`; `face overlaps face`; `face within face`;

**TopoElement**

`TopoElement` is a class that represents a topological element in a topology. It is an abstract class and is implemented by `TopoFace`, `TopoEdge`, and `TopoNode`. The `TopoElement` class has the following attributes:

- `id`: The unique identifier of the element.
- `type`: The type of the element (face, edge, or node).
- `geom`: The geometry of the element.
- `topology`: The topology object that the element belongs to.
- `parent_id`: The ID of the parent element (if any).
- `children`: An array of child element IDs (if any).
- `is_closed`: A boolean indicating if the element is closed.
- `is_valid`: A boolean indicating if the element is valid.
- `area`: The area of the element (if it is a face).
- `length`: The length of the element (if it is an edge).
- `z`: The Z-value of the element (if it is a node).

**Note**

When creating a topology, you must specify the type of each element. The `type` attribute is used to identify the element. The `geom` attribute is used to store the geometry of the element. The `topology` attribute is used to store the topology object that the element belongs to. The `parent_id` and `children` attributes are used to store the relationships between elements. The `is_closed` attribute is used to indicate if the element is closed. The `is_valid` attribute is used to indicate if the element is valid. The `area` and `length` attributes are used to store the area and length of the element. The `z` attribute is used to store the Z-value of the element.

**TopoElementArray**

```
SELECT te[1] AS id, te[2] AS type FROM
( SELECT ARRAY[1,2]::topology.topoelement AS te ) f;
 id | type
----+-----
  1 |    2
```

```
SELECT ARRAY[1,2]::topology.topoelement;
 te
-----
 {1,2}
```

```
-- topoelement &#xc5d0; &#xc694;&#xc18c; 3&#xac1c;&#xb97c; &#xac00;&#xc9c4; ←
   &#xb30;&#xc5f4;&#xc744; &#xb123;&#xc73c;&#xba74; &#xc5b4;&#xb5bb;&#xac8c; ←
   &#xb418;&#xb294;&#xc9c0; &#xbcf4;&#xc5ec;&#xc8fc;&#xb294; &#xc608;&#xc2dc;
-- &#xc8fc;&#xc758;: topoelement &#xb294; &#xc694;&#xc18c; 2&#xac1c;&#xb97c; ←
   &#xac00;&#xc9c4; &#xb30;&#xc5f4;&#xc774;&#xc5b4;&#xc57c; &#xd558;&#xbbc0;&#xb85c; ←
   &#xcc28;&#xc6d0; &#xd655;&#xc778;&#xc5d0; &#xc2e4;&#xd328;&#xd569;&#xb2c8;&#xb2e4;.
SELECT ARRAY[1,2,3]::topology.topoelement;
ERROR: value for domain topology.topoelement violates check constraint "dimensions"
```

**TopoElementArray**

[GetTopoGeomElements](#), [TopoElementArray](#), [TopoGeometry](#), [TopoGeom\\_addElement](#), [TopoGeom\\_remElement](#)

**10.2.2 TopoElementArray**

`TopoElementArray` — An array of `TopoElement` objects.

**TopoElement**

`TopoElement` is a class that represents a topological element in a topology. It is an abstract class and is implemented by `TopoFace`, `TopoEdge`, and `TopoNode`. The `TopoElement` class has the following attributes:

**Example 10.3.1**

```

SELECT '{{1,2},{4,3}}'::topology.topoelementarray As tea;
      tea
-----
{{1,2},{4,3}}

-- Example 10.3.2: TopoElementArray_Agg
SELECT ARRAY[ARRAY[1,2], ARRAY[4,3]]::topology.topoelementarray As tea;
      tea
-----
{{1,2},{4,3}}

-- Example 10.3.3: TopoElementArray_Agg with generate_series
SELECT topology.TopoElementArray_Agg(ARRAY[e,t]) As tea
FROM generate_series(1,4) As e CROSS JOIN generate_series(1,3) As t;
      tea
-----
{{1,1},{1,2},{1,3},{2,1},{2,2},{2,3},{3,1},{3,2},{3,3},{4,1},{4,2},{4,3}}

```

```

SELECT '{{1,2,4},{3,4,5}}'::topology.topoelementarray As tea;
ERROR:  value for domain topology.topoelementarray violates check constraint "dimensions"

```

**Example 10.3.4**

[TopoElement](#), [GetTopoGeomElementArray](#), [TopoElementArray\\_Agg](#)

## 10.3 TopoGeometry

### 10.3.1 AddTopoGeometryColumn

`AddTopoGeometryColumn` — Add a column of type `TopoGeometry` to a table. The column name is `column_name`, the schema name is `schema_name`, the table name is `table_name`, the feature type is `feature_type`, the topology name is `topology_name`, the layer ID is `layer_id`, and the feature type is `feature_type`.

#### Synopsis

```
integer AddTopoGeometryColumn(varchar topology_name, varchar schema_name, varchar table_name, varchar column_name, varchar feature_type);
```

```
integer AddTopoGeometryColumn(varchar topology_name, varchar schema_name, varchar table_name, varchar column_name, varchar feature_type, integer child_layer);
```

**Example 10.3.5**

```

-- Example 10.3.5: AddTopoGeometryColumn
SELECT AddTopoGeometryColumn('topology', 'public', 'table', 'column', 'LINESTRING', 1);
-- Example 10.3.6: AddTopoGeometryColumn with child_layer
SELECT AddTopoGeometryColumn('topology', 'public', 'table', 'column', 'LINESTRING', 1, 1);

```

```

    topology.layer &#xd14c;&#xc774;&#xbe14;&#xc5d0;
    &#xb808;&#xcf54;&#xb4dc;&#xb97c;
    &#xc8fc;&#xc5b4;&#xc9c4; &#xbaa8;&#xb4e0; &#xc815;&#xbcfc4;&#xc640; &#xd568;&#xae8; &#xcd94;&#xac00;&#xd560;
    &#xac83;&#xc785;&#xb2c8;&#xb2e4;.

```

```

[child_layer]&#xb97c; &#xb530;&#xb85c; &#xc124;&#xc815;&#xd558;&#xc9c0; &#xc54a;&#xc744; (&#xb610;&#xb294;
NULL&#xb85c; &#xc124;&#xc815;&#xd560;) &#xacbd;&#xc6b0;, &#xd574;&#xb2f9; &#xb808;&#xc774;&#xc5b4;&#xac00;
(&#xc6d0;&#xc2dc;&#xd615; &#xc9c0;&#xd615; &#xc694;&#xc18c;&#xb4e4;&#xb85c; &#xc774;&#xb8e8;&#xc5b4;&#xc9c4;)
&#xae30;&#xbcf8; TopoGeometry&#xb97c; &#xb2f4;&#xc744; &#xac83;&#xc785;&#xb2c8;&#xb2e4;. &#xadf8;&#xb807;&#xc9c0;
&#xc54a;&#xc744; &#xacbd;&#xc6b0;, &#xc774; &#xb808;&#xc774;&#xc5b4;&#xb294; (child_layer&#xc5d0;&#xc11c; &#xb098;
TopoGeometry&#xb4e4;&#xb85c; &#xc774;&#xb8e8;&#xc5b4;&#xc9c4;) &#xacc4;&#xce35; TopoGeometry&#xb97c; &#xb2f4;&#xc744;
&#xac83;&#xc785;&#xb2c8;&#xb2e4;.

```

```

&#xb808;&#xc774;&#xc5b4;&#xb97c; &#xc0dd;&#xc131;&#xd588;&#xb2e4;&#xba74; (AddTopoGeometryColumn&#xd568;&#xc774;
&#xb808;&#xc774;&#xc5b4;&#xc758; ID&#xb97c; &#xbc18;&#xd658;&#xd569;&#xb2c8;&#xb2e4;) &#xd574;&#xb2f9; &#xb808;
TopoGeometry&#xac1d;&#xccb4;&#xb97c; &#xc791;&#xc131;&#xd560; &#xc900;&#xbe44;&#xac00; &#xb41c; &#xac83;&#xc785;

```

```

&#xc720;&#xd6a8;&#xd55c; feature_type &#xc740; POINT, LINE, POLYGON, COLLECTION&#xc785;&#xb2c8;&#xb2e4;.

```

Availability: 1.1

### **&#xc608;&#xc2dc;**

```

-- &#xc774; &#xc608;&#xc2dc;&#xb97c; &#xc704;&#xd574; ma_topo ↔
  &#xc2a4;&#xd0a4;&#xb9c8;&#xc5d0; &#xc0c8; &#xd14c;&#xc774;&#xbe14;&#xc744; ↔
  &#xc0dd;&#xc131;&#xd588;&#xb2e4;&#xb294; &#xc810;&#xc5d0; ↔
  &#xc8fc;&#xc758;&#xd558;&#xc2ed;&#xc2dc;&#xc624;.
-- &#xb2e4;&#xb978; &#xc2a4;&#xd0a4;&#xb9c8;&#xc5d0; &#xc0dd;&#xc131;&#xd560; &#xc218; ↔
  &#xc788;&#xc5c8;&#xb294;&#xb370;&#xb3c4; &#xb9d0;&#xc785;&#xb2c8;&#xb2e4;. ↔
  &#xadf8;&#xb7ac;&#xb354;&#xb77c;&#xba74; topology_name&#xacfc; schema_name&#xc774; ↔
  &#xb2ec;&#xb77c;&#xc84c;&#xc744; &#xc81;&#xb2c8;&#xb2e4;.
CREATE SCHEMA ma;
CREATE TABLE ma.parcels(gid serial, parcel_id varchar(20) PRIMARY KEY, address text);
SELECT topology.AddTopoGeometryColumn('ma_topo', 'ma', 'parcels', 'topo', 'POLYGON');

```

```

CREATE SCHEMA ri;
CREATE TABLE ri.roads(gid serial PRIMARY KEY, road_name text);
SELECT topology.AddTopoGeometryColumn('ri_topo', 'ri', 'roads', 'topo', 'LINE');

```

### **&#xcc38;&#xace0;**

**DropTopoGeometryColumn, toTopoGeom, CreateTopology, CreateTopoGeom**

## 10.3.2 DropTopology

DropTopology — &#xc774;&#xc6a9;&#xc5d0; &#xc8fc;&#xc758;&#xd558;&#xc2ed;&#xc2dc;&#xc624;: &#xc9c0;&#xd615; &#xc2a4;&#xd0a4;&#xb9c8;&#xb97c; &#xc0ad;&#xc81c;&#xd558;&#xace0; topology.topology &#xd14c;&#xc774;&#xbe14;&#xc5d0; &#xd574;&#xb2f9; &#xcc38;&#xc870;&#xb97c;, &#xadf8;&#xb9ac;&#xace0; geometry\_columns &#xd14c;&#xc774;&#xbe14;&#xc5d0; &#xd574;&#xb2f9; &#xc2a4;&#xd0a4;&#xb9c8;&#xc5d0; &#xc788;&#xb294; &#xd14c;&#xc774;&#xbe14;&#xc758; &#xcc38;&#xc0ad;&#xc81c;&#xd569;&#xb2c8;&#xb2e4;.

### Synopsis

integer **DropTopology**(varchar topology\_schema\_name);

**DropTopology**

```
DropTopology(schema_name, table_name, column_name);
DropTopology(schema_name, table_name);
DropTopology(schema_name);
```

Availability: 1.1

**DropTopoGeometryColumn**

```
DropTopoGeometryColumn(schema_name, table_name, column_name);
DropTopoGeometryColumn(schema_name, table_name);
```

```
SELECT topology.DropTopology('ma_topo');
```

**DropTopoGeometryColumn****DropTopoGeometryColumn****10.3.3 DropTopoGeometryColumn**

**DropTopoGeometryColumn** — `schema_name`, `table_name`, `column_name`; `DropTopoGeometryColumn` removes the geometry column from the table. The column must be of type `geometry` or `geometry_z`. The column must be a primary key or a foreign key.

**Synopsis**

```
text DropTopoGeometryColumn(varchar schema_name, varchar table_name, varchar column_name);
```

**DropTopoGeometryColumn**

```
DropTopoGeometryColumn(schema_name, table_name, column_name);
DropTopoGeometryColumn(schema_name, table_name);
DropTopoGeometryColumn(schema_name);
```

Availability: 1.1

**DropTopoGeometryColumn**

```
SELECT topology.DropTopoGeometryColumn('ma_topo', 'parcel_topo', 'topo');
```

**;**

[AddTopoGeometryColumn](#)

### 10.3.4 Populate\_Topology\_Layer

`Populate_Topology_Layer` — Adds missing entries to `topology.layer` table by reading metadata from topo tables.

#### Synopsis

setof record `Populate_Topology_Layer()`;

**;**

Adds missing entries to the `topology.layer` table by inspecting topology constraints on tables. This function is useful for fixing up entries in topology catalog after restores of schemas with topo data.

It returns the list of entries created. Returned columns are `schema_name`, `table_name`, `feature_column`.

2.3.0 [AddTopoGeometryColumn](#)

**;**

```
SELECT CreateTopology('strk_topo');
CREATE SCHEMA strk;
CREATE TABLE strk.parcels(gid serial, parcel_id varchar(20) PRIMARY KEY, address text);
SELECT topology.AddTopoGeometryColumn('strk_topo', 'strk', 'parcels', 'topo', 'POLYGON');
-- this will return no records because this feature is already registered
SELECT *
  FROM topology.Populate_Topology_Layer();

-- let's rebuild
TRUNCATE TABLE topology.layer;

SELECT *
  FROM topology.Populate_Topology_Layer();

SELECT topology_id,layer_id, schema_name As sn, table_name As tn, feature_column As fc
FROM topology.layer;
```

```
schema_name | table_name | feature_column
-----+-----+-----
strk        | parcels    | topo
(1 row)

topology_id | layer_id | sn | tn | fc
-----+-----+-----+-----+-----
          2 |         2 | strk | parcels | topo
(1 row)
```

**;**

[AddTopoGeometryColumn](#)

### 10.3.5 TopologySummary

TopologySummary — Takes a topology name and provides summary totals of types of objects in topology.

#### Synopsis

```
text TopologySummary(varchar topology_schema_name);
```

#### Examples

Takes a topology name and provides summary totals of types of objects in topology.

```
2.0.0 >>> SELECT TopologySummary('city_data');
```

#### Output

```
SELECT topology.topologysummary('city_data');
           topologysummary
-----
Topology city_data (329), SRID 4326, precision: 0
22 nodes, 24 edges, 10 faces, 29 topogeoms in 5 layers
Layer 1, type Polygonal (3), 9 topogeoms
  Deploy: features.land_parcels.feature
Layer 2, type Puntal (1), 8 topogeoms
  Deploy: features.traffic_signs.feature
Layer 3, type Lineal (2), 8 topogeoms
  Deploy: features.city_streets.feature
Layer 4, type Polygonal (3), 3 topogeoms
  Hierarchy level 1, child layer 1
  Deploy: features.big_parcels.feature
Layer 5, type Puntal (1), 1 topogeoms
  Hierarchy level 1, child layer 2
  Deploy: features.big_signs.feature
```

#### Related Functions

[Topology\\_Load\\_Tiger](#)

### 10.3.6 ValidateTopology

ValidateTopology — Returns a set of validate\_topology\_return\_type objects detailing issues with topology.

#### Synopsis

```
setof validate_topology_return_type ValidateTopology(varchar toponame, geometry bbox);
```

#### Examples

Returns a set of `validate_topology_return_type` objects detailing issues with topology, optionally limiting the check to the area specified by the `bbox` parameter.

List of possible errors, what they mean and what the returned ids represent are displayed below:

|   | <b>id1</b>   | <b>id2</b>                              | <b>Meaning</b>   |
|---|--|---|--|
| coincident nodes  | Identifier of first node.  | Identifier of second node.              | Two nodes have the same geometry.  |
| edge crosses node(<br>(<br>&#xc00;&#xb85c;&#xc9c0;&#xb974;&#xb294;&#xacbd;&#xacc4;&#xc120;))          | Identifier of the edge.  | Identifier of the node.                 | An edge has a node in its interior. See <a href="#">ST_Relate</a> .  |
| invalid edge(<br>(<br>&#xc54a;&#xc740;&#xacbd;&#xacc4;&#xc120;))                                      | Identifier of the edge.  |   | An edge geometry is invalid. See <a href="#">ST_IsValid</a> .  |
| edge not simple(<br>(<br>&#xc54a;&#xc740;&#xacbd;&#xacc4;&#xc120;))                                   | Identifier of the edge.  |   | An edge geometry has self-intersections. See <a href="#">ST_IsSimple</a> .   |
| edge crosses edge(<br>(<br>&#xc00;&#xb85c;&#xc9c0;&#xb974;&#xb294;&#xacbd;&#xacc4;&#xc120;))          | Identifier of first edge.  | Identifier of second edge.              | Two edges have an interior intersection. See <a href="#">ST_Relate</a> .   |
| edge start node geometry mismatch(<br>(<br>&#xb178;&#xb4dc;&#xb3c4;&#xd615;&#xbd88;&#xc77c;&#xce58;)) | Identifier of the edge.  | Identifier of the indicated start node. | The geometry of the node indicated as the starting node for an edge does not match the first point of the edge geometry. See <a href="#">ST_StartPoint</a> . |
| edge end node geometry mismatch(<br>(<br>&#xb178;&#xb4dc;&#xb3c4;&#xd615;&#xbd88;&#xc77c;&#xce58;))   | Identifier of the edge.  | Identifier of the indicated end node.   | The geometry of the node indicated as the ending node for an edge does not match the last point of the edge geometry. See <a href="#">ST_EndPoint</a> .      |
| face without edges(<br>(<br>&#xc5c6;&#xb294;&#xd45c;&#xba74;))  | Identifier of the orphaned face.   |   | No edge reports an existing face on either of its sides (left_face, right_face).   |
| face has no rings(<br>(<br>&#xc5c6;&#xb294;&#xd45c;&#xba74;))   | Identifier of the partially-defined face.  |   | Edges reporting a face on their sides do not form a ring.  |
| face has wrong mbr  | Identifier of the face with wrong mbr cache.   |   | Minimum bounding rectangle of a face does not match minimum bounding box of the collection of edges reporting the face on their sides.                       |
| hole not in advertised face   | Signed identifier of an edge, identifying the ring. See <a href="#">GetRingEdges</a> . |   | A ring of edges reporting a face on its exterior is contained in different face.   |
| not-isolated node has not-containing_face   | Identifier of the ill-defined node.  |   | A node which is reported as being on the boundary of one or more edges is indicating a containing face.  |

| error                                   | id1  | id2  | Meaning   |
|---|--|--|---|
| isolated node has containing_face       | Identifier of the ill-defined node.  |  | A node which is not reported as being on the boundary of any edges is lacking the indication of a containing face.  |
| isolated node has wrong containing_face | Identifier of the misrepresented node.   |  | A node which is not reported as being on the boundary of any edges indicates a containing face which is not the actual face containing it. See <a href="#">GetFaceContainingPoint</a> . |
| invalid next_right_edge                 | Identifier of the misrepresented edge.   | Signed id of the edge which should be indicated as the next right edge.                | The edge indicated as the next edge encountered walking on the right side of an edge is wrong.  |
| invalid next_left_edge                  | Identifier of the misrepresented edge.   | Signed id of the edge which should be indicated as the next left edge.                 | The edge indicated as the next edge encountered walking on the left side of an edge is wrong.   |
| mixed face labeling in ring             | Signed identifier of an edge, identifying the ring. See <a href="#">GetRingEdges</a> . |  | Edges in a ring indicate conflicting faces on the walking side. This is also known as a "Side Location Conflict".   |
| non-closed ring                         | Signed identifier of an edge, identifying the ring. See <a href="#">GetRingEdges</a> . |  | A ring of edges formed by following next_left_edge/next_right_edge attributes starts and ends on different nodes.   |
| face has multiple shells                | Identifier of the contended face.  | Signed identifier of an edge, identifying the ring. See <a href="#">GetRingEdges</a> . | More than a one ring of edges indicate the same face on its interior.   |

1.0.0 error; id1; id2; 'edge crosses node';

2.0.0 error; id1; id2; 'edge crosses node';

2.2.0 error; id1; id2; 'edge crosses node';

Changed: 3.2.0 added optional bbox parameter, perform face labeling and edge linking checks.

**error;**

```
SELECT * FROM topology.ValidateTopology('ma_topo');
    error      | id1 | id2
-----+-----+-----
face without edges | 1 |
```

**validateTopologyRelation**

[validateTopologyRelation](#), [Topology\\_Load\\_Tiger](#)

### 10.3.7 ValidateTopologyRelation

ValidateTopologyRelation — Returns info about invalid topology relation records

#### Synopsis

setof record **ValidateTopologyRelation**(varchar toponame);

**RETURNS SETOF RECORD;**

Returns a set records giving information about invalidities in the relation table of the topology.

Availability: 3.2.0

**validateTopologyRelation**

[ValidateTopology](#)

### 10.3.8 FindTopology

FindTopology — Returns a topology record by different means.

#### Synopsis

topology **FindTopology**(TopoGeometry topogeom);  
 topology **FindTopology**(regclass layerTable, name layerColumn);  
 topology **FindTopology**(name layerSchema, name layerTable, name layerColumn);  
 topology **FindTopology**(text topoName);  
 topology **FindTopology**(int id);

**RETURNS SETOF RECORD;**

Takes a topology identifier or the identifier of a topology-related object and returns a topology.topology record.

Availability: 3.2.0

**findTopology**

```
SELECT name (findTopology('features.land_parcel', 'feature'));
 name
-----
 city_data
(1 row)
```

**findTopology**

[FindLayer](#)

### 10.3.9 FindLayer

FindLayer — Returns a topology.layer record by different means.

#### Synopsis

```
topology.layer FindLayer(TopoGeometry tg);
topology.layer FindLayer(regclass layer_table, name feature_column);
topology.layer FindLayer(name schema_name, name table_name, name feature_column);
topology.layer FindLayer(integer topology_id, integer layer_id);
```

**Parameters:**

Takes a layer identifier or the identifier of a topology-related object and returns a topology.layer record.

Availability: 3.2.0

**Example:**

```
SELECT layer_id(findLayer('features.land_parcels', 'feature'));
 layer_id
-----
         1
(1 row)
```

**See also:**

[FindTopology](#)

## 10.4 Topology Statistics Management

Adding elements to a topology triggers many database queries for finding existing edges that will be split, adding nodes and updating edges that will node with the new linework. For this reason it is useful that statistics about the data in the topology tables are up-to-date.

PostGIS Topology population and editing functions do not automatically update the statistics because a updating stats after each and every change in a topology would be overkill, so it is the caller's duty to take care of that.



#### Note

That the statistics updated by autovacuum will NOT be visible to transactions which started before autovacuum process completed, so long-running transactions will need to run ANALYZE themselves, to use updated statistics.

## 10.5 Topology Management

### 10.5.1 CreateTopology

CreateTopology — Creates a topology from a set of geometry objects. The topology is created with the following options:

## Synopsis

```
integer CreateTopology(varchar topology_schema_name);
integer CreateTopology(varchar topology_schema_name, integer srid);
integer CreateTopology(varchar topology_schema_name, integer srid, double precision prec);
integer CreateTopology(varchar topology_schema_name, integer srid, double precision prec, boolean hasz);
```

### CREATE TOPOLOGY

Creates a new schema with name `topology_name` consisting of tables (`edge_data`, `face`, `node`, `relation` and registers this new topology in the `topology.topology` table. It returns the id of the topology in the `topology` table. The `srid` is the spatial reference identified as defined in `spatial_ref_sys` table for that topology. Topologies must be uniquely named. The tolerance is measured in the units of the spatial reference system. If the tolerance (`prec`) is not specified defaults to 0.

```
SQL/MM; ST_InitTopoGeo; hasz; tolerance; topology_name; topology_schema_name; srid; prec; hasz;
```

Availability: 1.1

Enhanced: 2.0 added the signature accepting `hasZ`

### CREATE TOPOID

```
SELECT topology.CreateTopology('ma_topo', 26986, 0.5);
```

```
SELECT topology.CreateTopology('ri_topo', 3438) As topoid;
```

```
topoid
-----
2
```

```
SELECT topology.CreateTopology('ri_topo', 3438) As topoid;
```

```
topoid
```

```
-----
```

```
2
```

### ST\_COPY\_TOPOLOGY

Section [4.5, ST\\_InitTopoGeo, Topology\\_Load\\_Tiger](#)

## 10.5.2 CopyTopology

`CopyTopology` — `varchar existing_topology_name`, `varchar new_name`;

## Synopsis

```
integer CopyTopology(varchar existing_topology_name, varchar new_name);
```

**Notes**

existing\_topology\_name, SRID, new\_topology\_name, existing\_topology\_name, Topo-Geometry

**Note**

topology.layer, feature\_column, TopoGeometry, schema\_name, table\_name

2.0.0

**Example**

```
SELECT topology.CopyTopology('ma_topo', 'ma_topo_bakup');
```

**Section 4.5, CreateTopology****10.5.3 ST\_InitTopoGeo**

ST\_InitTopoGeo — topology.topology, ID

**Synopsis**

```
text ST_InitTopoGeo(varchar topology_schema_name);
```

**Notes**

CreateTopology, SQL-MM, ID

Availability: 1.1



This method implements the SQL/MM specification. SQL-MM 3 Topo-Geo and Topo-Net 3: Routine Details: X.3.17

**CREATE TOPOLOGY**

```
SELECT topology.ST_InitTopoGeo('topo_schema_to_create') AS topocreation;
          astopocreation
```

```
-----
Topology-Geometry 'topo_schema_to_create' (id:7) created.
```

**CREATE TOPOLOGY****CreateTopology****10.5.4 ST\_CreateTopoGeo**

**ST\_CreateTopoGeo** — Create a topology from a collection of geometries. The topology is created in the same schema as the collection. The topology is created with the name `topology_name` and the geometry column is named `geom`. The topology is created with the name `topology_name` and the geometry column is named `geom`. The topology is created with the name `topology_name` and the geometry column is named `geom`.

**Synopsis**

```
text ST_CreateTopoGeo(varchar atopology, geometry acollection);
```

**DESCRIPTION**

**ST\_CreateTopoGeo** creates a topology from a collection of geometries. The topology is created in the same schema as the collection. The topology is created with the name `topology_name` and the geometry column is named `geom`. The topology is created with the name `topology_name` and the geometry column is named `geom`. The topology is created with the name `topology_name` and the geometry column is named `geom`.

**ST\_CreateTopoGeo** creates a topology from a collection of geometries. The topology is created in the same schema as the collection. The topology is created with the name `topology_name` and the geometry column is named `geom`. The topology is created with the name `topology_name` and the geometry column is named `geom`. The topology is created with the name `topology_name` and the geometry column is named `geom`.

**ST\_CreateTopoGeo** creates a topology from a collection of geometries. The topology is created in the same schema as the collection. The topology is created with the name `topology_name` and the geometry column is named `geom`. The topology is created with the name `topology_name` and the geometry column is named `geom`. The topology is created with the name `topology_name` and the geometry column is named `geom`.



This method implements the SQL/MM specification. SQL-MM: Topo-Geo and Topo-Net 3: Routine Details -- X.3.18

**CREATE TOPOLOGY**

```
-- Create topology
SELECT topology.ST_CreateTopoGeo('ri_topo',
  ST_GeomFromText('MULTILINESTRING((384744 236928,384750 236923,384769 236911,384799
    236895,384811 236890,384833 236884,
    384844 236882,384866 236881,384879 236883,384954 236898,385087 236932,385117 236938,
    385167 236938,385203 236941,385224 236946,385233 236950,385241 236956,385254 236971,
    385260 236979,385268 236999,385273 237018,385273 237037,385271 237047,385267 237057,
    385225 237125,385210 237144,385192 237161,385167 237192,385162 237202,385159 237214,
    385159 237227,385162 237241,385166 237256,385196 237324,385209 237345,385234 237375,
    385237 237383,385238 237399,385236 237407,385227 237419,385213 237430,385193 237439,
    385174 237451,385170 237455,385169 237460,385171 237475,385181 237503,385190 237521,
    385200 237533,385206 237538,385213 237541,385221 237542,385235 237540,385242 237541,
    385249 237544,385260 237555,385270 237570,385289 237584,385292 237589,385291
    237596,385284 237630))',3438)
);

-- Populate topology
SELECT topology.st_createtopogeo
-----
Topology ri_topo populated
```

```
-- <
CREATE TABLE ri.roads(gid serial PRIMARY KEY, road_name text);

SELECT topology.AddTopoGeometryColumn('ri_topo', 'ri', 'roads', 'topo', 'LINE');
```

**Notes:**

[AddTopoGeometryColumn](#), [CreateTopology](#), [DropTopology](#)

## 10.5.5 TopoGeo\_AddPoint

**TopoGeo\_AddPoint** — Adds a point to an existing topology and returns its identifier. The given point will snap to existing nodes or edges within given tolerance. An existing edge may be split by the snapped point.

### Synopsis

```
integer TopoGeo_AddPoint(varchar atopology, geometry apoint, float8 tolerance);
```

**Notes:**

Adds a point to an existing topology and returns its identifier. The given point will snap to existing nodes or edges within given tolerance. An existing edge may be split by the snapped point.

2.0.0 < .

**Notes:**

[TopoGeo\\_AddLineString](#), [TopoGeo\\_AddPolygon](#), [AddNode](#), [CreateTopology](#)

## 10.5.6 TopoGeo\_AddLineString

**TopoGeo\_AddLineString** — Adds a linestring to an existing topology using a tolerance and possibly splitting existing edges/faces. Returns edge identifiers.

### Synopsis

```
SETOF integer TopoGeo_AddLineString(varchar atopology, geometry aline, float8 tolerance);
```

**Notes:**

Adds a linestring to an existing topology and returns a set of edge identifiers forming it up. The given line will snap to existing nodes or edges within given tolerance. Existing edges and faces may be split by the line.



#### Note

Updating statistics about topologies being loaded via this function is up to caller, see [maintaining statistics during topology editing and population](#).

2.0.0 < .

---

**TopoGeo\_AddPolygon**

[TopoGeo\\_AddPoint](#), [TopoGeo\\_AddPolygon](#), [AddEdge](#), [CreateTopology](#)

## 10.5.7 TopoGeo\_AddPolygon

`TopoGeo_AddPolygon` — Adds a polygon to an existing topology using a tolerance and possibly splitting existing edges/faces. Returns face identifiers.

### Synopsis

SETOF integer `TopoGeo_AddPolygon`(varchar atopology, geometry apoly, float8 tolerance);

**Details**

Adds a polygon to an existing topology and returns a set of face identifiers forming it up. The boundary of the given polygon will snap to existing nodes or edges within given tolerance. Existing edges and faces may be split by the boundary of the new polygon.



#### Note

Updating statistics about topologies being loaded via this function is up to caller, see [maintaining statistics during topology editing and population](#).

---

2.0.0 &#xc84;&#xc804;&#xbd80;&#xd130; &#xc0ac;&#xc6a9;&#xd560; &#xc218; &#xc788;&#xc2b5;&#xb2c8;&#xb2e4;.

**TopoGeo\_AddPolygon**

[TopoGeo\\_AddPoint](#), [TopoGeo\\_AddLineString](#), [AddFace](#), [CreateTopology](#)

## 10.6 &#xc9c0;&#xd615; &#xd3b8;&#xc9d1;&#xc790;

### 10.6.1 ST\_AddIsoNode

`ST_AddIsoNode` — &#xc9c0;&#xd615; &#xc548;&#xc758; &#xd45c;&#xba74;&#xc5d0; &#xace0;&#xb9bd;&#xb41c;(isolated) &#xb178;&#xb4dc;&#xb97c; &#xcd94;&#xac00;&#xd558;&#xace0; &#xc0c8; &#xb178;&#xb4dc;&#xc758; ID&#xb97c; &#xb18; &#xd45c;&#xba74;&#xc774; NULL&#xc77c; &#xacbd;&#xc6b0;, &#xadf8;&#xb798;&#xb3c4; &#xb178;&#xb4dc;&#xb97c; &#xc0dd;&#xc131;&#xd569;&#xb2c8;&#xb2e4;.

### Synopsis

integer `ST_AddIsoNode`(varchar atopology, integer aface, geometry apoint);

---

**&#xc124;&#xba85;**

atopology &#xc9c0;&#xd615;&#xc758; aface ID(faceid)&#xb97c; &#xac00;&#xc9c4; &#xae30;&#xc874; &#xd45c;&#xba74;&#x  
 &#xd3ec;&#xc778;&#xd2b8; &#xc704;&#xce58;&#xac00; apoint &#xc778; &#xace0;&#xb9bd;&#xb41c; &#xb178;&#xb4dc;&#x  
 &#xcd94;&#xac00;&#xd558;&#xace0; &#xc0c8; &#xb178;&#xb4dc;&#xc758; ID(nodeid)&#xb97c; &#xbc18;&#xd658;&#xd569;&  
 &#xd3ec;&#xc778;&#xd2b8; &#xb3c4;&#xd615;&#xacfc; &#xc9c0;&#xd615;&#xc758; &#xaf5;&#xac04; &#xcc38;&#xc870;  
 &#xc2dc;&#xc2a4;&#xd15c;(SRID)&#xc774; &#xb2e4;&#xb97c; &#xacbd;&#xc6b0;, apoint &#xac00; &#xd3ec;&#xc778;&#xd  
 &#xb3c4;&#xd615;&#xc774; &#xc544;&#xb2d0; &#xacbd;&#xc6b0;, &#xd3ec;&#xc778;&#xd2b8;&#xac00; NULL&#xc77c;  
 &#xacbd;&#xc6b0;, &#xb610;&#xb294; &#xd3ec;&#xc778;&#xd2b8;&#xac00; &#xae30;&#xc874; &#xacbd;&#xacc4;&#xc120;&#  
 (&#xb610;&#xb294; &#xc678;&#xafdc;&#xc120;&#xc77c;&#xc9c0;&#xb77c;&#xb3c4;) &#xad50;&#xcc28;&#xd558;&#xb294;  
 &#xacbd;&#xc6b0; &#xc608;&#xc678;&#xac00; &#xbc1c;&#xc0dd;&#xd569;&#xb2c8;&#xb2e4;. &#xd574;&#xb2f9; &#xd3ec;&#  
 &#xc704;&#xce58;&#xc5d0; &#xc774;&#xbbf8; &#xb178;&#xb4dc;&#xac00; &#xc874;&#xc7ac;&#xd560; &#xacbd;&#xc6b0;&#  
 &#xc608;&#xc678;&#xac00; &#xbc1c;&#xc0dd;&#xd569;&#xb2c8;&#xb2e4;.

aface &#xac00; NULL&#xc774; &#xc544;&#xb2cc;&#xb370; apoint &#xac00; &#xd45c;&#xba74; &#xb0b4;&#xbd80;&#xc5d  
 &#xc788;&#xc9c0; &#xc54a;&#xc744; &#xacbd;&#xc6b0;, &#xc608;&#xc678;&#xac00; &#xbc1c;&#xc0dd;&#xd569;&#xb2c8;&#x

Availability: 1.1



This method implements the SQL/MM specification. SQL-MM: Topo-Net Routines: X+1.3.1

**&#xc608;&#xc2dc;****&#xcc38;&#xace0;**

[AddNode](#), [CreateTopology](#), [DropTopology](#), [ST\\_Intersects](#)

## 10.6.2 ST\_AddIsoEdge

ST\_AddIsoEdge — &#xc9c0;&#xd615;&#xc5d0; &#xb450; &#xac1c;&#xc758; &#xace0;&#xb9bd;&#xb41c; &#xae30;&#xc874;  
 &#xb178;&#xb4dc; anode &#xc640; anothernode &#xb97c; &#xc5f0;&#xacb0;&#xd558;&#xb294; alinestring &#xb3c4;&  
 &#xc815;&#xc758;&#xd558;&#xb294; &#xace0;&#xb9bd;&#xb41c; &#xacbd;&#xacc4;&#xc120;&#xc744; &#xcd94;&#xac00;&#x  
 &#xc0c8; &#xacbd;&#xacc4;&#xc120;&#xc758; ID(edgeid)&#xb97c; &#xbc18;&#xd658;&#xd569;&#xb2c8;&#xb2e4;.

### Synopsis

integer **ST\_AddIsoEdge**(varchar atopology, integer anode, integer anothernode, geometry alinestring);

**&#xc124;&#xba85;**

&#xc9c0;&#xd615;&#xc5d0; &#xb450; &#xac1c;&#xc758; &#xace0;&#xb9bd;&#xb41c; &#xae30;&#xc874; &#xb178;&#xb4dc;  
 anode &#xc640; anothernode &#xb97c; &#xc5f0;&#xacb0;&#xd558;&#xb294; alinestring &#xb3c4;&#xd615;&#xc774;  
 &#xc815;&#xc758;&#xd558;&#xb294; &#xace0;&#xb9bd;&#xb41c; &#xacbd;&#xacc4;&#xc120;&#xc744; &#xcd94;&#xac00;&#x  
 &#xc0c8; &#xacbd;&#xacc4;&#xc120;&#xc758; ID(edgeid)&#xb97c; &#xbc18;&#xd658;&#xd569;&#xb2c8;&#xb2e4;.

alinestring &#xb3c4;&#xd615;&#xacfc; &#xc9c0;&#xd615;&#xc758; &#xaf5;&#xac04; &#xcc38;&#xc870; &#xc2dc;&#xc2a4;  
 &#xb2e4;&#xb97c; &#xacbd;&#xc6b0;, &#xc785;&#xb825; &#xc778;&#xc218; &#xac00;&#xc6b4;&#xb370; &#xd558;&#xb098;&  
 NULL&#xc77c; &#xacbd;&#xc6b0;, &#xd558;&#xb098; &#xc774;&#xc0c1;&#xc758; &#xd45c;&#xba74;&#xc774; &#xb178;&#xb  
 &#xb2f4;&#xace0; &#xc788;&#xc744; &#xacbd;&#xc6b0;, &#xb610;&#xb294; &#xb178;&#xb4dc;&#xac00; &#xae30;&#xc874;  
 &#xacbd;&#xacc4;&#xc120;&#xc758; &#xc2dc;&#xc791; &#xb610;&#xb294; &#xc885;&#xb2e8; &#xb178;&#xb4dc;&#xc77c;  
 &#xacbd;&#xc6b0;, &#xc608;&#xc678;&#xac00; &#xbc1c;&#xc0dd;&#xd569;&#xb2c8;&#xb2e4;.

alinestring &#xc774; anode &#xc640; anothernode &#xac00; &#xc18d;&#xd574; &#xc788;&#xb294; &#xd45c;&#xba74;  
 &#xb0b4;&#xbd80;&#xc5d0; &#xc5c6;&#xc744; &#xacbd;&#xc6b0;, &#xc608;&#xc678;&#xac00; &#xbc1c;&#xc0dd;&#xd569;&#x

anode, anothernode, alinestring, &#x2dc;, &#x791;, &#x810;, &#xbc0f;, &#x885;, &#xb2e8;, &#xc544;, &#xb2d0;, &#xacbd;, &#xc6b0;, &#xc608;, &#xc678;, &#xac00;, &#xbc1c;, &#xc0dd;, &#xd569;, &#xb2c8;, &#xb2e4;.

Availability: 1.1



This method implements the SQL/MM specification. SQL-MM: Topo-Geo and Topo-Net 3: Routine Details: X.3.4

&#xc608;&#xc2dc;

&#xcc38;&#xace0;

[ST\\_AddIsoNode](#), [ST\\_IsSimple](#), [ST\\_Within](#)

### 10.6.3 ST\_AddEdgeNewFaces

ST\_AddEdgeNewFaces — &#xc0c8;, &#xacbd;, &#xacc4;, &#xc120;, &#xc744;, &#xcd94;, &#xac00;, &#xd558;, &#xace0;, &#xadf8;, &#xacfc;, &#xc815;, &#xc5d0;, &#xc11c;, &#xacbd;, &#xacc4;, &#xc120;, &#xc774;, &#xd45c;, &#xba74;, &#xc744;, &#xbd84;, &#xd560;, &#xacbd;, &#xc6b0;, &#xc6d0;, &#xbf8;, &#xd45c;, &#xba74;, &#xc744;, &#xc0ad;, &#xc81c;, &#xd558;, &#xace0;, &#xc0c8;, &#xd45c;, &#x2&#xac1c;, &#xb85c;, &#xad50;, &#xccb4;, &#xd569;, &#xb2c8;, &#xb2e4;.

#### Synopsis

integer ST\_AddEdgeNewFaces(varchar atopology, integer anode, integer anothernode, geometry acurve);

&#xc124;, &#xba85;

&#xc0c8;, &#xacbd;, &#xacc4;, &#xc120;, &#xc744;, &#xcd94;, &#xac00;, &#xd558;, &#xace0;, &#xadf8;, &#xacfc;, &#xc815;, &#xc5d0;, &#xacbd;, &#xacc4;, &#xc120;, &#xc774;, &#xd45c;, &#xba74;, &#xc744;, &#xbd84;, &#xd560;, &#xd560;, &#xacbd;, &#xc6b0;, &#xc6d0;, &#xd45c;, &#xba74;, &#xc744;, &#xc0ad;, &#xc81c;, &#xd558;, &#xace0;, &#xc0c8;, &#xd45c;, &#xba74;, 2&#xac1c;, &#xb85c;, &#xad50;, &#xc0c8;, &#xb85c;, &#xcd94;, &#xac00;, &#xb41c;, &#xacbd;, &#xacc4;, &#xc120;, &#xc758;, ID&#xb97c;, &#xbc18;, &#xd658;, &#xd569;, &

&#xbaa8;, &#xb4e0;, &#xae30;, &#xc874;, &#xc758;, &#xacb0;, &#xd569;, &#xb41c;, &#xacbd;, &#xacc4;, &#xc120;, &#xbc0f;, &#xad00;, &#xacb0;, &#xacfc;, &#xc5d0;, &#xb9de;, &#xcdb0;, &#xc5c5;, &#xb370;, &#xc774;, &#xd2b8;, &#xd569;, &#xb2c8;, &#xb2e4;.

&#xc5b4;, &#xb5a4;, &#xc778;, &#xc218;, &#xac00;, NULL&#xc77c;, &#xacbd;, &#xc6b0;, &#xc8fc;, &#xc5b4;, &#xc9c4;, &#xb178;, &#xb&#xc54c;, &#xb824;, &#xc9c0;, &#xc9c0;, &#xc54a;, &#xc740;, &#xacbd;, &#xc6b0;, (&#xb178;, &#xb4dc;, &#xac00;, &#xc774;, &#xbbf8;, &#xc9c0;, &#xd615;, &#xc2a4;, &#xd0a4;, &#xb9c8;, &#xc758;, node &#xd14c;, &#xc774;, &#xbe14;, &#xc5d0;, &#xc874;, &#xc7ac;, &#xd5&#xd569;, &#xb2c8;, &#xb2e4;), acurve &#xac00;, LINESTRING &#xc774;, &#xc544;, &#xb2d0;, &#xacbd;, &#xc6b0;, anode &#xc640;, anothernode &#xac00;, acurve &#xc758;, &#xc2dc;, &#xc791;, &#xc810;, &#xbc0f;, &#xc885;, &#xb2e8;, &#xc810;, &#xc7&#xc544;, &#xb2d0;, &#xacbd;, &#xc6b0;, &#xc624;, &#xb958;, &#xac00;, &#xbc1c;, &#xc0dd;, &#xd569;, &#xb2c8;, &#xb2e4;.

acurve &#xb3c4;, &#xd615;, &#xacfc;, &#xc9c0;, &#xd615;, &#xc758;, &#xacf5;, &#xac04;, &#xcc38;, &#xc870;, &#xc2dc;, &#xc2a4;, &#xc&#xb2e4;, &#xb97c;, &#xacbd;, &#xc6b0;, &#xc608;, &#xc678;, &#xac00;, &#xbc1c;, &#xc0dd;, &#xd569;, &#xb2c8;, &#xb2e4;.

2.0 &#xbc84;, &#xc804;, &#xbd80;, &#xd130;, &#xc0ac;, &#xc6a9;, &#xd560;, &#xc218;, &#xc788;, &#xc2b5;, &#xb2c8;, &#xb2e4;.



This method implements the SQL/MM specification. SQL-MM: Topo-Geo and Topo-Net 3: Routine Details: X.3.12

&#xc608;&#xc2dc;

&#xcc38;&#xace0;

[ST\\_RemEdgeNewFace](#)

[ST\\_AddEdgeModFace](#)

### 10.6.4 ST\_AddEdgeModFace

ST\_AddEdgeModFace — `geometry`, `geometry`, `integer`, `integer`, `integer`, `geometry`

#### Synopsis

integer ST\_AddEdgeModFace(varchar atopolgy, integer anode, integer anothernode, geometry acurve);

#### Notes

ST\_AddEdgeModFace(geom, anode, anothernode, acurve) returns a geometry of the same type as the input geometry, with the specified edge added to the face.

#### Note



ST\_AddEdgeModFace(geom, anode, anothernode, acurve) returns a geometry of the same type as the input geometry, with the specified edge added to the face. The function will fail if the edge is not a valid edge of the face.

ST\_AddEdgeModFace(geom, anode, anothernode, acurve) returns a geometry of the same type as the input geometry, with the specified edge added to the face. The function will fail if the edge is not a valid edge of the face.

ST\_AddEdgeModFace(geom, anode, anothernode, acurve) returns a geometry of the same type as the input geometry, with the specified edge added to the face. The function will fail if the edge is not a valid edge of the face.

ST\_AddEdgeModFace(geom, anode, anothernode, acurve) returns a geometry of the same type as the input geometry, with the specified edge added to the face. The function will fail if the edge is not a valid edge of the face.

2.0 ST\_AddEdgeModFace(geom, anode, anothernode, acurve) returns a geometry of the same type as the input geometry, with the specified edge added to the face. The function will fail if the edge is not a valid edge of the face.



This method implements the SQL/MM specification. SQL-MM: Topo-Geo and Topo-Net 3: Routine Details: X.3.13

#### Notes

#### Notes

#### ST\_RemEdgeModFace

#### ST\_AddEdgeNewFaces

### 10.6.5 ST\_RemEdgeNewFace

ST\_RemEdgeNewFace — `geometry`, `integer`, `integer`, `geometry`

## Synopsis

integer **ST\_RemEdgeNewFace**(varchar atopolgy, integer anedge);

### Parameters

**atopolgy**: The topology to be modified.  
**anedge**: The edge to be removed.

**Return Value**: Returns the integer ID of the new face created by removing the edge. If the edge is not found, returns NULL.

Refuses to remove an edge participating in the definition of an existing TopoGeometry. Refuses to heal two faces if any TopoGeometry is defined by only one of them (and not the other).

**Notes**: This function is implemented in the SQL/MM specification. SQL-MM: Topo-Geo and Topo-Net 3: Routine Details: X.3.14

2.0



This method implements the SQL/MM specification. SQL-MM: Topo-Geo and Topo-Net 3: Routine Details: X.3.14

### See Also

**ST\_RemEdgeModFace**

**ST\_AddEdgeNewFaces**

## 10.6.6 ST\_RemEdgeModFace

**ST\_RemEdgeModFace** — Removes an edge from a topology and modifies the faces it was part of. Returns the integer ID of the new face created by removing the edge. If the edge is not found, returns NULL.

## Synopsis

integer **ST\_RemEdgeModFace**(varchar atopolgy, integer anedge);

### Parameters

**atopolgy**: The topology to be modified.  
**anedge**: The edge to be removed.

**Return Value**: Returns the integer ID of the new face created by removing the edge. If the edge is not found, returns NULL.



`acurve` &#xb3c4;&#xd615;&#xacfc; &#xc9c0;&#xd615;&#xc758; &#xacf5;&#xac04; &#xcc38;&#xc870; &#xc2dc;&#xc2a4;&#xc  
&#xb2e4;&#xb97c; &#xacbd;&#xc6b0; &#xc608;&#xc678;&#xac00; &#xbc1c;&#xc0dd;&#xd569;&#xb2c8;&#xb2e4;.

&#xc0c8; `acurve` &#xac00; &#xb2e8;&#xc21c; &#xb3c4;&#xd615;&#xc774; &#xc544;&#xb2d0; &#xacbd;&#xc6b0;, &#xc624;&#xc  
&#xbc1c;&#xc0dd;&#xd569;&#xb2c8;&#xb2e4;.

&#xc608;&#xc804; &#xc704;&#xce58;&#xc5d0;&#xc11c; &#xc0c8; &#xc704;&#xce58;&#xb85c; &#xacbd;&#xacc4;&#xc120;&#xc  
&#xc774;&#xb3d9;&#xc2dc;&#xd0ac; &#xb54c; &#xc7a5;&#xc560;&#xbb3c;&#xc5d0; &#xbd80;&#xb52a;&#xd788;&#xb294;  
&#xacbd;&#xc6b0; &#xc624;&#xb958;&#xac00; &#xbc1c;&#xc0dd;&#xd569;&#xb2c8;&#xb2e4;.

1.1.0 &#xbc84;&#xc804;&#xbd80;&#xd130; &#xc0ac;&#xc6a9;&#xd560; &#xc218; &#xc788;&#xc2b5;&#xb2c8;&#xb2e4;.

&#xac1c;&#xc120; &#xc0ac;&#xd56d;: 2.0.0&#bc84;&#xc804;&#xbd80;&#xd130; &#xc9c0;&#xd615;&#xc758; &#xc77c;&#xad  
&#xac15;&#xc81c;&#xd569;&#xb2c8;&#xb2e4;.



This method implements the SQL/MM specification. SQL-MM: Topo-Geo and Topo-Net 3: Routine Details X.3.6

### &#xc608;&#xc2dc;

```

SELECT topology.ST_ChangeEdgeGeom('ma_topo', 1,
    ST_GeomFromText('LINESTRING(227591.9 893900.4,227622.6 893844.3,227641.6
    893816.6, 227704.5 893778.5)', 26986) );
-----
Edge 1 changed

```

### &#xcc38;&#xace0;

[ST\\_AddEdgeModFace](#)

[ST\\_RemEdgeModFace](#)

[ST\\_ModEdgeSplit](#)

## 10.6.8 ST\_ModEdgeSplit

**ST\_ModEdgeSplit** — &#xae30;&#xc874; &#xacbd;&#xacc4;&#xc120;&#xc744; &#xb530;&#xb77c; &#xc0c8; &#xb178;&#xb4dc;&#xc  
&#xcd94;&#xac00;&#xd55c; &#xb2e4;&#xc74c;, &#xc6d0;&#xbcf8; &#xacbd;&#xacc4;&#xc120;&#xc744; &#xc218;&#xc815;&#xc  
&#xc0c8; &#xacbd;&#xacc4;&#xc120;&#xc744; &#xcd94;&#xac00;&#xd574;&#xc11c; &#xacbd;&#xacc4;&#xc120;&#xc744;  
&#xbd84;&#xd560;&#xd569;&#xb2c8;&#xb2e4;.

### Synopsis

integer **ST\_ModEdgeSplit**(varchar atopology, integer anedge, geometry apoint);

### &#xc124;&#xba85;

&#xae30;&#xc874; &#xacbd;&#xacc4;&#xc120;&#xc744; &#xb530;&#xb77c; &#xc0c8; &#xb178;&#xb4dc;&#xb97c; &#xcd94;&#xc  
&#xb2e4;&#xc74c;, &#xc6d0;&#xbcf8; &#xacbd;&#xacc4;&#xc120;&#xc744; &#xc218;&#xc815;&#xd558;&#xace0; &#xc0c8;  
&#xacbd;&#xacc4;&#xc120;&#xc744; &#xcd94;&#xac00;&#xd574;&#xc11c; &#xacbd;&#xacc4;&#xc120;&#xc744; &#xbd84;&#xc  
&#xae30;&#xc874;&#xc758; &#xbaa8;&#xb4e0; &#xacb0;&#xd569;&#xb41c; &#xacbd;&#xacc4;&#xc120;&#xb4e4; &#xbc0f;  
&#xad00;&#xacc4;&#xc131;&#xc744; &#xc0c8;&#xb85c;&#xc774; &#xbd84;&#xd560;&#xb41c; &#xacbd;&#xacc4;&#xc120;&#xc  
&#xb9de;&#xcdb0; &#xc5c5;&#xb370;&#xc774;&#xd2b8;&#xd569;&#xb2c8;&#xb2e4;. &#xc0c8;&#xb85c; &#xcd94;&#xac00;&#xc  
&#xb178;&#xb4dc;&#xc758; &#xc2dd;&#xbcc4;&#xc790;&#xb97c; &#xbc18;&#xd658;&#xd569;&#xb2c8;&#xb2e4;.

Availability: 1.1

&#xbcc0;&#xacbd; &#xc0ac;&#xd56d;: 2.0 &#xbbf8;&#xb9cc; &#bc84;&#xc804;&#xc5d0;&#xc11c;, &#xc774; &#xd568;&#xc21  
**ST\_ModEdgesSplit**&#xc774;&#xb77c;&#xb294; &#xc798;&#xbabb;&#xb41c; &#xba85;&#xce6d;&#xc774;&#xc5c8;&#xc2b5;&#xc



This method implements the SQL/MM specification. SQL-MM: Topo-Geo and Topo-Net 3: Routine Details: X.3.9

**Example 10.6.8**

```
-- Example 10.6.8: AddEdge and ST_ModEdgeSplit
SELECT topology.AddEdge('ma_topo', ST_GeomFromText('LINESTRING(227592 893910, 227600 893910)', 26986) ) As edgeid;

-- edgeid --
3

-- Example 10.6.9: ST_ModEdgeSplit
SELECT topology.ST_ModEdgeSplit('ma_topo', 3, ST_SetSRID(ST_Point(227594, 893910), 26986) ) As node_id;
-----
7
```

**Example 10.6.9**

[ST\\_NewEdgesSplit](#), [ST\\_ModEdgeHeal](#), [ST\\_NewEdgeHeal](#), [AddEdge](#)

**10.6.9 ST\_ModEdgeHeal**

**ST\_ModEdgeHeal** — Heals two edges by deleting the node connecting them, modifying the first edge and deleting the second edge. Returns the id of the deleted node.

**Synopsis**

```
int ST_ModEdgeHeal(varchar atopology, integer anedge, integer anotheredge);
```

**Example 10.6.9**

Heals two edges by deleting the node connecting them, modifying the first edge and deleting the second edge. Returns the id of the deleted node. Updates all existing joined edges and relationships accordingly.

2.0 [New in 2.0.0](#); [New in 2.0.0](#);



This method implements the SQL/MM specification. SQL-MM: Topo-Geo and Topo-Net 3: Routine Details: X.3.9

**Example 10.6.9**

[ST\\_ModEdgeSplit](#) [ST\\_NewEdgesSplit](#)

**10.6.10 ST\_NewEdgeHeal**

**ST\_NewEdgeHeal** — Heals two edges by deleting the node connecting them, deleting both edges, and replacing them with an edge whose direction is the same as the first edge provided.

**Synopsis**

```
int ST_NewEdgeHeal(varchar atopology, integer anedge, integer anotheredge);
```

**ST\_HealEdges**

Heals two edges by deleting the node connecting them, deleting both edges, and replacing them with an edge whose direction is the same as the first edge provided. Returns the id of the new edge replacing the healed ones. Updates all existing joined edges and relationships accordingly.

2.0



This method implements the SQL/MM specification. SQL-MM: Topo-Geo and Topo-Net 3: Routine Details: X.3.9

**ST\_ModEdgeHeal**

[ST\\_ModEdgeHeal](#) [ST\\_ModEdgeSplit](#) [ST\\_NewEdgesSplit](#)

**10.6.11 ST\_MoveIsoNode**

**ST\_MoveIsoNode** — Moves an isolated node in a topology from one point to another. If new `apoint` geometry exists as a node an error is thrown. Returns description of move.

**Synopsis**

text **ST\_MoveIsoNode**(varchar `atopology`, integer `anode`, geometry `apoint`);

**ST\_MoveIsoNode**

`apoint` is not a point, the existing node is not isolated (is a start or end point of an existing edge), new node location intersects an existing edge (even at the end points) or the new location is in a different face (since 3.2.0) then an exception is thrown.

If any arguments are null, the `apoint` is not a point, the existing node is not isolated (is a start or end point of an existing edge), new node location intersects an existing edge (even at the end points) or the new location is in a different face (since 3.2.0) then an exception is thrown.

2.0.0

Enhanced: 3.2.0 ensures the node cannot be moved in a different face



This method implements the SQL/MM specification. SQL-MM: Topo-Net Routines: X.3.2

**ST\_MoveIsoNode**

```
-- <code>SELECT topology.ST_AddIsoNode('ma_topo', NULL, ST_GeomFromText('POINT(227579 893916)', 26986)) As nodeid;
-----
       7
-- <code>SELECT topology.ST_MoveIsoNode('ma_topo', 7, ST_GeomFromText('POINT(227579.5 893916.5)', 26986)) As descrip;
-----
Isolated Node 7 moved to location 227579.5,893916.5
```

**ST\_AddIsoNode**

**ST\_AddIsoNode**

## 10.6.12 ST\_NewEdgesSplit

**ST\_NewEdgesSplit** — Splits a set of edges into two sets based on a point. The function returns a table with two columns: `edgeid` and `newnodeid`. The `edgeid` column contains the ID of the original edge, and the `newnodeid` column contains the ID of the new node created by the split. The function is implemented in C and is available in the `postgis` extension.

### Synopsis

integer **ST\_NewEdgesSplit**(varchar topology, integer anedge, geometry apoint);

**ST\_NewEdgesSplit**

`apoint` is the point to split the edges. The function returns a table with two columns: `edgeid` and `newnodeid`. The `edgeid` column contains the ID of the original edge, and the `newnodeid` column contains the ID of the new node created by the split. The function is implemented in C and is available in the `postgis` extension.

`edgeid` is the ID of the original edge. `newnodeid` is the ID of the new node created by the split. The function is implemented in C and is available in the `postgis` extension.

Availability: 1.1



This method implements the SQL/MM specification. SQL-MM: Topo-Net Routines: X.3.8

**ST\_NewEdgesSplit**

```
-- ST_GeomFromText('LINESTRING(227575 893917,227592 893900)') As edgeid;
SELECT topology.AddEdge('ma_topo', ST_GeomFromText('LINESTRING(227575 893917,227592 893900)') As edgeid;
-- ST_GeomFromText('POINT(227578.5 893913.5)') As newnodeid;
SELECT topology.ST_NewEdgesSplit('ma_topo', 2, ST_GeomFromText('POINT(227578.5 893913.5)') As newnodeid;
newnodeid
-----
2
-- ST_GeomFromText('POINT(227578.5 893913.5)') As newnodeid;
SELECT topology.ST_NewEdgesSplit('ma_topo', 2, ST_GeomFromText('POINT(227578.5 893913.5)') As newnodeid;
newnodeid
-----
6
```

**ST\_RemoveIsoNode**

[ST\\_ModEdgeSplit](#) [ST\\_ModEdgeHeal](#) [ST\\_NewEdgeHeal](#) [AddEdge](#)

### 10.6.13 ST\_RemoveIsoNode

**ST\_RemoveIsoNode** — Removes an isolated node and returns description of action. If the node is not isolated, then an exception is thrown.

#### Synopsis

text **ST\_RemoveIsoNode**(varchar atopology, integer anode);

**ST\_RemoveIsoNode**

Removes an isolated node and returns description of action. If the node is not isolated, then an exception is thrown.

Availability: 1.1



This method implements the SQL/MM specification. SQL-MM: Topo-Geo and Topo-Net 3: Routine Details: X+1.3.3

**ST\_RemoveIsoNode**

```
-- <math>x_{d45}</math>; <math>x_{ba74}</math>; <math>x_{774}</math>; <math>x_{5c6}</math>; <math>x_{b294}</math>; <math>x_{ace0}</math>; <math>x_{b9bd}</math>; <math>x_{b178}</math>; <math>x_{b4dc}</math>; <math>x_{b97c}</math>; <math>x_{c81c}</math>; <math>x_{ac70}</math>; --
SELECT topology.ST_RemoveIsoNode('ma_topo', 7) As result;
      result
-----
Isolated node 7 removed
```

**ST\_RemoveIsoNode**

[ST\\_AddIsoNode](#)

### 10.6.14 ST\_RemoveIsoEdge

**ST\_RemoveIsoEdge** — Removes an isolated edge and returns description of action. If the edge is not isolated, then an exception is thrown.

#### Synopsis

text **ST\_RemoveIsoEdge**(varchar atopology, integer anedge);

**ST\_RemoveIsolatedNode**

Removes an isolated edge and returns description of action. If the edge is not isolated, then an exception is thrown.

Availability: 1.1



This method implements the SQL/MM specification. SQL-MM: Topo-Geo and Topo-Net 3: Routine Details: X+1.3.3

**ST\_RemoveIsolatedNode**

```
-- ST_RemoveIsolatedNode('ma_topo', 7) As result;
SELECT topology.ST_RemoveIsolatedNode('ma_topo', 7) As result;
-----
Isolated node 7 removed
```

**ST\_AddIsoNode****ST\_AddIsoNode****10.7 ST\_RemoveIsolatedNode****10.7.1 GetEdgeByPoint**

GetEdgeByPoint — Finds the edge-id of an edge that intersects a given point.

**Synopsis**

integer **GetEdgeByPoint**(varchar atopology, geometry apoint, float8 toll);

**ST\_GetEdgeById**

Retrieves the id of an edge that intersects a Point.

```
ST_GetEdgeById('ma_topo', 'POINT(100 100)', 0.001) As result;
-----
1
```

If apoint doesn't intersect an edge, returns 0 (zero).

```
ST_GetEdgeById('ma_topo', 'POINT(100 100)', 0.001) As result;
-----
0
```

**Note**

```
ST_GetEdgeById('ma_topo', 'POINT(100 100)', 0.001) As result;
-----
1
```

GEOS 3.10.0

2.0.0

**GetEdgeByPoint**

`GetEdgeByPoint` — Finds edge intersecting a given point.

```
SELECT topology.GetEdgeByPoint('ma_topo',geom, 1) As withlmtol, topology.GetEdgeByPoint('↔
ma_topo',geom,0) As withnotol
FROM ST_GeomFromEWKT('SRID=26986;POINT(227622.6 893843)') As geom;
withlmtol | withnotol
-----+-----
          2 |          0
```

```
SELECT topology.GetEdgeByPoint('ma_topo',geom, 1) As nearnode
FROM ST_GeomFromEWKT('SRID=26986;POINT(227591.9 893900.4)') As geom;

-- &#xc624;&#xb958; &#xbc1c;&#xc0dd; --
ERROR:  Two or more edges found
```

**GetFaceByPoint**

[AddEdge](#), [GetNodeByPoint](#), [GetFaceByPoint](#)

**10.7.2 GetFaceByPoint**

`GetFaceByPoint` — Finds face intersecting a given point.

**Synopsis**

integer **GetFaceByPoint**(varchar atopology, geometry apoint, float8 toll);

**GetFaceByPoint**

Finds a face referenced by a Point, with given tolerance.

The function will effectively look for a face intersecting a circle having the point as center and the tolerance as radius.

If no face intersects the given query location, 0 is returned (universal face).

If more than one face intersect the query location an exception is thrown.

2.0.0 [GetFaceByPoint](#) — Finds face intersecting a given point.

Enhanced: 3.2.0 more efficient implementation and clearer contract, stops working with invalid topologies.

**GetFaceByPoint**

```
SELECT topology.GetFaceByPoint('ma_topo',geom, 10) As withlmtol, topology.GetFaceByPoint('↔
ma_topo',geom,0) As withnotol
FROM ST_GeomFromEWKT('POINT(234604.6 899382.0)') As geom;

withlmtol | withnotol
-----+-----
          1 |          0
```

```
SELECT topology.GetFaceByPoint('ma_topo',geom, 1) As nearnode
FROM ST_GeomFromEWKT('POINT(227591.9 893900.4)') As geom;

-- &#xc624;&#xb958; &#xbc1c;&#xc0dd; --
ERROR:  Two or more faces found
```

---

**GetFaceContainingPoint**

[GetFaceContainingPoint](#), [AddFace](#), [GetNodeByPoint](#), [GetEdgeByPoint](#)

### 10.7.3 GetFaceContainingPoint

**GetFaceContainingPoint** — Finds the face containing a point.

#### Synopsis

integer **GetFaceContainingPoint**(text atopology, geometry apoint);

**Return Value**

Returns the id of the face containing a point.

An exception is thrown if the point falls on a face boundary.



#### Note

The function relies on a valid topology, using edge linking and face labeling.

---

Availability: 3.2.0

**GetFaceGeometry**

[ST\\_GetFaceGeometry](#)

### 10.7.4 GetNodeByPoint

**GetNodeByPoint** — Finds the node-id of a node at a point location.

#### Synopsis

integer **GetNodeByPoint**(varchar atopology, geometry apoint, float8 tol1);

**Return Value**

Retrieves the id of a node at a point location.

The function returns an integer (id-node) given a topology, a POINT and a tolerance. If tolerance = 0 means exact intersection, otherwise retrieves the node from an interval.

If apoint doesn't intersect a node, returns 0 (zero).

If use tolerance > 0 and there is more than one node near the point then an exception is thrown.



#### Note

**Parameters**  
 tolerance = 0 means exact intersection, otherwise retrieves the node from an interval.  
 If apoint doesn't intersect a node, returns 0 (zero).  
 If use tolerance > 0 and there is more than one node near the point then an exception is thrown.

---

GEOS

2.0.0

---

**Examples**

Find the nearest node to a point:

```
SELECT topology.GetNodeByPoint('ma_topo',geom, 1) As nearnode
FROM ST_GeomFromEWKT('SRID=26986;POINT(227591.9 893900.4)') As geom;
-----
nearnode
-----
1
```

```
SELECT topology.GetNodeByPoint('ma_topo',geom, 1000) As too_much_tolerance
FROM ST_GeomFromEWKT('SRID=26986;POINT(227591.9 893900.4)') As geom;

-- Error message:
ERROR: Two or more nodes found
```

**Related Functions**

[AddEdge](#), [GetEdgeByPoint](#), [GetFaceByPoint](#)

**10.7.5 GetTopologyID**

**GetTopologyID** — Returns the topology ID for a given topology name.

**Synopsis**

integer **GetTopologyID**(varchar toponame);

**Arguments**

**toponame** text: The name of the topology.

Availability: 1.1

**Examples**

```
SELECT topology.GetTopologyID('ma_topo') As topo_id;
-----
topo_id
-----
1
```

**Related Functions**

[CreateTopology](#), [DropTopology](#), [GetTopologyName](#), [GetTopologySRID](#)

**10.7.6 GetTopologySRID**

**GetTopologySRID** — Returns the SRID for a given topology name.

## Synopsis

integer **GetTopologyID**(varchar toponame);

### GetTopologyID

GetTopologyID — Returns the SRID of the topology. The SRID is returned as an integer. The SRID is the same as the SRID of the topology. The SRID is the same as the SRID of the topology.

2.0.0 Availability: 1.1

### GetTopologySRID

```
SELECT topology.GetTopologySRID('ma_topo') As SRID;
SRID
-----
4326
```

### GetTopologyName

[CreateTopology](#), [DropTopology](#), [GetTopologyName](#), [GetTopologyID](#)

## 10.7.7 GetTopologyName

GetTopologyName — Returns the name of the topology. The name is returned as a varchar. The name is the same as the name of the topology.

## Synopsis

varchar **GetTopologyName**(integer topology\_id);

### GetTopologyName

GetTopologyName — Returns the name of the topology. The name is returned as a varchar. The name is the same as the name of the topology.

Availability: 1.1

### GetTopologyName

```
SELECT topology.GetTopologyName(1) As topo_name;
topo_name
-----
ma_topo
```

### GetTopologyName

[CreateTopology](#), [DropTopology](#), [GetTopologyID](#), [GetTopologySRID](#)

## 10.7.8 ST\_GetFaceEdges

ST\_GetFaceEdges — aface

### Synopsis

getfaceedges\_returntype ST\_GetFaceEdges(varchar atopolgy, integer aface);

#### Parameters

aface

2.0



This method implements the SQL/MM specification. SQL-MM 3 Topo-Geo and Topo-Net 3: Routine Details: X.3.5

#### SQL Examples

```
-- topology.ST_GetFaceEdges('tt', 1)
SELECT (topology.ST_GetFaceEdges('tt', 1)).*
-- sequence | edge
-----+-----
1 | -4
2 | 5
3 | 7
4 | -6
5 | 1
6 | 2
7 | 3
(7 rows)
```

```
-- topology.ST_GetFaceEdges('tt', 1)
SELECT t.seq, t.edge, geom
FROM topology.ST_GetFaceEdges('tt', 1) As t(seq, edge)
INNER JOIN tt.edge AS e ON abs(t.edge) = e.edge_id;
```

#### Related Functions

[GetRingEdges](#), [AddFace](#), [ST\\_GetFaceGeometry](#)

## 10.7.9 ST\_GetFaceGeometry

ST\_GetFaceGeometry — Returns the geometry of a face in a topology. The face is identified by its ID.

### Synopsis

geometry ST\_GetFaceGeometry(varchar atopology, integer aface);

#### Parameters

**atopology**: The name of the topology.  
**aface**: The ID of the face to be returned.

Availability: 1.1



This method implements the SQL/MM specification. SQL-MM 3 Topo-Geo and Topo-Net 3: Routine Details: X.3.16

#### Example

```
-- AddFace (234776.9 899563.7,234896.5 899456.7,234914 899436.4,234946.6 899356.9,
--         234872.5 899328.7,234891 899285.4,234992.5 899145,234890.6 899069,
--         234755.2 899255.4,234612.7 899379.4,234776.9 899563.7)
SELECT ST_AsText(topology.ST_GetFaceGeometry('ma_topo', 1)) As facegeomwkt;
-- Returns:
--
--         facegeomwkt
-----
POLYGON((234776.9 899563.7,234896.5 899456.7,234914 899436.4,234946.6 899356.9,
234872.5 899328.7,234891 899285.4,234992.5 899145,234890.6 899069,
234755.2 899255.4,234612.7 899379.4,234776.9 899563.7))
```

#### See Also

[AddFace](#)

## 10.7.10 GetRingEdges

GetRingEdges — Returns the edges of a ring in a topology. The ring is identified by its ID. The edges are returned as a set of line segments.

### Synopsis

getfaceedges\_returntype GetRingEdges(varchar atopology, integer aring, integer max\_edges=null);

#### Parameters

**atopology**: The name of the topology.  
**aring**: The ID of the ring to be returned.  
**max\_edges**: The maximum number of edges to be returned. If null, all edges are returned.

**ST\_GetFaceEdges**

ST\_GetFaceEdges(*geom*)  
Returns the edges of the face of the geometry *geom*. The edges are returned as a set of *LINESTRING* geometries. The edges are ordered by their position in the face, starting from the first edge and proceeding in a clockwise direction.

ST\_GetFaceEdges(*geom*, *order*)  
Returns the edges of the face of the geometry *geom*, ordered by the integer *order*. The edges are returned as a set of *LINESTRING* geometries. The edges are ordered by their position in the face, starting from the first edge and proceeding in a clockwise direction.

max\_edges *integer*, NULL  
Returns the edges of the face of the geometry *geom*, limited to a maximum of *max\_edges* edges. The edges are returned as a set of *LINESTRING* geometries. The edges are ordered by their position in the face, starting from the first edge and proceeding in a clockwise direction.



**Note**

ST\_GetFaceEdges(*geom*, *order*)  
Returns the edges of the face of the geometry *geom*, ordered by the integer *order*. The edges are returned as a set of *LINESTRING* geometries. The edges are ordered by their position in the face, starting from the first edge and proceeding in a clockwise direction.

ST\_GetFaceEdges(*geom*, *order*, *max\_edges*)  
Returns the edges of the face of the geometry *geom*, ordered by the integer *order*, limited to a maximum of *max\_edges* edges. The edges are returned as a set of *LINESTRING* geometries. The edges are ordered by their position in the face, starting from the first edge and proceeding in a clockwise direction.

**ST\_GetNodeEdges**

ST\_GetNodeEdges(*geom*)  
Returns the edges of the nodes of the geometry *geom*. The edges are returned as a set of *LINESTRING* geometries. The edges are ordered by their position in the nodes, starting from the first edge and proceeding in a clockwise direction.

**10.7.11 GetNodeEdges**

GetNodeEdges(*geom*)  
Returns the edges of the nodes of the geometry *geom*. The edges are returned as a set of *LINESTRING* geometries. The edges are ordered by their position in the nodes, starting from the first edge and proceeding in a clockwise direction.

**Synopsis**

getfaceedges\_returntype **GetNodeEdges**(varchar topology, integer anode);

**ST\_GetNodeEdges**

ST\_GetNodeEdges(*geom*)  
Returns the edges of the nodes of the geometry *geom*. The edges are returned as a set of *LINESTRING* geometries. The edges are ordered by their position in the nodes, starting from the first edge and proceeding in a clockwise direction.



**Note**

The `ST_Polygonize` function returns a `Geometry` object representing the union of all faces defined by the input edges. The result is a `Geometry` object of type `POLYGON`. The function is implemented in C and is available in the `postgis` extension.

2.0 The function is available in the `postgis` extension.

**Usage**

`getfaceedges_returntype, GetRingEdges, ST_Azimuth`

## 10.8 Polygonization

### 10.8.1 Polygonize

Polygonize — Finds and registers all faces defined by topology edges.

**Synopsis**

text `Polygonize`(varchar toponame);

**Return Value**

Registers all faces that can be built out a topology edge primitives.

The function returns a `Geometry` object representing the union of all faces defined by the input edges. The result is a `Geometry` object of type `POLYGON`. The function is implemented in C and is available in the `postgis` extension.



**Note**

The `ST_Polygonize` function returns a `Geometry` object representing the union of all faces defined by the input edges. The result is a `Geometry` object of type `POLYGON`. The function is implemented in C and is available in the `postgis` extension.



**Note**

The `ST_Polygonize` function returns a `Geometry` object representing the union of all faces defined by the input edges. The result is a `Geometry` object of type `POLYGON`. The function is implemented in C and is available in the `postgis` extension.

2.0.0 The function is available in the `postgis` extension.

**Usage**

`AddFace, ST_Polygonize`

## 10.8.2 AddNode

AddNode — `integer AddNode(varchar topologyname, geometry apoint, boolean allowEdgeSplitting=false, boolean computeContainingFace=false);`

### Synopsis

integer **AddNode**(varchar topologyname, geometry apoint, boolean allowEdgeSplitting=false, boolean computeContainingFace=false);

#### **Parameters**

`topologyname` — Name of the topology to which the node is added.  
`apoint` — Geometry of the point to be added.  
`allowEdgeSplitting` — If true, the function will split edges if necessary to accommodate the new node.  
`computeContainingFace` — If true, the function will compute the face containing the point.

`computeContainingFace` — If true, the function will compute the face containing the point.

#### Note



apoint — Geometry of the point to be added.  
 allowEdgeSplitting — If true, the function will split edges if necessary to accommodate the new node.  
 computeContainingFace — If true, the function will compute the face containing the point.

2.0.0 — If true, the function will compute the face containing the point.

#### **Examples**

```
SELECT topology.AddNode('ma_topo', ST_GeomFromText('POINT(227641.6 893816.5)', 26986) ) As
  nodeid;
--
nodeid
-----
4
```

#### **Related Functions**

[AddEdge](#), [CreateTopology](#)

### 10.8.3 AddEdge

AddEdge — `ST_AddEdge` (`toponame` `geometry` `aline`);

#### Synopsis

integer **AddEdge**(varchar toponame, geometry aline);

#### `ST_AddEdge`

`ST_AddEdge` (`toponame` `geometry` `aline`);

#### Note



`aline` `geometry` `aline`; `edgeid` `integer`

#### Note



`aline` `geometry` `aline`; `srid` `integer`

GEOS `boolean`

2.0.0 `boolean`

#### `ST_AddEdge`

```
SELECT topology.AddEdge('ma_topo', ST_GeomFromText('LINESTRING(227575.8 893917.2, 227591.9 893900.4)', 26986) ) As edgeid;
-- geometry;
edgeid
-----
1
```

```

SELECT topology.AddEdge('ma_topo', ST_GeomFromText('LINESTRING(227591.9 893900.4,227622.6
893844.2,227641.6 893816.5,
227704.5 893778.5)', 26986) ) As edgeid;
--
edgeid
-----
2

SELECT topology.AddEdge('ma_topo', ST_GeomFromText('LINESTRING(227591.2 893900, 227591.9
893900.4,
227704.5 893778.5)', 26986) ) As edgeid;
--
ERROR: Edge intersects (not on endpoints) with existing edge 1

```

**TopoGeo\_AddLineString, CreateTopology, Section 4.5**

**TopoGeo\_AddLineString, CreateTopology, Section 4.5**

## 10.8.4 AddFace

**AddFace** — (face primitive)

### Synopsis

integer **AddFace**(varchar toponame, geometry apolygon, boolean force\_new=false);

**Note**

(face primitive)

left\_face, right\_face, containing\_face



### Note

edge, next\_left\_edge, next\_right\_edge

apolygon, force\_new

apolygon, force\_new

```

force_new = true;
ID = ID;

```

**Note**

force\_new = true; ID = ID;

**Note**

```

apolygon srid = srid;

```

```


```

**608;2dc;**

```

-- generate_series(1,10000) As i
SELECT topology.AddEdge('ma_topo', ST_MakeLine(ST_PointN(geom,i), ST_PointN(geom, i + 1) ))
As edgeid
FROM (SELECT ST_NPoints(geom) AS npt, geom
FROM
(SELECT ST_Boundary(ST_GeomFromText ('POLYGON((234896.5 899456.7,234914
899436.4,234946.6 899356.9,234872.5 899328.7,
234891 899285.4,234992.5 899145, 234890.6 899069,234755.2 899255.4,
234612.7 899379.4,234776.9 899563.7,234896.5 899456.7))', 26986) )
) As geoms) As facen CROSS JOIN generate_series(1,10000) As i
WHERE i < npt;
--
edgeid
-----
3
4
5
6
7
8

```

```

 9
 10
 11
 12
(10 rows)
-- &#xadf8; &#xb2e4;&#xc74c; &#xd45c;&#xba74;&#xc744; ←
  &#xcd94;&#xac00;&#xd569;&#xb2c8;&#xb2e4;. --

SELECT topology.AddFace('ma_topo',
  ST_GeomFromText('POLYGON((234896.5 899456.7,234914 899436.4,234946.6 899356.9,234872.5 ←
    899328.7,
    234891 899285.4,234992.5 899145, 234890.6 899069,234755.2 899255.4,
    234612.7 899379.4,234776.9 899563.7,234896.5 899456.7))', 26986) ) As faceid;
-- &#xacb0;&#xacfc; --
faceid
-----
 1

```

**&#xcc38;&#xace0;**

[AddEdge, CreateTopology, Section 4.5](#)

## 10.8.5 ST\_Simplify

**ST\_Simplify** — &#xb354;&#xae00;&#xb7ec;&#xc2a4;-&#xd328;&#xcee4;(Douglas-Peucker) &#xc54c;&#xace0;&#xb9ac;&#xc998;&#xd1b5;&#xd574; &#xc785;&#xb825; TopoGeometry &#xc758; "&#xb2e8;&#xc21c;&#xd654;&#xb41c;" &#xb3c4;&#xd615; &#xbc84;&#xc804;&#xc744; &#xb18;&#xd658;&#xd569;&#xb2c8;&#xb2e4;.

### Synopsis

geometry **ST\_Simplify**(TopoGeometry tg, float8 tolerance);

**&#xc124;&#xba85;**

&#xac01; &#xad6c;&#xc131; &#xc694;&#xc18c;&#xc758; &#xacbd;&#xacc4;&#xc120;&#xc5d0; &#xb354;&#xae00;&#xb7ec;&#xd328;&#xcee4;(Douglas-Peucker) &#xc54c;&#xace0;&#xb9ac;&#xc998;&#xc744; &#xc791;&#xc6a9;&#xd574;&#xc11c; &#xc785;&#xb825; TopoGeometry &#xc758; "&#xb2e8;&#xc21c;&#xd654;&#xb41c;" &#xb3c4;&#xd615; &#xbc84;&#xc804;&#xc744; &#xb18;&#xd658;&#xd569;&#xb2c8;&#xb2e4;.

### Note



&#xb18;&#xd658;&#xb41c; &#xb3c4;&#xd615;&#xc774; &#xb2e8;&#xc21c;&#xd558;&#xc9c0;  
 &#xc54a;&#xac70;&#xb098;, &#xc720;&#xd6a8;&#xd558;&#xc9c0; &#xc54a;&#xc744; &#xc218;&#xb3c4;  
 &#xc788;&#xc2b5;&#xb2c8;&#xb2e4;.  
 &#xb2e8;&#xc21c;&#xc131;/&#xc720;&#xd6a8;&#xc131;&#xc744; &#xc720;&#xc9c0;&#xd558;&#xb294;  
 &#xb370; &#xad6c;&#xc131; &#xc694;&#xc18c;&#xc758; &#xacbd;&#xacc4;&#xc120;&#xc744;  
 &#xbd84;&#xd560;&#xd558;&#xb294; &#xac83;&#xc774; &#xb3c4;&#xc6c0;&#xc774; &#xb420; &#xc218;&#xb3c4;  
 &#xc788;&#xc2b5;&#xb2c8;&#xb2e4;.

GEOS &#xbaa8;&#xb4c8;&#xb85c; &#xc2e4;&#xd589;

2.1.0 &#xbc84;&#xc804;&#xbd80;&#xd130; &#xc0ac;&#xc6a9;&#xd560; &#xc218; &#xc788;&#xc2b5;&#xb2c8;&#xb2e4;.

**ST\_Simplify**

[ST\\_Simplify](#), [ST\\_IsSimple](#), [ST\\_IsValid](#), [ST\\_ModEdgeSplit](#)

## 10.8.6 RemoveUnusedPrimitives

**RemoveUnusedPrimitives** — Removes topology primitives which not needed to define existing TopoGeometry objects.

### Synopsis

```
int RemoveUnusedPrimitives(text topology_name, geometry bbox);
```

**ST\_RemoveUnusedPrimitives**

Finds all primitives (nodes, edges, faces) that are not strictly needed to represent existing TopoGeometry objects and removes them, maintaining topology validity (edge linking, face labeling) and TopoGeometry space occupation.

No new primitive identifiers are created, but rather existing primitives are expanded to include merged faces (upon removing edges) or healed edges (upon removing nodes).

Availability: 3.3.0

**ST\_ModEdgeHeal**

[ST\\_ModEdgeHeal](#), [ST\\_RemEdgeModFace](#)

## 10.9 TopoGeometry

### 10.9.1 CreateTopoGeom

**CreateTopoGeom** — Creates a TopoGeometry object for layer denoted by `layer_id` and registers it in the relations table in the `toponame` schema. `tg_type` is an integer: 1:[multi]point (punctal), 2:[multi]line (lineal), 3:[multi]poly (areal), 4:collection. `layer_id` is the layer id in the topology.layer table.

### Synopsis

```
topogeometry CreateTopoGeom(varchar toponame, integer tg_type, integer layer_id, topoelementarray tg_objs);
topogeometry CreateTopoGeom(varchar toponame, integer tg_type, integer layer_id);
```

**ST\_CreateTopoGeom**

Creates a topogeometry object for layer denoted by `layer_id` and registers it in the relations table in the `toponame` schema. `tg_type` is an integer: 1:[multi]point (punctal), 2:[multi]line (lineal), 3:[multi]poly (areal), 4:collection. `layer_id` is the layer id in the topology.layer table.

**ST\_CreateTopoGeom**

**ST\_CreateTopoGeom**

Availability: 1.1

**TopologyElementArray**

Create a topogeom in ri\_topo schema for layer 2 (our ri\_roads), of type (2) LINE, for the first edge (we loaded in ST\_CreateTopoGeo)

```
INSERT INTO ri.ri_roads(road_name, topo) VALUES('Unknown', topology.CreateTopoGeom('ri_topo ←
', 2, 2, '{{1,2}}':topology.topoelementarray);
```

**TopologyElementArray**

```
&#xd45c;&#xba74;&#xb4e4;&#xc758; &#xc9d1;&#xd569;&#xc73c;&#xb85c; &#xd615;&#xc131;&#xb3fc;&#xc57c; &#xd558;&#
&#xb3c4;&#xd615;&#xc744; &#xac00;&#xc9c0;&#xace0; &#xc788;&#xb2e4;&#xace0; &#xd574;&#xbd05;&#xc2dc;&#xb2e4;.
&#xc608;&#xb97c; &#xb4e4;&#xc5b4; blockgroups &#xd14c;&#xc774;&#xbe14;&#xc774; &#xc788;&#xb294;&#xb370; &#xac01;
&#xbe14;&#xb85d; &#xadf8;&#xb8f9;&#xc758; TopoGeometry&#xb97c; &#xc54c;&#xace0; &#xc2f6;&#xc2b5;&#xb2c8;&#xb2e4
&#xb370;&#xc774;&#xd130;&#xac00; &#xc644;&#xbcbd;&#xd558;&#xac8c; &#xc815;&#xb82c;&#xb3fc; &#xc788;&#xb2e4;&#
&#xb2e4;&#xc74c;&#xacfc; &#xac19;&#xc774; &#xd560; &#xc218; &#xc788;&#xc2b5;&#xb2c8;&#xb2e4;:
```

```
-- TopoGeometry &#xc5f4;&#xc744; &#xc0dd;&#xc131; --
SELECT topology.AddTopoGeometryColumn(
    'topo_boston',
    'boston', 'blockgroups', 'topo', 'POLYGON');

-- addtopogeometrycolumn --
1

-- &#xbaa8;&#xb4e0; &#xac83;&#xc774; &#xacbd;&#xacc4;&#xc120;&#xacfc; ←
&#xc644;&#xbcbd;&#xd558;&#xac8c; &#xc815;&#xb82c;&#xb41c;&#xb2e4;&#xace0; ←
&#xac00;&#xc815;&#xd558;&#xace0;
-- &#xc5f4;&#xc744; &#xc5c5;&#xb370;&#xc774;&#xd2b8;&#xd569; &#xb2c8;&#xb2e4; .
UPDATE boston.blockgroups AS bg
    SET topo = topology.CreateTopoGeom('topo_boston'
    , 3, 1
    , foo.bfaces)
FROM (SELECT b.gid, topology.TopoElementArray_Agg(ARRAY[f.face_id,3]) As bfaces
    FROM boston.blockgroups As b
        INNER JOIN topo_boston.face As f ON b.geom && f.mbr
        WHERE ST_Covers(b.geom, topology.ST_GetFaceGeometry('topo_boston', f.face_id))
        GROUP BY b.gid) As foo
WHERE foo.gid = bg.gid;
```

```
--the world is rarely perfect allow for some error
--count the face if 50% of it falls
-- within what we think is our blockgroup boundary
UPDATE boston.blockgroups AS bg
    SET topo = topology.CreateTopoGeom('topo_boston'
    , 3, 1
    , foo.bfaces)
FROM (SELECT b.gid, topology.TopoElementArray_Agg(ARRAY[f.face_id,3]) As bfaces
    FROM boston.blockgroups As b
        INNER JOIN topo_boston.face As f ON b.geom && f.mbr
        WHERE ST_Covers(b.geom, topology.ST_GetFaceGeometry('topo_boston', f.face_id))
        OR
        ( ST_Intersects(b.geom, topology.ST_GetFaceGeometry('topo_boston', f.face_id))
            AND ST_Area(ST_Intersection(b.geom, topology.ST_GetFaceGeometry('topo_boston', ←
            f.face_id) ) ) >
            ST_Area(topology.ST_GetFaceGeometry('topo_boston', f.face_id))*0.5
        )
        GROUP BY b.gid) As foo
WHERE foo.gid = bg.gid;
```

```
-- and if we wanted to convert our topogeometry back
-- to a denormalized geometry aligned with our faces and edges
-- cast the topo to a geometry
-- The really cool thing is my new geometries
-- are now aligned with my tiger street centerlines
UPDATE boston.blockgroups SET new_geom = topo::geometry;
```

**TopoGeometry**

Add **TopoGeometryColumn**, **toTopoGeom**, **ST\_CreateTopoGeo**, **ST\_GetFaceGeometry**, **TopoElementArray**, **TopoElementArray\_Agg**

## 10.9.2 toTopoGeom

**toTopoGeom** — Converts a simple Geometry into a topo geometry.

### Synopsis

topogeometry **toTopoGeom**(geometry geom, varchar toponame, integer layer\_id, float8 tolerance);

topogeometry **toTopoGeom**(geometry geom, topogeometry topogeom, float8 tolerance);

**TopoGeometry**

**TopoGeometry** (topogeometry topogeom, varchar toponame, integer layer\_id, float8 tolerance);

**TopoGeometry** (topogeometry topogeom, topogeometry topogeom, float8 tolerance);

**TopoGeometry** (topogeometry topogeom, varchar toponame, integer layer\_id, float8 tolerance);

**TopoGeometry** (topogeometry topogeom, varchar toponame, integer layer\_id, float8 tolerance, integer relation\_id);

**TopoGeometry** (topogeometry topogeom, varchar toponame, integer layer\_id, float8 tolerance, integer relation\_id, integer face\_id);

**TopoGeometry** (topogeometry topogeom, varchar toponame, integer layer\_id, float8 tolerance, integer relation\_id, integer face\_id, integer edge\_id);

**TopoGeometry** (topogeometry topogeom, varchar toponame, integer layer\_id, float8 tolerance, integer relation\_id, integer face\_id, integer edge\_id, integer vertex\_id);

**TopoGeometry** (topogeometry topogeom, varchar toponame, integer layer\_id, float8 tolerance, integer relation\_id, integer face\_id, integer edge\_id, integer vertex\_id, integer point\_id);

**TopoGeometry** (topogeometry topogeom, varchar toponame, integer layer\_id, float8 tolerance, integer relation\_id, integer face\_id, integer edge\_id, integer vertex\_id, integer point\_id, integer area\_id);

**TopoGeometry** (topogeometry topogeom, varchar toponame, integer layer\_id, float8 tolerance, integer relation\_id, integer face\_id, integer edge\_id, integer vertex\_id, integer point\_id, integer area\_id, integer volume\_id);

**TopoGeometry** (topogeometry topogeom, varchar toponame, integer layer\_id, float8 tolerance, integer relation\_id, integer face\_id, integer edge\_id, integer vertex\_id, integer point\_id, integer area\_id, integer volume\_id, integer surface\_id);

**TopoGeometry** (topogeometry topogeom, varchar toponame, integer layer\_id, float8 tolerance, integer relation\_id, integer face\_id, integer edge\_id, integer vertex\_id, integer point\_id, integer area\_id, integer volume\_id, integer surface\_id, integer volume\_id);

**TopoGeometry** (topogeometry topogeom, varchar toponame, integer layer\_id, float8 tolerance, integer relation\_id, integer face\_id, integer edge\_id, integer vertex\_id, integer point\_id, integer area\_id, integer volume\_id, integer surface\_id, integer volume\_id, integer surface\_id);

**TopoGeometry** (topogeometry topogeom, varchar toponame, integer layer\_id, float8 tolerance, integer relation\_id, integer face\_id, integer edge\_id, integer vertex\_id, integer point\_id, integer area\_id, integer volume\_id, integer surface\_id, integer volume\_id, integer surface\_id, integer volume\_id);

**TopoGeometry** (topogeometry topogeom, varchar toponame, integer layer\_id, float8 tolerance, integer relation\_id, integer face\_id, integer edge\_id, integer vertex\_id, integer point\_id, integer area\_id, integer volume\_id, integer surface\_id, integer volume\_id, integer surface\_id, integer volume\_id);

**TopoGeometry** (topogeometry topogeom, varchar toponame, integer layer\_id, float8 tolerance, integer relation\_id, integer face\_id, integer edge\_id, integer vertex\_id, integer point\_id, integer area\_id, integer volume\_id, integer surface\_id, integer volume\_id, integer surface\_id, integer volume\_id);

**TopoGeometry** (topogeometry topogeom, varchar toponame, integer layer\_id, float8 tolerance, integer relation\_id, integer face\_id, integer edge\_id, integer vertex\_id, integer point\_id, integer area\_id, integer volume\_id, integer surface\_id, integer volume\_id, integer surface\_id, integer volume\_id);

**TopoGeometry** (topogeometry topogeom, varchar toponame, integer layer\_id, float8 tolerance, integer relation\_id, integer face\_id, integer edge\_id, integer vertex\_id, integer point\_id, integer area\_id, integer volume\_id, integer surface\_id, integer volume\_id, integer surface\_id, integer volume\_id);

**;**

;

```
-- &#xc544;&#xc9c1; &#xc9c0;&#xd615;&#xc744; &#xc124;&#xc815;&#xd558;&#xc9c0; ←
&#xc54a;&#xc558;&#xb2e4;&#xba74; &#b2e4;&#xc74c; &#xcffc;&#xb9ac;&#xb97c; ←
&#xc2e4;&#xd589;&#xd558;&#xc2ed;&#xc2dc;&#xc624;.
-- &#xc5b4;&#xb5a4; &#xd5c8;&#xc6a9; &#xc624;&#xcc28;&#xb3c4; ←
&#xd5c8;&#xc6a9;&#xd558;&#xc9c0; &#xc54a;&#xb294; &#xc9c0;&#xd615;&#xc744; ←
&#xc0dd;&#xc131;&#xd569;&#xb2c8;&#xb2e4;.
SELECT topology.CreateTopology('topo_boston_test', 2249);
-- &#xc0c8; &#xd14c;&#xc774;&#xbe14; &#xc0dd;&#xc131;
CREATE TABLE nei_topo(gid serial primary key, nei varchar(30));
-- &#xc0c8; &#xd14c;&#xc774;&#xbe14;&#xc5d0; TopoGeometry &#xc5f4;&#xc744; &#xcd94;&#xac00;
SELECT topology.AddTopoGeometryColumn('topo_boston_test', 'public', 'nei_topo', 'topo', ' ←
MULTIPOLYGON') As new_layer_id;
new_layer_id
-----
1
-- &#xc0c8; TopoGeometry &#xc5f4;&#xc744; &#xcc44;&#xc6b0;&#xb294; &#xb370; &#xc0c8; ←
&#xb808;&#xc774;&#xc5b4; ID&#xb97c; &#xc774;&#xc6a9;
-- &#xc0c8; &#xb808;&#xc774;&#xc5b4;&#xc5d0; &#xd5c8;&#xc6a9; &#xc624;&#xcc28;&#xac00; 0 ←
&#xc778; TopoGeometry&#xb97c; &#xcd94;&#xac00;
INSERT INTO nei_topo(nei, topo)
SELECT nei, topology.toTopoGeom(geom, 'topo_boston_test', 1)
FROM neighborhoods
WHERE gid BETWEEN 1 and 15;
-- &#xc791;&#xc5c5; &#xb0b4;&#xc6a9;&#xc744; &#xd655;&#xc778;&#xd558;&#xb824;&#xba74; ←
&#xb2e4;&#xc74c; &#xcffc;&#xb9ac;&#xb97c; &#xc774;&#xc6a9;&#xd569;&#xb2c8;&#xb2e4;. --
SELECT * FROM
topology.TopologySummary('topo_boston_test');
-- &#xc694;&#xc57d;&#xbb38; --
Topology topo_boston_test (5), SRID 2249, precision 0
61 nodes, 87 edges, 35 faces, 15 topogeoms in 1 layers
Layer 1, type Polygonal (3), 15 topogeoms
Deploy: public.nei_topo.topo
-- &#xbaa8;&#xb4e0; TopoGeometry &#xd3f4;&#xb9ac;&#xace4;&#xc744; 10 ←
&#xbbf8;&#xd130;&#xc529; &#xc904;&#xc785;&#xb2c8;&#xb2e4;.
UPDATE nei_topo SET topo = ST_Buffer(clearTopoGeom(topo), -10);
-- &#xc704;&#xc758; &#xc5f0;&#xc0b0;&#xc73c;&#xb85c; &#xb0a8;&#xc740; ←
&#xc544;&#xbb34;&#xac83;&#xb3c4; &#xc5c6;&#xb294; &#xd1a0;&#xc9c0; (no-one-lands) ←
&#xb97c; &#xc9d1;&#xacc4;&#xd569;&#xb2c8;&#xb2e4;.
-- GRASS&#xc5d0;&#xc11c;&#xb294; &#xc774;&#xac78; "polygon0 layer"&#xb77c;&#xace0; ←
&#xbd80;&#xb97c; &#xac81;&#xb2c8;&#xb2e4;.
SELECT ST_GetFaceGeometry('topo_boston_test', f.face_id)
FROM topo_boston_test.face f
WHERE f.face_id
> 0 -- don't consider the universe face
AND NOT EXISTS ( -- check that no TopoGeometry references the face
SELECT * FROM topo_boston_test.relation
WHERE layer_id = 1 AND element_id = f.face_id
);
```





**Example:**

```
-- TopoGeometry tg;
--> SELECT ST_AddElement(tg, ST_GeomFromText('LINESTRING(0 0, 1 0)'));
UPDATE mylayer SET tg = TopoGeom_removeElement(tg, '{43,3}');
```

**See also:**

[TopoGeom\\_addElement](#), [CreateTopoGeom](#)

## 10.10.4 TopoGeom\_addTopoGeom

`TopoGeom_addTopoGeom` — Adds element of a TopoGeometry to the definition of another TopoGeometry.

### Synopsis

topogeometry **TopoGeom\_addTopoGeom**(topogeometry tgt, topogeometry src);

**Parameters:**

Adds the elements of a **TopoGeometry** to the definition of another TopoGeometry, possibly changing its cached type (type attribute) to a collection, if needed to hold all elements in the source object.

The two TopoGeometry objects need be defined against the *same* topology and, if hierarchically defined, need be composed by elements of the same child layer.

Availability: 3.2

**Example:**

```
-- Set an "overall" TopoGeometry value to be composed by all
-- elements of specific TopoGeometry values
UPDATE mylayer SET tg_overall = TopoGeom_addTopoGeom(
    TopoGeom_addTopoGeom(
        clearTopoGeom(tg_overall),
        tg_specific1
    ),
    tg_specific2
);
```

**See also:**

[TopoGeom\\_addElement](#), [clearTopoGeom](#), [CreateTopoGeom](#)

## 10.10.5 toTopoGeom

`toTopoGeom` — Adds a geometry shape to an existing topo geometry.

**Parameters:**

Refer to [toTopoGeom](#).



**GetTopoGeomElementArray**

**TopoElement**

**TopoGeom\_addElement**, **TopoGeom\_remElement**

### 10.11.3 ST\_SRID

**ST\_SRID** — Returns the spatial reference identifier for a topogeometry.

#### Synopsis

integer **ST\_SRID**(topogeometry tg);

**Returns**

Returns the spatial reference identifier for the ST\_Geometry as defined in spatial\_ref\_sys table. Section [4.5](#)



#### Note

spatial\_ref\_sys table is a table that catalogs all spatial reference systems known to PostGIS and is used for transformations from one spatial reference system to another. So verifying you have the right spatial reference system identifier is important if you plan to ever transform your geometries.

Availability: 3.2.0



This method implements the SQL/MM specification. SQL-MM 3: 14.1.5

**Example**

```
SELECT ST_SRID(ST_GeomFromText('POINT(-71.1043 42.315)', 4326));
--result
4326
```

**See Also**

Section [4.5](#), [ST\\_SetSRID](#), [ST\\_Transform](#), [ST\\_SRID](#)

## 10.12 TopoGeometry

### 10.12.1 AsGML

**AsGML** — TopoGeometry to GML



```

SELECT topology.AsGML(topo) As rdgml
FROM ri.roads
WHERE road_name = 'Unknown';

-- rdgml--
<gml:TopoCurve>
  <gml:directedEdge>
    <gml:Edge gml:id="E1">
      <gml:directedNode orientation="-">
        <gml:Node gml:id="N1"/>
      </gml:directedNode>
      <gml:directedNode
></gml:directedNode>
      <gml:curveProperty>
        <gml:Curve srsName="urn:ogc:def:crs:EPSG::3438">
          <gml:segments>
            <gml:LineStringSegment>
              <gml:posList srsDimension="2"
>384744 236928 384750 236923 384769 236911 384799 236895 384811 236890
              384833 236884 384844 236882 384866 236881 384879 236883 384954 ←
              236898 385087 236932 385117 236938
              385167 236938 385203 236941 385224 236946 385233 236950 385241 ←
              236956 385254 236971
              385260 236979 385268 236999 385273 237018 385273 237037 385271 ←
              237047 385267 237057 385225 237125
              385210 237144 385192 237161 385167 237192 385162 237202 385159 ←
              237214 385159 237227 385162 237241
              385166 237256 385196 237324 385209 237345 385234 237375 385237 ←
              237383 385238 237399 385236 237407
              385227 237419 385213 237430 385193 237439 385174 237451 385170 ←
              237455 385169 237460 385171 237475
              385181 237503 385190 237521 385200 237533 385206 237538 385213 ←
              237541 385221 237542 385235 237540 385242 237541
              385249 237544 385260 237555 385270 237570 385289 237584 385292 ←
              237589 385291 237596 385284 237630</gml:posList>
            </gml:LineStringSegment>
          </gml:segments>
        </gml:Curve>
      </gml:curveProperty>
    </gml:Edge>
  </gml:directedEdge>
</gml:TopoCurve
>

```

&#xc774;&#xc804; &#xc608;&#xc2dc;&#xc640; &#xb3d9;&#xc77c;&#xd558;&#xc9c0;&#xb9cc; &#xb124;&#xc784;&#xc2a4;&#xc4f0;&#xc9c0; &#xc54a;&#xc2b5;&#xb2c8;&#xb2e4;.

```

SELECT topology.AsGML(topo, '') As rdgml
FROM ri.roads
WHERE road_name = 'Unknown';

-- rdgml--
<TopoCurve>
  <directedEdge>
    <Edge id="E1">
      <directedNode orientation="-">
        <Node id="N1"/>
      </directedNode>
      <directedNode
></directedNode>
      <curveProperty>

```

```

    <Curve srsName="urn:ogc:def:crs:EPSG::3438">
      <segments>
        <LineStringSegment>
          <posList srsDimension="2"
>384744 236928 384750 236923 384769 236911 384799 236895 384811 236890
          384833 236884 384844 236882 384866 236881 384879 236883 384954 ←
            236898 385087 236932 385117 236938
          385167 236938 385203 236941 385224 236946 385233 236950 385241 ←
            236956 385254 236971
          385260 236979 385268 236999 385273 237018 385273 237037 385271 ←
            237047 385267 237057 385225 237125
          385210 237144 385192 237161 385167 237192 385162 237202 385159 ←
            237214 385159 237227 385162 237241
          385166 237256 385196 237324 385209 237345 385234 237375 385237 ←
            237383 385238 237399 385236 237407
          385227 237419 385213 237430 385193 237439 385174 237451 385170 ←
            237455 385169 237460 385171 237475
          385181 237503 385190 237521 385200 237533 385206 237538 385213 ←
            237541 385221 237542 385235 237540 385242 237541
          385249 237544 385260 237555 385270 237570 385289 237584 385292 ←
            237589 385291 237596 385284 237630</posList>
        </LineStringSegment>
      </segments>
    </Curve>
  </curveProperty>
</Edge>
</directedEdge>
</TopoCurve
>

```

**&#x38;&#x3e0;**

CreateTopoGeom, ST\_CreateTopoGeo

## 10.12.2 AsTopoJSON

AsTopoJSON — TopoGeometry&#xc758; TopoJSON&#xd45c;&#xd604;&#xc2dd;&#xc744; &#xbc18;&#xd658;&#xd569;&#xb2c8;&#xb2e4;

### Synopsis

text AsTopoJSON(topogeometry tg, regclass edgeMapTable);

**&#xc124;&#xba85;**

TopoGeometry&#xc758; TopoJSON &#xd45c;&#xd604;&#xc2dd;&#xc744; &#xbc18;&#xd658;&#xd569;&#xb2c8;&#xb2e4;. edgeMapTable &#xc774; NULL&#xc774; &#xc544;&#xb2d0; &#xacbd;&#xc6b0; &#xacbd;&#xacc4;&#xc120; &#xc2dd;&#xb2c8;&#xd638;(arc) &#xc778;&#xb371;&#xc2a4;&#xc5d0; &#xb9e4;&#xd551;&#xd558;&#xb294; &#xd0d0;&#xc0c9;/&#xc80&#xc4f0;&#xc77c; &#xac83;&#xc785;&#xb2c8;&#xb2e4;. &#xcd5c;&#xc885; &#xbb38;&#xc11c;&#xc5d0; &#xc870;&#xbc00;&#&#xc6d0;&#xd638;" &#xbc30;&#xc5f4;&#xc744; &#xc0ac;&#xc6a9;&#xd560; &#xc218; &#xc788;&#xac8c; &#xd558;&#xae30; &#xc704;&#xd574;&#xc11c;&#xc785;&#xb2c8;&#xb2e4;.

&#xd14c;&#xc774;&#xbe14;&#xc744; &#xc124;&#xc815;&#xd560; &#xacbd;&#xc6b0; &#xd14c;&#xc774;&#xbe14;&#xc774; &#xc21c;&#xcc28;(serial)" &#xc720;&#xd615; "arc\_id" &#xd56d;&#xbaa9;&#xacfc; &#xc815;&#xc218;&#xd615; "edge\_id" &#xd56d;&#xbaa9;&#xc744; &#xac00;&#xc9c0;&#xace0; &#xc788;&#xc5b4;&#xc57c; &#xd569;&#xb2c8;&#xb2e4;. &#xcf54;&#&#xd574;&#xb2f9; &#xd56d;&#xbaa9;&#xc5d0; &#xc778;&#xb371;&#xc2a4;&#xb97c; &#xcd94;&#xac00;&#xd558;&#xb294; &#xd3b8;&#xc774; &#xc88b;&#xc2b5;&#xb2c8;&#xb2e4;.

**Note**

TopoJSON `0-1` "edgeMapTable"

TopoJSON, `0-1` "edgeMapTable"

2.1.0, `0-1`

2.2.1 (puntal) `0-1`

`0-1`

**ST\_AsGeoJSON**

`0-1`

```
CREATE TEMP TABLE edgemap(arc_id serial, edge_id int unique);

-- 0-1;
SELECT '{ "type": "Topology", "transform": { "scale": [1,1], "translate": [0,0] }, "objects ←
": { '

-- 0-1;
UNION ALL SELECT '' || feature_name || ': ' || AsTopoJSON(feature, 'edgemap')
FROM features.big_parcel WHERE feature_name = 'P3P4';

-- 0-1;
WITH edges AS (
  SELECT m.arc_id, e.geom FROM edgemap m, city_data.edge e
  WHERE e.edge_id = m.edge_id
), points AS (
  SELECT arc_id, (st_dumppoints(geom)).* FROM edges
), compare AS (
  SELECT p2.arc_id,
     CASE WHEN p1.path IS NULL THEN p2.geom
          ELSE ST_Translate(p2.geom, -ST_X(p1.geom), -ST_Y(p1.geom))
     END AS geom
  FROM points p2 LEFT OUTER JOIN points p1
  ON ( p1.arc_id = p2.arc_id AND p2.path[1] = p1.path[1]+1 )
  ORDER BY arc_id, p2.path
), arcsdump AS (
  SELECT arc_id, (regexp_matches( ST_AsGeoJSON(geom), '\[.*\]'))[1] as t
  FROM compare
), arcs AS (
  SELECT arc_id, '[' || array_to_string(array_agg(t), ',') || ']' as a FROM arcsdump
  GROUP BY arc_id
  ORDER BY arc_id
)
SELECT ', "arcs": [' UNION ALL
SELECT array_to_string(array_agg(a), E',\n') from arcs

-- 0-1;
UNION ALL SELECT ']'::text as t;
```

```
--
{ "type": "Topology", "transform": { "scale": [1,1], "translate": [0,0] }, "objects": {
  "P3P4": { "type": "MultiPolygon", "arcs": [[[-1]], [[6,5,-5,-4,-3,1]]]
}, "arcs": [
  [[25,30],[6,0],[0,10],[-14,0],[0,-10],[8,0]],
  [[35,6],[0,8]],
  [[35,6],[12,0]],
  [[47,6],[0,8]],
  [[47,14],[0,8]],
  [[35,22],[12,0]],
  [[35,14],[0,8]]
]]
}
```

## 10.13 ST\_Equals

### 10.13.1 Equals

Equals — Returns true if two TopoGeometry objects are equal. The objects must be of the same type (e.g., both are TopoGeometry or both are TopoLineString).

#### Synopsis

boolean **Equals**(topogeometry tg1, topogeometry tg2);

#### Notes

Two TopoGeometry objects are equal if they have the same topology and the same set of elements. The elements must be of the same type (e.g., both are TopoLineString or both are TopoPolygon).



#### Note

ST\_Equals is a binary function that returns true if two TopoGeometry objects are equal. The objects must be of the same type (e.g., both are TopoGeometry or both are TopoLineString). The objects must also have the same topology and the same set of elements. The elements must be of the same type (e.g., both are TopoLineString or both are TopoPolygon).

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



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

#### ST\_Equals

#### ST\_Equals

[ST\\_Equals](#), [ST\\_Equals](#)

## 10.13.2 ST\_Intersects

ST\_Intersects(*topogeometry* *tg1*, *topogeometry* *tg2*);

### Synopsis

boolean ST\_Intersects(*topogeometry* *tg1*, *topogeometry* *tg2*);

### Notes

ST\_Intersects(*topogeometry* *tg1*, *topogeometry* *tg2*);

#### Note



ST\_Intersects(*topogeometry* *tg1*, *topogeometry* *tg2*);

1.1.0 ST\_Intersects(*topogeometry* *tg1*, *topogeometry* *tg2*);



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

### ST\_Intersects

### ST\_Intersects

### ST\_Intersects

## 10.14 Importing and exporting Topologies

Once you have created topologies, and maybe associated topological layers, you might want to export them into a file-based format for backup or transfer into another database.

Using the standard dump/restore tools of PostgreSQL is problematic because topologies are composed by a set of tables (4 for primitives, an arbitrary number for layers) and records in metadata tables (topology.topology and topology.layer). Additionally, topology identifiers are not univoque across databases so that parameter of your topology will need to be changes upon restoring it.

In order to simplify export/restore of topologies a pair of executables are provided: `pgtopo_export` and `pgtopo_import`. Example usage:

```
pgtopo_export dev_db topo1 | pgtopo_import topo1 | psql staging_db
```

### 10.14.1 Using the Topology exporter

The `pgtopo_export` script takes the name of a database and a topology and outputs a dump file which can be used to import the topology (and associated layers) into a new database.

By default `pgtopo_export` writes the dump file to the standard output so that it can be piped to `pgtopo_import` or redirected to a file (refusing to write to terminal). You can optionally specify an output filename with the `-f` commandline switch.

By default `pgtopo_export` includes a dump of all layers defined against the given topology. This may be more data than you need, or may be non-working (in case your layer tables have complex dependencies) in which case you can request skipping the layers with the `--skip-layers` switch and deal with those separately.

Invoking `pgtopo_export` with the `--help` (or `-h` for short) switch will always print short usage string.

The dump file format is a compressed tar archive of a `pgtopo_export` directory containing at least a `pgtopo_dump_version` file with format version info. As of version 1 the directory contains tab-delimited CSV files with data of the topology primitive tables (node, edge\_data, face, relation), the topology and layer records associated with it and (unless `--skip-layers` is given) a custom-format PostgreSQL dump of tables reported as being layers of the given topology.

### 10.14.2 Using the Topology importer

The `pgtopo_import` script takes a `pgtopo_export` format topology dump and a name to give to the topology to be created and outputs an SQL script reconstructing the topology and associated layers.

The generated SQL file will contain statements that create a topology with the given name, load primitive data in it, restores and registers all topology layers by properly linking all TopoGeometry values to their correct topology.

By default `pgtopo_import` reads the dump from the standard input so that it can be used in conjunction with `pgtopo_export` in a pipeline. You can optionally specify an input filename with the `-f` commandline switch.

By default `pgtopo_import` includes in the output SQL file the code to restore all layers found in the dump.

This may be unwanted or non-working in case your target database already have tables with the same name as the ones in the dump. In that case you can request skipping the layers with the `--skip-layers` switch and deal with those separately (or later).

SQL to only load and link layers to a named topology can be generated using the `--only-layers` switch. This can be useful to load layers AFTER resolving the naming conflicts or to link layers to a different topology (say a spatially-simplified version of the starting topology).

## Chapter 11

# 11. Raster Data Types

### 11.1 Raster Data Types

The raster data types are defined in the `raster2pgsql` module. The raster data types are defined in the `raster2pgsql` module. The raster data types are defined in the `raster2pgsql` module. The raster data types are defined in the `raster2pgsql` module.

#### 11.1.1 raster2pgsql

The `raster2pgsql` module is used to create raster data types. The raster data types are defined in the `raster2pgsql` module. The raster data types are defined in the `raster2pgsql` module. The raster data types are defined in the `raster2pgsql` module.

Since the `raster2pgsql` is compiled as part of PostGIS most often (unless you compile your own GDAL library), the raster types supported by the executable will be the same as those compiled in the GDAL dependency library. To get a list of raster types your particular `raster2pgsql` supports use the `-G` switch. These should be the same as those provided by your PostGIS install documented here [ST\\_GDALDrivers](#) if you are using the same GDAL library for both.

**Note**

GDAL PostGIS raster2pgsql PostGIS

Note!

**Note**

GDAL PostGIS raster2pgsql

Note!

raster2pgsql raster\_options\_go\_here raster\_file someschema.sometable > out.sql

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

-? raster2pgsql raster\_options\_go\_here raster\_file someschema.sometable > out.sql

-G raster2pgsql raster\_options\_go\_here raster\_file someschema.sometable > out.sql

-c raster2pgsql raster\_options\_go\_here raster\_file someschema.sometable > out.sql

-a raster2pgsql raster\_options\_go\_here raster\_file someschema.sometable > out.sql

-p raster2pgsql raster\_options\_go\_here raster\_file someschema.sometable > out.sql

&#x798;&#xc2a4;&#xd130; &#xc2a4;&#xd130; &#xc2a4;&#xd130; &#xc2a4;&#xd130;

-C raster2pgsql raster\_options\_go\_here raster\_file someschema.sometable > out.sql

-x raster2pgsql raster\_options\_go\_here raster\_file someschema.sometable > out.sql





```

&#xb9e4;&#xc0ac;&#xcd94;&#xc138;&#xce20; &#xc8fc;&#xc758; &#xbbf8;&#xd130; &#xb2e8;&#xc704; &#xd56d;&#xacf5;&#x
&#xd0c0;&#xc77c; &#xb798;&#xc2a4;&#xd130;&#xb4e4;&#xc744; aerial &#xc774;&#xb77c;&#xb294; &#xc2a4;&#xd0a4;&#x
&#xb85c;&#xb4dc;&#xd558;&#xace0;, &#xc804;&#xc9cb4; &#xbdf0;&#xc640; 2&#xb808;&#xbca8; &#xbc0f; 4&#xb808;&#xbca8;
&#xc624;&#xb84;&#xbdf0; &#xd14c;&#xc774;&#xbe14;&#xc744; &#xc0dd;&#xc131;&#xd55c; &#xb2e4;&#xc74c;, &#bcf5;&#x
&#xaa8;&#xb4dc;&#xb97c; &#xd1b5;&#xd574; (&#xc911;&#xac04; &#xb2e8;&#xacc4; &#xd30c;&#xc77c; &#xc5c6;&#xc774;
DB&#xb85c; &#xc9c1;&#xc811;) &#xc0bd;&#xc785;&#xd558;&#xba70;, &#xac15;&#xc81c;&#xb85c; &#xaa8;&#xb4e0;
&#xc791;&#xc5c5;&#xc744; &#xc0c1;&#xd638;&#xc98;&#xb9ac;&#xd558;&#xc9c0; &#xc54a;&#xb3c4;&#xb85d; -e &#xd50c;&#x
&#xc0ac;&#xc6a9;&#xd558;&#xc2ed;&#xc2dc;&#xc624;(&#xc791;&#xc5c5;&#xc774; &#xc644;&#xb8cc;&#xb418;&#xae38;
&#xae30;&#xb2e4;&#xb9ac;&#xc9c0; &#xc54a;&#xace0; &#xd14c;&#xc774;&#xbe14;&#xc5d0; &#xb4e4;&#xc5b4;&#xc624;&#x
&#xb370;&#xc774;&#xd130;&#xb97c; &#xbc14;&#xb85c; &#xc0b4;&#xd3b4;&#xbcf4;&#xace0;&#xc790; &#xd560; &#xb54c;
&#xc720;&#xc6a9;&#xd569;&#xb2c8;&#xb2e4;). &#xb798;&#xc2a4;&#xd130;&#xb97c; 128x128 &#xd53d;&#xc140; &#xd0c0;&#x
&#xbd84;&#xc808;&#xd55c; &#xb2e4;&#xc74c; &#xb798;&#xc2a4;&#xd130; &#xc81c;&#xc57d;&#xc870;&#xac74;&#xc744;
&#xc801;&#xc6a9;&#xd558;&#xc2ed;&#xc2dc;&#xc624;. &#xd14c;&#xc774;&#xbe14; &#xc0bd;&#xc785; &#xb300;&#xc2e0;
&#bcf5;&#xc0ac; &#xaa8;&#xb4dc;&#xb97c; &#xc774;&#xc6a9;&#xd558;&#xc2ed;&#xc2dc;&#xc624;. &#xd0c0;&#xc77c;&#x
&#xc798;&#xb77c;&#xc838; &#xb098;&#xc628; &#xd0c0;&#xc77c; &#xd30c;&#xc77c; &#xba85;&#xc744; &#xb2f4;&#xc744;
&#xc218; &#xc788;&#xb3c4;&#xb85d; -F &#xd50c;&#xb798;&#xadf8;&#xb85c; "filename"&#xc774;&#xb77c;&#xb294; &#xd544;
&#xd3ec;&#xd568;&#xc2dc;&#xd0a4;&#xc2ed;&#xc2dc;&#xc624;.

```

```

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

```

```

-- &#xc9c0;&#xc6d0;&#xb418;&#xb294; &#xb798;&#xc2a4;&#xd130; &#xc720;&#xd615;&#xc758; ←
    &#xaa9;&#xb85d;&#xc744; &#xc5bb;&#xc73c;&#xb824;&#xba74; :
raster2pgsql -G

```

```

-G &#xc635;&#xc158;&#xc774; &#xb2e4;&#xc74c;&#xacfc; &#xac19;&#xc740; &#xaa9;&#xb85d;&#xc744; &#xcd9c;&#xb825;&#x
&#xac83;&#xc785;&#xb2c8;&#xb2e4;:

```

Available GDAL raster formats:

```

Virtual Raster
GeoTIFF
National Imagery Transmission Format
Raster Product Format TOC format
ECRG TOC format
Erdas Imagine Images (.img)
CEOS SAR Image
CEOS Image
JAXA PALSAR Product Reader (Level 1.1/1.5)
Ground-based SAR Applications Testbed File Format (.gff)
ELAS
Arc/Info Binary Grid
Arc/Info ASCII Grid
GRASS ASCII Grid
SDTS Raster
DTED Elevation Raster
Portable Network Graphics
JPEG JFIF
In Memory Raster
Japanese DEM (.mem)
Graphics Interchange Format (.gif)
Graphics Interchange Format (.gif)
Envisat Image Format
Maptech BSB Nautical Charts
X11 Pixmap Format
MS Windows Device Independent Bitmap
SPOT DIMAP
AirSAR Polarimetric Image
RadarSat 2 XML Product
PCIDSK Database File
PCRaster Raster File

```

ILWIS Raster Map  
SGI Image File Format 1.0  
SRTMHGT File Format  
Leveller heightfield  
Terragen heightfield  
USGS Astrogeology ISIS cube (Version 3)  
USGS Astrogeology ISIS cube (Version 2)  
NASA Planetary Data System  
EarthWatch .TIL  
ERMapper .ers Labelled  
NOAA Polar Orbiter Level 1b Data Set  
FIT Image  
GRIdded Binary (.grb)  
Raster Matrix Format  
EUMETSAT Archive native (.nat)  
Idrisi Raster A.1  
Intergraph Raster  
Golden Software ASCII Grid (.grd)  
Golden Software Binary Grid (.grd)  
Golden Software 7 Binary Grid (.grd)  
COSAR Annotated Binary Matrix (TerraSAR-X)  
TerraSAR-X Product  
DRDC COASP SAR Processor Raster  
R Object Data Store  
Portable Pixmap Format (netpbm)  
USGS DOQ (Old Style)  
USGS DOQ (New Style)  
ENVI .hdr Labelled  
ESRI .hdr Labelled  
Generic Binary (.hdr Labelled)  
PCI .aux Labelled  
Vexcel MFF Raster  
Vexcel MFF2 (HKV) Raster  
Fuji BAS Scanner Image  
GSC Geogrid  
EOSAT FAST Format  
VTP .bt (Binary Terrain) 1.3 Format  
Erdas .LAN/.GIS  
Convair PolGASP  
Image Data and Analysis  
NLAPS Data Format  
Erdas Imagine Raw  
DIPEX  
FARSITE v.4 Landscape File (.lcp)  
NOAA Vertical Datum .GTX  
NADCON .los/.las Datum Grid Shift  
NTv2 Datum Grid Shift  
ACE2  
Snow Data Assimilation System  
Swedish Grid RIK (.rik)  
USGS Optional ASCII DEM (and CDED)  
GeoSoft Grid Exchange Format  
Northwood Numeric Grid Format .grd/.tab  
Northwood Classified Grid Format .grc/.tab  
ARC Digitized Raster Graphics  
Standard Raster Product (ASRP/USRP)  
Magellan topo (.blx)  
SAGA GIS Binary Grid (.sdat)  
Kml Super Overlay  
ASCII Gridded XYZ  
HF2/HFZ heightfield raster  
OziExplorer Image File

```
USGS LULC Composite Theme Grid
Arc/Info Export E00 GRID
ZMap Plus Grid
NOAA NGS Geoid Height Grids
```

### 11.1.2 PostGIS

```
&#xc0ac;&#xc6a9;&#xc790;&#xcac0; &#xb798;&#xc2a4;&#xd130;&#xc640; &#xb798;&#xc2a4;&#xd130; &#xd14c;&#xc774;&#xb370;&#xc774;&#xd130;&#xbca0;&#xc774;&#xc2a4; &#xb0b4;&#xbd80;&#xc5d0; &#xc0dd;&#xc131;&#xd558;&#xb824;
&#xd558;&#xb294; &#xacbd;&#xc6b0;&#xcac0; &#xb9ce;&#xc744; &#xac83;&#xc785;&#xb2c8;&#xb2e4;. &#xadf8;&#xb7f0;
&#xc791;&#xc5c5;&#xc744; &#xc704;&#xd55c;. &#xb118;&#xce58;&#xace0;&#xb3c4; &#xb0a8;&#xc744; &#xb9ce;&#xc740;
&#xd568;&#xc218;&#xcac0; &#xc788;&#xc2b5;&#xb2c8;&#xb2e4;. &#xc77c;&#xb18;&#xc801;&#xc778; &#xb2e8;&#xacc4;&#xb2e4;
&#xc74c;&#xacfc; &#xc19;&#xc2b5;&#xb2c8;&#xb2e4;.
```

1. &#xc0c8; &#xb798;&#xc2a4;&#xd130; &#xb808;&#xcf54;&#xb4dc;&#xb97c; &#xb2f4;&#xc744; &#xb798;&#xc2a4;&#xd130; &#xc5f4;&#xc744; &#xcac0;&#xc9c4; &#xd14c;&#xc774;&#xbe14;&#xc744; &#xb2e4;&#xc74c;&#xacfc; &#xc19;&#xc774; &#xc0dd;&#xc131;&#xd558;&#xc2ed;&#xc2dc;&#xc624;;

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

2. &#xd574;&#xb2f9; &#xaa9;&#xd45c;&#xb97c; &#xb3c4;&#xc640;&#xc904; &#xd568;&#xc218;&#xcac0; &#xb9ce;&#xc774; &#xc788;&#xc2b5;&#xb2c8;&#xb2e4;. &#xb2e4;&#xb978; &#xb798;&#xc2a4;&#xd130;&#xc5d0;&#xc11c; &#xd30c;&#xc0c8; &#xc54a;&#xc740; &#xb798;&#xc2a4;&#xd130;&#xb97c; &#xc0dd;&#xc131;&#xd558;&#xb294; &#xacbd;&#xc6b0;.
 **ST\_MakeEmptyRaster** &#xd568;&#xc218;&#xc640; **ST\_AddBand** &#xd568;&#xc218;&#xb97c; &#xc21c;&#xc11c;&#xb300; &#xc0ac;&#xc6a9;&#xd558;&#xb294; &#xd3b8;&#xc774; &#xc88b;&#xc2b5;&#xb2c8;&#xb2e4;.
 &#xb3c4;&#xd615;&#xc73c;&#xb85c;&#xbd80;&#xd130;&#xb3c4; &#xb798;&#xc2a4;&#xd130;&#xb97c; &#xc0dd;&#xc11c; &#xc218; &#xc788;&#xc2b5;&#xb2c8;&#xb2e4;. &#xadf8;&#xb7ec;&#xb824;&#xba74; **ST\_AsRaster** &#xd568;&#xc218;&#xc544;&#xb9c8;&#xb3c4; **ST\_Union** , **ST\_MapAlgebraFct** &#xb610;&#xb294; &#xb9f5; &#xb300;&#xc218;(algebra) &#xd568;&#xc218; &#xacc4;&#xc5f4;&#xc758; &#xc5b4;&#xb5a4; &#xd568;&#xc218;&#xc640;&#xb3c4; &#xc19;&#xc74c; &#xb2e4;&#xb978; &#xd568;&#xc218;&#xc640; &#xd568;&#xaed8; &#xc0ac;&#xc6a9;&#xd558;&#xb294; &#xd3b8;&#xc774; &#xc88b;&#xc2b5;&#xb2c8;&#xb2e4;.
 &#xae30;&#xc874; &#xd14c;&#xc774;&#xbe14;&#xb85c;&#xbd80;&#xd130; &#xc0c8; &#xb798;&#xc2a4;&#xd130; &#xd14c;&#xc774;&#xbe14;&#xc744; &#xc0dd;&#xc131;&#xd558;&#xb294; &#xb370;&#xc5d0;&#xb294; &#xb354;&#xc0c8; &#xb9ce;&#xc740; &#xc120;&#xd0dd;&#xc9c0;&#xcac0; &#xc788;&#xc2b5;&#xb2c8;&#xb2e4;. &#xc608;&#xb97c; &#xb4e4;&#xc5b4; **ST\_Transform** &#xd568;&#xc218;&#xb97c; &#xc0ac;&#xc6a9;&#xd558;&#xba74; &#xae30;&#xc874; &#xd14c;&#xc774;&#xbe14;&#xacfc;&#xb294; &#xb2e4;&#xb978; &#xd22c;&#xc601;&#xc774; &#xc801;&#xc6a9;&#xb41; &#xb798;&#xc2a4;&#xd130; &#xd14c;&#xc774;&#xbe14;&#xc744; &#xc0dd;&#xc131;&#xd560; &#xc218; &#xc788;&#xc218; &#xc77c;&#xb2e8; &#xc0ac;&#xc6a9;&#xc790; &#xd14c;&#xc774;&#xbe14;&#xc744; &#xcc44;&#xc6b0;&#xb294; &#xc791;&#xc5c5;&#xc744; &#xb9c8;&#xcce4;&#xb2e4;&#xba74;. &#xb2e4;&#xc74c;&#xacfc; &#xc19;&#xc774; &#xb798;&#xc2a4;&#xd130; &#xc5f4;&#xc5d0; &#xb300;&#xd574; &#xacf5;&#xcac04; &#xc778;&#xb371;&#xc2a4;&#xb97c; &#xc0dd;&#xc131;&#xd558;&#xb294; &#xd3b8;&#xc774; &#xc88b;&#xc2b5;&#xb2c8;&#xb2e4;;

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

```
&#xb798;&#xc2a4;&#xd130; &#xc5f0;&#xc0b0;&#xc790; &#xb300;&#xbd80;&#xbd84;&#xc774; &#xb798;&#xc2a4;&#xd130; &#xbcf8;&#xb85d; &#xaeed;&#xc9c8;(convex hull)&#xc744; &#xae30;&#xb18;&#xc73c;&#xb85c; &#xd558;&#xae30;
&#xb54c;&#xbb38;&#xc5d0; ST_ConvexHull &#xd568;&#xc218;&#xb97c; &#xc0ac;&#xc6a9;&#xd588;&#xb2e4;&#xb294;
&#xc810;&#xc5d0; &#xc8fc;&#xc758;&#xd558;&#xc2ed;&#xc2dc;&#xc624;.
```

**Note**

PostGIS 2.0 raster2pgsql -s 990000 -t 256x256 -I -R /vsis3/your.bucket.com/your\_file.tif your\_table | psql your\_db

4. **AddRasterConstraints** your\_table; ALTER RASTER your\_table SET CONSTRAINTS

### 11.1.3 Using "out db" cloud rasters

The `raster2pgsql` tool uses GDAL to access raster data, and can take advantage of a key GDAL feature: the ability to read from rasters that are **stored remotely** in cloud "object stores" (e.g. AWS S3, Google Cloud Storage).

Efficient use of cloud stored rasters requires the use of a "cloud optimized" format. The most well-known and widely used is the **"cloud optimized GeoTIFF"** format. Using a non-cloud format, like a JPEG, or an un-tiled TIFF will result in very poor performance, as the system will have to download the entire raster each time it needs to access a subset.

First, load your raster into the cloud storage of your choice. Once it is loaded, you will have a URI to access it with, either an "http" URI, or sometimes a URI specific to the service. (e.g., "s3://bucket/object"). To access non-public buckets, you will need to supply GDAL config options to authenticate your connection. Note that this command is *reading* from the cloud raster and *writing* to the database.

```
AWS_ACCESS_KEY_ID=xxxxxxxxxxxxxxxxxxxxxxxx \
AWS_SECRET_ACCESS_KEY=xxxxxxxxxxxxxxxxxxxxxxxx \
raster2pgsql \
-s 990000 \
-t 256x256 \
-I \
-R \
/vsis3/your.bucket.com/your_file.tif \
your_table \
| psql your_db
```

Once the table is loaded, you need to give the database permission to read from remote rasters, by setting two permissions, `postgis.enable_outdb_rasters` and `postgis.gdal_enabled_drivers`.

```
SET postgis.enable_outdb_rasters = true;
SET postgis.gdal_enabled_drivers TO 'ENABLE_ALL';
```

To make the changes sticky, set them directly on your database. You will need to re-connect to experience the new settings.

```
ALTER DATABASE your_db SET postgis.enable_outdb_rasters = true;
ALTER DATABASE your_db SET postgis.gdal_enabled_drivers TO 'ENABLE_ALL';
```

For non-public rasters, you may have to provide access keys to read from the cloud rasters. The same keys you used to write the `raster2pgsql` call can be set for use inside the database, with the `postgis.gdal_vsi_options` configuration. Note that multiple options can be set by space-separating the `key=value` pairs.

```
SET postgis.gdal_vsi_options = 'AWS_ACCESS_KEY_ID=xxxxxxxxxxxxxxxxxxxxxxxx
AWS_SECRET_ACCESS_KEY=xxxxxxxxxxxxxxxxxxxxxxxx';
```

Once you have the data loaded and permissions set you can interact with the raster table like any other raster table, using the same functions. The database will handle all the mechanics of connecting to the cloud data when it needs to read pixel data.



- `srid` Section 4.5
- `scale_x` **ST\_ScaleX**
- `scale_y` **ST\_ScaleY**
- `blocksize_x` **ST\_Width**
- `blocksize_y` **ST\_Height**
- `same_alignment` **ST\_SameAlignment**
- `regular_blocking`
- `num_bands` **ST\_NumBands**
- `pixel_types` **ST\_BandPixelType**
- `nodata_values` **ST\_BandNoDataValue**
- `out_db`



raster\_overviews

- o\_table\_catalog
- o\_table\_schema
- o\_table\_name
- o\_raster\_column
- r\_table\_catalog
- r\_table\_schema
- r\_table\_name
- r\_raster\_column
- overview\_factor



```
                                , $input_srid) ) As new_rast
FROM aerials.boston
WHERE
    ST_Intersects(rast, ST_Transform(ST_MakeEnvelope(-71.1217, 42.227, -71.1210, 42.218, 4326), 26986) );
$result = pg_query($sql);
$row = pg_fetch_row($result);
pg_free_result($result);
if ($row === false) return;
echo pg_unescape_bytea($row[0]);
?>
```

### 11.3.2 ST\_AsPNG, ST\_Union, ST\_Transform, ST\_AsGDALRaster, ASP.NET C#, ASP.NET bin

ST\_Union, ST\_Transform, ST\_AsGDALRaster, ASP.NET C#, ASP.NET bin

ASP.NET bin

ST\_Union, ST\_Transform, ST\_AsGDALRaster, ASP.NET C#, ASP.NET bin

ASP.NET bin

[http://mywebserver/test\\_raster.php?srid=2249](http://mywebserver/test_raster.php?srid=2249)

ASP.NET bin

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

```
// TestRaster.ashx
<%@ WebHandler Language="C#" Class="TestRaster" %>
using System;
```

```
using System.Data;
using System.Web;
using Npgsql;

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

    public bool IsReusable {
        get { return false; }
    }

    public byte[] GetResults(HttpContext context)
    {
        byte[] result = null;
        NpgsqlCommand command;
        string sql = null;
        int input_srid = 26986;
    try {
        using (NpgsqlConnection conn = new NpgsqlConnection(System.↵
            Configuration.ConfigurationManager.ConnectionStrings["DSN"].↵
            ConnectionString)) {
            conn.Open();

            if (context.Request["srid"] != null)
            {
                input_srid = Convert.ToInt32(context.Request["srid"]);
            }
            sql = @"SELECT ST_AsPNG(
                ST_Transform(
                    ST_AddBand(
                        ST_Union(rast,1), ARRAY[ST_Union(rast,2),ST_Union(rast,3)]
                            ,:input_srid) ) As new_rast
                FROM aerials.boston
                WHERE
                    ST_Intersects(rast,
                        ST_Transform(ST_MakeEnvelope(-71.1217, 42.227, ↵
                            -71.1210, 42.218,4326),26986) )";
            command = new NpgsqlCommand(sql, conn);
            command.Parameters.Add(new NpgsqlParameter("input_srid", input_srid));

            result = (byte[]) command.ExecuteScalar();
            conn.Close();
        }
    }
    catch (Exception ex)
    {
        result = null;
        context.Response.Write(ex.Message.Trim());
    }
    return result;
}
```

### 11.3.3 `jdbc4.jar`; `SaveQueryImage.java`; `SaveQueryImage.jar`; `Manifest.txt`; `SaveQueryImage.class`

`SaveQueryImage.java` is a Java class that uses the `jdbc4.jar` to connect to a PostgreSQL database and retrieve a raster image. The class uses the `ST_AsPNG` function to convert the raster to a PNG image.

<http://jdbc.postgresql.org/download.html>; PostgreSQL JDBC Driver; `SaveQueryImage.jar`; `Manifest.txt`; `SaveQueryImage.class`.

`SaveQueryImage.class` is the compiled version of the `SaveQueryImage.java` file. It is used to run the program.

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

`SaveQueryImage.class` is the compiled version of the `SaveQueryImage.java` file. It is used to run the program.

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

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

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

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

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

        Connection conn = null;

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

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

            ResultSet rs = sGetImg.executeQuery();

            FileOutputStream fout;
            try
```

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

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

### 11.3.4 PLPython SQL

PLPython SQL

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

```
-- PostgreSQL
-- PostgreSQL (daemon)
--
SELECT write_file(ST_AsPNG(
    ST_AsRaster(ST_Buffer(ST_Point(1,5),j*5, 'quad_segs=2'),150*j, 150*j, '8BUI',100)),
    'C:/temp/slices'|| j || '.png')
FROM generate_series(1,5) As j;

write_file
-----
C:/temp/slices1.png
```

```
C:/temp/slices2.png
C:/temp/slices3.png
C:/temp/slices4.png
C:/temp/slices5.png
```

### 11.3.5 PSQL

```
PSQL
-- <#xc548;#xd0c0;#xae5d;#xac8c;#xb3c4; PSQL <#xc740; <#xbc14;#xc774;#xb108;#xb9ac; <#xcd9c;#xb825;
<#xc791;#xc5c5;#xc5d0; <#xc27d;#xac8c; <#xc0ac;#xc6a9;#xd560; <#xc218; <#xc788;#xb294; <#xb0b4;#xc7a5;
<#xae30;#xb2a5;#xc131;#xc744; <#xac00;#xc9c0;#xace0; <#xc788;#xc9c0; <#xc54a;#xc2b5;#xb2c8;#xb2e4;.
PostgreSQL <#xc758; <#xb808;#xac70;#xc2dc; <#xb300;#xc6a9;#xb7c9; <#xac1d;#xc9c0;#xc6d0;#xc5d0;
<#xc5b4;#xb290; <#xc815;#xb3c4; <#xc21f;#xac00;#xb77d;#xb9cc; <#xc5b9;#xb294; <#xaf3c;#xc218;#xc5d0;
<#xac00;#xae5d;#xc2b5;#xb2c8;#xb2e4;. PSQL <#xc744; <#xc0ac;#xc6a9;#xd558;#xb824;#xba74;. <#xba3c;#xc8
<#xc0ac;#xc6a9;#xc790;#xb370;#xc774;#xd130;#xbca0;#xc774;#xc2a4;#xc5d0; <#xc5f0;#xacb0;#xb41c;
<#xc0ac;#xc6a9;#xc790;#xc758; PSQL <#xba85;#xb839;#xd589;#xc744; <#xc2e4;#xd589;#xd558;#xc2ed;#xc2d
<#xd30c;#xc774;#xc36c; <#xc811;#xadfc;#xbc95;#xacfc;#xb294; <#xb2ec;#xb9ac;. <#xc774; <#xc811;#xadfc;#xc
<#xc0ac;#xc6a9;#xc790;#xc758; <#xb85c;#xccec; <#xc2dc;#xc2a4;#xd15c;#xc5d0; <#xd30c;#xc77c;#xc744;
<#xc0dd;#xc131;#xd569;#xb2c8;#xb2e4;.
```

```
SELECT oid, lowrite(lo_open(oid, 131072), png) As num_bytes
FROM
  ( VALUES (lo_create(0),
    ST_AsPNG( (SELECT rast FROM aerials.boston WHERE rid=1) )
  ) ) As v(oid,png);
-- <#xb2e4;#xc74c;#xacfc; <#xbe44;#xc2b7;#xd558;#xac8c; <#xcd9c;#xb825;#xb420; <#
<#xac83;#xc785;#xb2c8;#xb2e4;. --
oid | num_bytes
-----+-----
2630819 | 74860

-- <#xadf8; <#xb2e4;#xc74c; oid<#xb97c; <#xae30;#xc5b5;#xd558;#xace0; c:/test.png <#
<#xb97c; <#xc0ac;#xc6a9;#xc790;#xc758;
-- <#xb85c;#xccec; <#xc2dc;#xc2a4;#xd15c; <#xd30c;#xc77c; <#xacbd;#xb85c; <#
<#xc704;#xc58;#xb85c; <#xb300;#xc9c0;#xd558;#xc2ed;#xc2dc;#xc624;.
\lo_export 2630819 'C:/temp/aerial_samp.png'

-- <#xc774; <#xad6c;#xbb38;#xc740; DB<#xc5d0; <#xc788;#xb294; <#xb300;#xc6a9;#xb7c9; <#
<#xac1d;#xc9c0; <#xc800;#xc7a5;#xc18c;#xc5d0;#xc11c; <#xd30c;#xc77c;#xc744; <#
<#xc0ad;#xc81c;#xd569;#xb2c8;#xb2e4;.
```

```
SELECT lo_unlink(2630819);
```

## Chapter 12

# Creating a Raster Table

PostGIS 3.3.6dev provides the `raster` data type, which is used to store raster data in a database. The `raster` data type is a complex type that contains a lot of information about the raster, such as its dimensions, scale, and SRID. The `raster` data type is used to store raster data in a database, and it is used to create a raster table. The `raster` data type is used to store raster data in a database, and it is used to create a raster table.

```
raster
```

Section 11.1

CREATE TABLE dummy\_rast(rid integer, rast raster);

```
INSERT INTO dummy_rast(rid, rast)
VALUES (1,
('01' -- little endian (uint8 ndr)
||
'0000' -- version (uint16 0)
||
'0000' -- nBands (uint16 0)
||
'000000000000000040' -- scaleX (float64 2)
||
'00000000000000840' -- scaleY (float64 3)
||
'000000000000E03F' -- ipX (float64 0.5)
||
'000000000000E03F' -- ipY (float64 0.5)
||
'0000000000000000' -- skewX (float64 0)
||
'0000000000000000' -- skewY (float64 0)
||
'00000000' -- SRID (int32 0)
||
'0A00' -- width (uint16 10)
||
'1400' -- height (uint16 20)
)::raster
),
```



```
initialvalue double precision initialvalue double precision
```

```
nodataval double precision nodataval double precision NODATA NULL
```

```
ST_AddBand
```

### 12.1.3 rastbandarg

```
rastbandarg — rastbandarg
```

```
ST_ConvexHull
```

```
rast raster rast raster
```

```
nband integer nband integer
```

```
ST_ConvexHull
```

```
ST_ConvexHull
```

### 12.1.4 raster

```
raster — raster
```

```
ST_ConvexHull
```

raster is a spatial data type used to represent raster data such as those imported from JPEGs, TIFFs, PNGs, digital elevation models. Each raster has 1 or more bands each having a set of pixel values. Rasters can be georeferenced.

#### Note



```
GDAL ST_ConvexHull
```



```
1BB', NULL)::reclassarg;
```

```
SELECT ROW(1, '0-100]:0, (100-255:1', '1BB', NULL)::reclassarg;
```

```
ST_Reclass
```

```
ST_Reclass
```

## 12.1.6 summarystats

**summarystats** — **ST\_SummaryStats**, **ST\_SummaryStatsAgg**

```
ST_SummaryStats, ST_SummaryStatsAgg
```

**ST\_SummaryStats**, **ST\_SummaryStatsAgg**

**count integer**

**sum double precision**

**mean double precision**

**stdev double precision**

**min double precision**

**max double precision**

```
ST_SummaryStats, ST_SummaryStatsAgg
```

```
ST_SummaryStats, ST_SummaryStatsAgg
```

## 12.1.7 unionarg

**unionarg** — **ST\_Union**, **ST\_UnionAgg**

```
ST_Union, ST_UnionAgg
```

**ST\_Union**, **ST\_UnionAgg**

**nband integer**

**uniontype text** **ST\_Union**

**ST\_Union**

**ST\_Union**

## 12.2 AddRasterConstraints

### 12.2.1 AddRasterConstraints

**AddRasterConstraints** — Adds raster constraints to a loaded raster table for a specific column that constrains spatial ref, scaling, blocksize, alignment, bands, band type and a flag to denote if raster column is regularly blocked. The table must be loaded with data for the constraints to be inferred. Returns true if the constraint setting was accomplished and issues a notice otherwise.

#### Synopsis

```
boolean AddRasterConstraints(name rasttable, name rastcolumn, boolean srid, boolean scale_x, boolean scale_y, boolean
blocksize_x, boolean blocksize_y, boolean same_alignment, boolean regular_blocking, boolean num_bands=true, boolean
pixel_types=true, boolean no_data_values=true, boolean out_db=true, boolean extent=true);
boolean AddRasterConstraints(name rasttable, name rastcolumn, text[] VARIADIC constraints);
boolean AddRasterConstraints(name rastschema, name rasttable, name rastcolumn, text[] VARIADIC constraints);
boolean AddRasterConstraints(name rastschema, name rasttable, name rastcolumn, boolean srid=true, boolean scale_x=true,
boolean scale_y=true, boolean blocksize_x=true, boolean blocksize_y=true, boolean same_alignment=true, boolean regular_blocking=true,
boolean num_bands=true, boolean pixel_types=true, boolean no_data_values=true, boolean out_db=true, boolean extent=true);
```

**Parameters**

```
raster_columns name of the raster column;
rastschema name of the raster schema;
srid SRID of the raster column;
scale_x, scale_y scaling factors for the raster column;
blocksize_x, blocksize_y block sizes for the raster column;
same_alignment, regular_blocking flags to denote if the raster column is regularly blocked;
num_bands number of bands for the raster column;
pixel_types, no_data_values flags to denote if the raster column has pixel types and no data values;
out_db flag to denote if the raster column is stored in a different database;
extent flag to denote if the raster column has an extent;
constraints list of constraints for the raster column.
```

```
raster2pgsql name of the raster column;
srid SRID of the raster column;
scale_x, scale_y scaling factors for the raster column;
blocksize_x, blocksize_y block sizes for the raster column;
same_alignment, regular_blocking flags to denote if the raster column is regularly blocked;
num_bands number of bands for the raster column;
pixel_types, no_data_values flags to denote if the raster column has pixel types and no data values;
out_db flag to denote if the raster column is stored in a different database;
extent flag to denote if the raster column has an extent;
constraints list of constraints for the raster column.
```

```
SPATIAL_REF_SYS name of the spatial reference system;
srid SRID of the spatial reference system;
scale_x, scale_y scaling factors for the spatial reference system;
blocksize_x, blocksize_y block sizes for the spatial reference system;
same_alignment, regular_blocking flags to denote if the spatial reference system is regularly blocked;
num_bands number of bands for the spatial reference system;
pixel_types, no_data_values flags to denote if the spatial reference system has pixel types and no data values;
out_db flag to denote if the spatial reference system is stored in a different database;
extent flag to denote if the spatial reference system has an extent;
constraints list of constraints for the spatial reference system.
```

- blocksize**: X, Y
- blocksize\_x**: X
- blocksize\_y**: Y
- extent**: bounding box of the raster column
- num\_bands**: number of bands for the raster column
- pixel\_types**: list of pixel types for the raster column



**DropRasterConstraints**

```
CREATE TABLE public.myrasters2(rid SERIAL primary key, rast raster);
INSERT INTO myrasters2(rast)
SELECT ST_AddBand(ST_MakeEmptyRaster(1000, 1000, 0.3, -0.3, 2, 2, 0, 0,4326), 1, '8BSI':: ←
    text, -129, NULL);

SELECT AddRasterConstraints('public'::name, 'myrasters2'::name, 'rast'::name, ' ←
    regular_blocking', 'blocksize');
-- &#xc548;&#xb0b4;&#xbb38; &#xcd9c;&#xb825; --
NOTICE: Adding regular blocking constraint
NOTICE: Adding blocksize-X constraint
NOTICE: Adding blocksize-Y constraint
```

**DropRasterConstraints**

Section 11.2.1, [ST\\_AddBand](#), [ST\\_MakeEmptyRaster](#), [DropRasterConstraints](#), [ST\\_BandPixelType](#), [ST\\_SRID](#)

## 12.2.2 DropRasterConstraints

**DropRasterConstraints** — Drop raster constraints from a raster table. The function takes the following arguments:

**DropRasterConstraints**(name rasttable, name rastcolumn, boolean srid, boolean scale\_x, boolean scale\_y, boolean blocksize\_x, boolean blocksize\_y, boolean same\_alignment, boolean regular\_blocking, boolean num\_bands=true, boolean pixel\_types=true, boolean nodata\_values=true, boolean out\_db=true, boolean extent=true);

**DropRasterConstraints**(name rastschema, name rasttable, name rastcolumn, boolean srid=true, boolean scale\_x=true, boolean scale\_y=true, boolean blocksize\_x=true, boolean blocksize\_y=true, boolean same\_alignment=true, boolean regular\_blocking=false, boolean num\_bands=true, boolean pixel\_types=true, boolean nodata\_values=true, boolean out\_db=true, boolean extent=true);

**DropRasterConstraints**(name rastschema, name rasttable, name rastcolumn, text[] constraints);

### Synopsis

**DropRasterConstraints**(name rasttable, name rastcolumn, boolean srid, boolean scale\_x, boolean scale\_y, boolean blocksize\_x, boolean blocksize\_y, boolean same\_alignment, boolean regular\_blocking, boolean num\_bands=true, boolean pixel\_types=true, boolean nodata\_values=true, boolean out\_db=true, boolean extent=true);

**DropRasterConstraints**(name rastschema, name rasttable, name rastcolumn, boolean srid=true, boolean scale\_x=true, boolean scale\_y=true, boolean blocksize\_x=true, boolean blocksize\_y=true, boolean same\_alignment=true, boolean regular\_blocking=false, boolean num\_bands=true, boolean pixel\_types=true, boolean nodata\_values=true, boolean out\_db=true, boolean extent=true);

**DropRasterConstraints**(name rastschema, name rasttable, name rastcolumn, text[] constraints);

**DropRasterConstraints**

**AddRasterConstraints** Add raster constraints to a raster table. The function takes the following arguments:

**AddRasterConstraints**(name rasttable, name rastcolumn, boolean srid, boolean scale\_x, boolean scale\_y, boolean blocksize\_x, boolean blocksize\_y, boolean same\_alignment, boolean regular\_blocking, boolean num\_bands=true, boolean pixel\_types=true, boolean nodata\_values=true, boolean out\_db=true, boolean extent=true, text[] constraints);

**AddRasterConstraints**(name rastschema, name rasttable, name rastcolumn, boolean srid=true, boolean scale\_x=true, boolean scale\_y=true, boolean blocksize\_x=true, boolean blocksize\_y=true, boolean same\_alignment=true, boolean regular\_blocking=false, boolean num\_bands=true, boolean pixel\_types=true, boolean nodata\_values=true, boolean out\_db=true, boolean extent=true, text[] constraints);

**AddRasterConstraints**(name rastschema, name rasttable, name rastcolumn, text[] constraints);

**DropRasterConstraints**

```
DROP TABLE mytable
```

**DropRasterConstraints**



```
ovschema; refschema;
search_path;
&#xd14c;&#xc774;&#xbe14;&#xc744; &#xc774;&#xc6a9;&#xd560; &#xac83;&#xc785;&#xb2c8;&#xb2e4;.
```

```
2.0.0 &#xbc84;&#xc804;&#xbd80;&#xd130; &#xc0ac;&#xc6a9;&#xd560; &#xc218; &#xc788;&#xc2b5;&#xb2c8;&#xb2e4;.
```

### &#xc608;&#xc2dc;

```
CREATE TABLE res1 AS SELECT
ST_AddBand(
  ST_MakeEmptyRaster(1000, 1000, 0, 0, 2),
  1, '8BSI'::text, -129, NULL
) r1;

CREATE TABLE res2 AS SELECT
ST_AddBand(
  ST_MakeEmptyRaster(500, 500, 0, 0, 4),
  1, '8BSI'::text, -129, NULL
) r2;

SELECT AddOverviewConstraints('res2', 'r2', 'res1', 'r1', 2);

-- raster_overviews &#xbd0;&#xc5d0; &#xc815;&#xd655;&#xd788; ↔
&#xb4f1;&#xb85d;&#xb410;&#xb294;&#xc9c0; &#xd655;&#xc778; --
SELECT o_table_name ot, o_raster_column oc,
       r_table_name rt, r_raster_column rc,
       overview_factor f
FROM raster_overviews WHERE o_table_name = 'res2';
  ot | oc | rt | rc | f
-----+-----+-----+-----+---
  res2 | r2 | res1 | r1 | 2
(1 row)
```

### &#xcc38;&#xace0;

Section [11.2.2, DropOverviewConstraints, ST\\_CreateOverview, AddRasterConstraints](#)

## 12.2.4 DropOverviewConstraints

DropOverviewConstraints — &#xb610; &#xb2e4;&#xb978; &#xb798;&#xc2a4;&#xd130; &#xc5f4;&#xc758; &#xbbf8;&#xb9ac;&#xb798;&#xc2a4;&#xd130; &#xc5f4;&#xc744; &#xd0dc;&#xadf8; &#xd574;&#xc81c;&#xd569;&#xb2c8;&#xb2e4;.

### Synopsis

```
boolean DropOverviewConstraints(name ovschema, name ovtbl, name ovcolumn);
boolean DropOverviewConstraints(name ovtbl, name ovcolumn);
```

### &#xc124;&#xba85;

```
raster_overviews &#xb798;&#xc2a4;&#xd130; &#xc74;&#xd0c8;&#xb85c;&#xadf8;&#xc5d0; &#xb610; &#xb2e4;&#xb978;
&#xb798;&#xc2a4;&#xd130; &#xc5f4;&#xc758; &#xbbf8;&#xb9ac;&#xbcf4;&#xae30;&#xb97c; &#xd45c;&#xcd9c;&#xd558;&#xb370;
&#xc4f0;&#xc774;&#xb294; &#xb798;&#xc2a4;&#xd130; &#xc5f4;&#xc758; &#xc81c;&#xc57d;&#xc870;&#xac74;&#xc81c;
&#xac70;&#xd569;&#xb2c8;&#xb2e4;.
```

```
ovs schema, search_path
&#x2013; &#xc2a4; &#xc94; &#xd558; &#xba74; &#xc11c; &#xc98; &#xc74c; &#xbc1c; &#xacac; &#xd55c; &#xd14c; &#xc774; &#xc774; &#xc6a9; &#xd560; &#xac83; &#xc785; &#xb2c8; &#xb2e4;.
```

```
2.0.0 &#xbc84; &#xc804; &#xbd80; &#xd130; &#xc0ac; &#xc6a9; &#xd560; &#xc218; &#xc788; &#xc2b5; &#xb2c8; &#xb2e4;.
```

**&#xc38; &#xace0;**

Section [11.2.2, AddOverviewConstraints, DropRasterConstraints](#)

## 12.2.5 PostGIS\_GDAL\_Version

PostGIS\_GDAL\_Version — PostGIS &#xc00; &#xc774; &#xc6a9; &#xd558; &#xace0; &#xc788; &#xb294; GDAL &#xb77c; &#xc774; &#xbc84; &#xc804; &#xc744; &#xbc18; &#xd658; &#xd569; &#xb2c8; &#xb2e4;.

### Synopsis

```
text PostGIS_GDAL_Version();
```

**&#xc124; &#xba85;**

```
PostGIS &#xc00; &#xc774; &#xc6a9; &#xd558; &#xace0; &#xc788; &#xb294; GDAL &#xb77c; &#xc774; &#xbe0c; &#xb7ec; &#xb9a
&#xbc84; &#xc804; &#xc744; &#xbc18; &#xd658; &#xd569; &#xb2c8; &#xb2e4;. &#xb610; GDAL &#xc774; &#xc790; &#xc4;
&#xb370; &#xc774; &#xd130; &#xd30c; &#xc77c; &#xc744; &#xc3e; &#xc744; &#xc218; &#xc788; &#xb294; &#xc9c0; &#xb3c4;
&#xd655; &#xc778; &#xd574; &#xc11c; &#bcf4; &#xace0; &#xd560; &#xac83; &#xc785; &#xb2c8; &#xb2e4;.
```

**&#xc608; &#xc2dc;**

```
SELECT PostGIS_GDAL_Version();
       postgis_gdal_version
-----
GDAL 1.11dev, released 2013/04/13
```

**&#xc38; &#xace0;**

[postgis.gdal\\_datapath](#)

## 12.2.6 PostGIS\_Raster\_Lib\_Build\_Date

PostGIS\_Raster\_Lib\_Build\_Date — &#xc804; &#xc4; &#xb798; &#xc2a4; &#xd130; &#xb77c; &#xc774; &#xbe0c; &#xb7ec; &#xb9a
&#xbe4c; &#xb4dc; &#xb0a0; &#xc9dc; &#xb97c; &#xbc18; &#xd658; &#xd569; &#xb2c8; &#xb2e4;.

### Synopsis

```
text PostGIS_Raster_Lib_Build_Date();
```

**&#xc124; &#xba85;**

```
&#xb798; &#xc2a4; &#xd130; &#xc758; &#xbe4c; &#xb4dc; &#xb0a0; &#xc9dc; &#xb97c; &#bcf4; &#xace0; &#xd569; &#xb2c8; &#xb2e4;.
```

**PostGIS\_Raster\_Lib\_Build\_Date();**

```
SELECT PostGIS_Raster_Lib_Build_Date();
postgis_raster_lib_build_date
-----
2010-04-28 21:15:10
```

**PostGIS\_Raster\_Lib\_Version**[PostGIS\\_Raster\\_Lib\\_Version](#)**12.2.7 PostGIS\_Raster\_Lib\_Version**

**PostGIS\_Raster\_Lib\_Version** — Returns the version of the PostGIS Raster Library. The version string is in the format `major.minor.patch`.

**Synopsis**

```
text PostGIS_Raster_Lib_Version();
```

**PostGIS\_Raster\_Lib\_Version();**

```
PostGIS_Raster_Lib_Version()
-----
2.0.0
```

**PostGIS\_Raster\_Lib\_Version**

```
SELECT PostGIS_Raster_Lib_Version();
postgis_raster_lib_version
-----
2.0.0
```

**PostGIS\_Lib\_Version**[PostGIS\\_Lib\\_Version](#)**12.2.8 ST\_GDALDrivers**

**ST\_GDALDrivers** — Returns a list of raster formats supported by PostGIS through GDAL. Only those formats with `can_write=True` can be used by `ST_AsGDALRaster`.

**Synopsis**

```
setof record ST_GDALDrivers(integer OUT idx, text OUT short_name, text OUT long_name, text OUT can_read, text OUT can_write, text OUT create_options);
```

### ST\_AsGDALRaster

Returns a list of raster formats short\_name, long\_name and creator options of each format supported by GDAL. Use the short\_name as input in the format parameter of **ST\_AsGDALRaster**. Options vary depending on what drivers your libgdal was compiled with. create\_options returns an xml formatted set of CreationOptionList/Option consisting of name and optional type, description and set of VALUE for each creator option for the specific driver.

Changed: 2.5.0 - add can\_read and can\_write columns.

2.0.6, 2.1.3 - GUC gdal\_enabled\_drivers, ST\_AsGDALRaster: 2.0.6, 2.1.3 - GUC gdal\_enabled\_drivers, ST\_AsGDALRaster: 2.0.6, 2.1.3 - GUC gdal\_enabled\_drivers, ST\_AsGDALRaster: 2.0.6, 2.1.3 - GUC gdal\_enabled\_drivers

2.0.0 - GDAL 1.6.0

### CREATE\_METADATA

```
SET postgis.gdal_enabled_drivers = 'ENABLE_ALL';
SELECT short_name, long_name, can_write
FROM st_gdaldrivers()
ORDER BY short_name;
```

| short_name | long_name                               | can_write |
|------------|---|-----------|
| AAIGrid    | Arc/Info ASCII Grid                     | t         |
| ACE2       | ACE2                                    | f         |
| ADRG       | ARC Digitized Raster Graphics           | f         |
| AIG        | Arc/Info Binary Grid                    | f         |
| AirSAR     | AirSAR Polarimetric Image               | f         |
| ARG        | Azavea Raster Grid format               | t         |
| BAG        | Bathymetry Attributed Grid              | f         |
| BIGGIF     | Graphics Interchange Format (.gif)      | f         |
| BLX        | Magellan topo (.blx)                    | t         |
| BMP        | MS Windows Device Independent Bitmap    | f         |
| BSB        | Maptech BSB Nautical Charts             | f         |
| PAux       | PCI .aux Labelled                       | f         |
| PCIDSK     | PCIDSK Database File                    | f         |
| PCRaster   | PCRaster Raster File                    | f         |
| PDF        | Geospatial PDF                          | f         |
| PDS        | NASA Planetary Data System              | f         |
| PDS4       | NASA Planetary Data System 4            | t         |
| PLMOAIC    | Planet Labs Mosaics API                 | f         |
| PLSCENES   | Planet Labs Scenes API                  | f         |
| PNG        | Portable Network Graphics               | t         |
| PNM        | Portable Pixmap Format (netpbm)         | f         |
| PRF        | Racurs PHOTOMOD PRF                     | f         |
| R          | R Object Data Store                     | t         |
| Rasterlite | Rasterlite                              | t         |
| RDA        | DigitalGlobe Raster Data Access driver  | f         |
| RIK        | Swedish Grid RIK (.rik)                 | f         |
| RMF        | Raster Matrix Format                    | f         |
| ROI_PAC    | ROI_PAC raster                          | f         |
| RPFTOC     | Raster Product Format TOC format        | f         |
| RRASTER    | R Raster                                | f         |
| RS2        | RadarSat 2 XML Product                  | f         |
| RST        | Idrisi Raster A.1                       | t         |
| SAFE       | Sentinel-1 SAR SAFE Product             | f         |
| SAGA       | SAGA GIS Binary Grid (.sdat, .sg-grd-z) | t         |
| SAR_CEOS   | CEOS SAR Image                          | f         |

|           |                                     |   |
|-----------|-------------------------------------|---|
| SDTS      | SDTS Raster                         | f |
| SENTINEL2 | Sentinel 2                          | f |
| SGI       | SGI Image File Format 1.0           | f |
| SNODAS    | Snow Data Assimilation System       | f |
| SRP       | Standard Raster Product (ASRP/USRP) | f |
| SRTMHGT   | SRTMHGT File Format                 | t |
| Terragen  | Terragen heightfield                | f |
| TIL       | EarthWatch .TIL                     | f |
| TSX       | TerraSAR-X Product                  | f |
| USGSDEM   | USGS Optional ASCII DEM (and CDED)  | t |
| VICAR     | MIPL VICAR file                     | f |
| VRT       | Virtual Raster                      | t |
| WCS       | OGC Web Coverage Service            | f |
| WMS       | OGC Web Map Service                 | t |
| WMTS      | OGC Web Map Tile Service            | t |
| XPM       | X11 PixMap Format                   | t |
| XYZ       | ASCII Gridded XYZ                   | t |
| ZMap      | ZMap Plus Grid                      | t |

**GDAL Options for JPEG**

```
-- JPEG; XML; ST_AsGDALRaster;
-- &#xc00; &#xacf5; &#xb418; &#xc9c0; &#xc54a; &#xc740; XML;
SELECT create_options
FROM st_gdaldrivers()
WHERE short_name = 'JPEG';
```

| oname              | otype   | descrip  |
|--------------------|---------|--|
| PROGRESSIVE        | boolean | whether to generate a progressive JPEG   |
| QUALITY            | int     | good=100, bad=0, default=75  |
| WORLDFILE          | boolean | whether to generate a worldfile  |
| INTERNAL_MASK      | boolean | whether to generate a validity mask  |
| COMMENT            | string  | Comment  |
| SOURCE_ICC_PROFILE | string  | ICC profile encoded in Base64  |
| EXIF_THUMBNAIL     | boolean | whether to generate an EXIF thumbnail(overview).<br>By default its max dimension will be 128 |
| THUMBNAIL_WIDTH    | int     | Forced thumbnail width   |
| THUMBNAIL_HEIGHT   | int     | Forced thumbnail height  |

(9 rows)

```
-- GeoTiff; XML;
SELECT create_options
FROM st_gdaldrivers()
WHERE short_name = 'GTiff';

<CreationOptionList>
  <Option name="COMPRESS" type="string-select">
    <Value
>NONE</Value>
    <Value
>LZW</Value>
    <Value
```

```
>PACKBITS</Value>
  <Value>
>JPEG</Value>
  <Value>
>CCITTRLE</Value>
  <Value>
>CCITTFAX3</Value>
  <Value>
>CCITTFAX4</Value>
  <Value>
>DEFLATE</Value>
  </Option>
  <Option name="PREDICTOR" type="int" description="Predictor Type"/>
  <Option name="JPEG_QUALITY" type="int" description="JPEG quality 1-100" default="75"/>
  <Option name="ZLEVEL" type="int" description="DEFLATE compression level 1-9" default ←
    ="6"/>
  <Option name="NBITS" type="int" description="BITS for sub-byte files (1-7), sub-uint16 ←
    (9-15), sub-uint32 (17-31)"/>
  <Option name="INTERLEAVE" type="string-select" default="PIXEL">
    <Value>
>BAND</Value>
  <Value>
>PIXEL</Value>
  </Option>
  <Option name="TILED" type="boolean" description="Switch to tiled format"/>
  <Option name="TFW" type="boolean" description="Write out world file"/>
  <Option name="RPB" type="boolean" description="Write out .RPB (RPC) file"/>
  <Option name="BLOCKXSIZE" type="int" description="Tile Width"/>
  <Option name="BLOCKYSIZE" type="int" description="Tile/Strip Height"/>
  <Option name="PHOTOMETRIC" type="string-select">
    <Value>
>MINISBLACK</Value>
  <Value>
>MINISWHITE</Value>
  <Value>
>PALETTE</Value>
  <Value>
>RGB</Value>
  <Value>
>CMYK</Value>
  <Value>
>YCBCR</Value>
  <Value>
>CIELAB</Value>
  <Value>
>ICCLAB</Value>
  <Value>
>ITULAB</Value>
  </Option>
  <Option name="SPARSE_OK" type="boolean" description="Can newly created files have ←
    missing blocks?" default="FALSE"/>
  <Option name="ALPHA" type="boolean" description="Mark first extrasample as being alpha ←
    "/>
  <Option name="PROFILE" type="string-select" default="GDALGeoTIFF">
    <Value>
>GDALGeoTIFF</Value>
  <Value>
>GeoTIFF</Value>
  <Value>
>BASELINE</Value>
  </Option>
  <Option name="PIXELTYPE" type="string-select">
```

```

    <Value
>DEFAULT</Value>
    <Value
>SIGNEDBYTE</Value>
  </Option>
  <Option name="BIGTIFF" type="string-select" description="Force creation of BigTIFF file ←
">
    <Value
>YES</Value>
    <Value
>NO</Value>
    <Value
>IF_NEEDED</Value>
    <Value
>IF_SAFER</Value>
  </Option>
  <Option name="ENDIANNESS" type="string-select" default="NATIVE" description="Force ←
endianness of created file. For DEBUG purpose mostly">
    <Value
>NATIVE</Value>
    <Value
>INVERTED</Value>
    <Value
>LITTLE</Value>
    <Value
>BIG</Value>
  </Option>
  <Option name="COPY_SRC_OVERVIEWS" type="boolean" default="NO" description="Force copy ←
of overviews of source dataset (CreateCopy())"/>
</CreationOptionList
>

```

```

-- GeoFiff &#xc5d0; &#xb300;&#xd55c; &#xc0dd;&#xc131; &#xc635;&#xc158; XML &#xc5f4;&#xc744; ←
&#xd45c;&#xb85c; &#xcd9c;&#xb825; --
SELECT (xpath('@name', g.opt))[1]::text As oname,
       (xpath('@type', g.opt))[1]::text As otype,
       (xpath('@description', g.opt))[1]::text As descrip,
       array_to_string(xpath('Value/text()', g.opt), ', ') As vals
FROM (SELECT unnest(xpath('/CreationOptionList/Option', create_options::xml)) As opt
FROM st_gdaldrivers()
WHERE short_name = 'GTiff') As g;

```

| oname        | otype         | descrip  | vals         |
|--------------|---------------|--|--------------|
| COMPRESS     | string-select |  | NONE, LZW, ← |
| PREDICTOR    | int           | Predictor Type   |              |
| JPEG_QUALITY | int           | JPEG quality 1-100   |              |
| ZLEVEL       | int           | DEFLATE compression level 1-9  |              |
| NBITS        | int           | BITS for sub-byte files (1-7), sub-uint16 (9-15), sub-uint32 (17-31) |              |
| INTERLEAVE   | string-select |  | BAND, PIXEL  |
| TILED        | boolean       | Switch to tiled format   |              |
| TFW          | boolean       | Write out world file   |              |

|                    |               |   |
|--------------------|---------------|---|
| RPB                | boolean       | Write out .RPB (RPC) file ↔                                   |
| BLOCKXSIZE         | int           | Tile Width ↔  |
| BLOCKYSIZE         | int           | Tile/Strip Height ↔   |
| PHOTOMETRIC        | string-select | ↔   |
|                    |               | MINISBLACK, ↔   |
|                    |               | MINISWHITE, PALETTE, RGB, CMYK, YCBCR, CIELAB, ICCLAB, ITULAB |
| SPARSE_OK          | boolean       | Can newly created files have missing blocks? ↔                |
| ALPHA              | boolean       | Mark first extrasample as being alpha ↔                       |
| PROFILE            | string-select | ↔   |
|                    |               | GDALGeoTIFF, ↔  |
|                    |               | GeoTIFF, BASELINE   |
| PIXELTYPE          | string-select | ↔   |
|                    |               | DEFAULT, ↔  |
|                    |               | SIGNEDBYTE  |
| BIGTIFF            | string-select | Force creation of BigTIFF file ↔                              |
|                    |               | YES, NO, IF_NEEDED, IF_SAFER                                  |
| ENDIANNESS         | string-select | Force endianness of created file. For DEBUG purpose ↔         |
|                    |               | mostly  |
|                    |               | NATIVE, INVERTED, LITTLE, BIG                                 |
| COPY_SRC_OVERVIEWS | boolean       | Force copy of overviews of source dataset (CreateCopy ↔       |
|                    |               | ())   |
| (19 rows)          |               |   |

**&#xc38;&#xace0;**

[ST\\_AsGDALRaster](#), [ST\\_SRID](#), [postgis.gdal\\_enabled\\_drivers](#)

## 12.2.9 ST\_Count

**ST\_Count** — Generates a set of vector contours from the provided raster band, using the [GDAL contouring algorithm](#).

### Synopsis

raster **ST\_TPI**(raster rast, integer nband, raster customextent, text pixeltype="32BF" , boolean interpolate\_nodata=FALSE );

**&#xc124;&#xba85;**

Generates a set of vector contours from the provided raster band, using the [GDAL contouring algorithm](#).

When the `fixed_levels` parameter is a non-empty array, the `level_interval` and `level_base` parameters are ignored.

The `polygonize` parameter currently has no effect. Use the [ST\\_Polygonize](#) function to convert contours into polygons.

Return values are a set of records with the following attributes:

**geomval** The geometry of the contour line.

**id** A unique identifier given to the contour line by GDAL.

**ST\_Value** The raster value the line represents. For an elevation DEM input, this would be the elevation of the output contour.

2.2.0 &#xbc84;&#xc804;&#xbd80;&#xd130; &#xc0ac;&#xc6a9;&#xd560; &#xc218; &#xc788;&#xc2b5;&#xb2c8;&#xb2e4;.

**SQL**

```
WITH c AS (
SELECT (ST_Contour(rast, 1, fixed_levels => ARRAY[100.0, 200.0, 300.0])).*
FROM dem_grid WHERE rid = 1
)
SELECT st_astext(geom), id, value
FROM c;
```

**ST\_MakeEmptyRaster****ST\_MakeEmptyRaster****12.2.10 ST\_MakeEmptyRaster**

**ST\_MakeEmptyRaster** — Interpolates a gridded surface based on an input set of 3-d points, using the X- and Y-values to position the points on the grid and the Z-value of the points as the surface elevation.

**Synopsis**

bytea **ST\_AsGDALRaster**(raster rast, text format, text[] options=NULL, integer srid=sameassource);

**Description**

Interpolates a gridded surface based on an input set of 3-d points, using the X- and Y-values to position the points on the grid and the Z-value of the points as the surface elevation. There are five interpolation algorithms available: inverse distance, inverse distance nearest-neighbor, moving average, nearest neighbor, and linear interpolation. See the [gdal\\_grid documentation](#) for more details on the algorithms and their parameters. For more information on how interpolations are calculated, see the [GDAL grid tutorial](#).

Input parameters are:

**input\_points** The points to drive the interpolation. Any geometry with Z-values is acceptable, all points in the input will be used.

**algorithm\_options** A string defining the algorithm and algorithm options, in the format used by [gdal\\_grid](#). For example, for an inverse-distance interpolation with a smoothing of 2, you would use "invdist:smoothing=2.0"

**template** A raster template to drive the geometry of the output raster. The width, height, pixel size, spatial extent and pixel type will be read from this template.

**template\_band\_num** By default the first band in the template raster is used to drive the output raster, but that can be adjusted with this parameter.

2.2.0

**SQL**

```
SELECT ST_InterpolateRaster(
  'MULTIPOINT(10.5 9.5 1000, 11.5 8.5 1000, 10.5 8.5 500, 11.5 9.5 500)::geometry,
  'invdist:smoothing:2.0',
  ST_AddBand(ST_MakeEmptyRaster(200, 400, 10, 10, 0.01, -0.005, 0, 0), '16BSI')
)
```



**ST\_AddBand**

ST\_AddBand(raster rast, integer index, text pixeltype, double precision initialvalue=0, double precision nodataval=NULL);  
 ST\_AddBand(raster rast, integer index, text pixeltype, double precision initialvalue=0, double precision nodataval=NULL);  
 ST\_AddBand(raster rast, text pixeltype, double precision initialvalue=0, double precision nodataval=NULL);  
 ST\_AddBand(raster torast, raster fromrast, integer fromband=1, integer torastindex=at\_end);  
 ST\_AddBand(raster torast, raster[] fromrasts, integer fromband=1, integer torastindex=at\_end);  
 ST\_AddBand(raster rast, integer index, text outdbfile, integer[] outdbindex, double precision nodataval=NULL);  
 ST\_AddBand(raster rast, text outdbfile, integer[] outdbindex, integer index=at\_end, double precision nodataval=NULL);

raster\_overviews  
 ST\_AddBand(raster rast, integer index, text pixeltype, double precision initialvalue=0, double precision nodataval=NULL);  
 ST\_AddBand(raster rast, integer index, text pixeltype, double precision initialvalue=0, double precision nodataval=NULL);  
 ST\_AddBand(raster rast, text pixeltype, double precision initialvalue=0, double precision nodataval=NULL);  
 ST\_AddBand(raster torast, raster fromrast, integer fromband=1, integer torastindex=at\_end);  
 ST\_AddBand(raster torast, raster[] fromrasts, integer fromband=1, integer torastindex=at\_end);  
 ST\_AddBand(raster rast, integer index, text outdbfile, integer[] outdbindex, double precision nodataval=NULL);  
 ST\_AddBand(raster rast, text outdbfile, integer[] outdbindex, integer index=at\_end, double precision nodataval=NULL);

ST\_AddBand(raster rast, integer index, text pixeltype, double precision initialvalue=0, double precision nodataval=NULL);  
 ST\_AddBand(raster rast, integer index, text pixeltype, double precision initialvalue=0, double precision nodataval=NULL);  
 ST\_AddBand(raster rast, text pixeltype, double precision initialvalue=0, double precision nodataval=NULL);  
 ST\_AddBand(raster torast, raster fromrast, integer fromband=1, integer torastindex=at\_end);  
 ST\_AddBand(raster torast, raster[] fromrasts, integer fromband=1, integer torastindex=at\_end);  
 ST\_AddBand(raster rast, integer index, text outdbfile, integer[] outdbindex, double precision nodataval=NULL);  
 ST\_AddBand(raster rast, text outdbfile, integer[] outdbindex, integer index=at\_end, double precision nodataval=NULL);

2.2.0 ST\_AddBand(raster rast, integer index, text pixeltype, double precision initialvalue=0, double precision nodataval=NULL);  
 ST\_AddBand(raster rast, integer index, text pixeltype, double precision initialvalue=0, double precision nodataval=NULL);  
 ST\_AddBand(raster rast, text pixeltype, double precision initialvalue=0, double precision nodataval=NULL);  
 ST\_AddBand(raster torast, raster fromrast, integer fromband=1, integer torastindex=at\_end);  
 ST\_AddBand(raster torast, raster[] fromrasts, integer fromband=1, integer torastindex=at\_end);  
 ST\_AddBand(raster rast, integer index, text outdbfile, integer[] outdbindex, double precision nodataval=NULL);  
 ST\_AddBand(raster rast, text outdbfile, integer[] outdbindex, integer index=at\_end, double precision nodataval=NULL);

**ST\_AddBand**

Output to generally better quality but slower to product format

```
SELECT ST_CreateOverview('mydata.mytable'::regclass, 'rast', 2, 'Lanczos');
```

Output to faster to process default nearest neighbor

```
SELECT ST_CreateOverview('mydata.mytable'::regclass, 'rast', 2);
```

**ST\_AddBand**

[ST\\_Retile](#), [AddOverviewConstraints](#), [AddRasterConstraints](#), Section 11.2.2

**12.3 ST\_AddBand****12.3.1 ST\_AddBand**

ST\_AddBand(raster rast, integer index, text pixeltype, double precision initialvalue=0, double precision nodataval=NULL);  
 ST\_AddBand(raster rast, integer index, text pixeltype, double precision initialvalue=0, double precision nodataval=NULL);  
 ST\_AddBand(raster rast, text pixeltype, double precision initialvalue=0, double precision nodataval=NULL);  
 ST\_AddBand(raster torast, raster fromrast, integer fromband=1, integer torastindex=at\_end);  
 ST\_AddBand(raster torast, raster[] fromrasts, integer fromband=1, integer torastindex=at\_end);  
 ST\_AddBand(raster rast, integer index, text outdbfile, integer[] outdbindex, double precision nodataval=NULL);  
 ST\_AddBand(raster rast, text outdbfile, integer[] outdbindex, integer index=at\_end, double precision nodataval=NULL);

**Synopsis**

- (1) raster **ST\_AddBand**(raster rast, addbandarg[] addbandargset);
- (2) raster **ST\_AddBand**(raster rast, integer index, text pixeltype, double precision initialvalue=0, double precision nodataval=NULL);
- (3) raster **ST\_AddBand**(raster rast, text pixeltype, double precision initialvalue=0, double precision nodataval=NULL);
- (4) raster **ST\_AddBand**(raster torast, raster fromrast, integer fromband=1, integer torastindex=at\_end);
- (5) raster **ST\_AddBand**(raster torast, raster[] fromrasts, integer fromband=1, integer torastindex=at\_end);
- (6) raster **ST\_AddBand**(raster rast, integer index, text outdbfile, integer[] outdbindex, double precision nodataval=NULL);
- (7) raster **ST\_AddBand**(raster rast, text outdbfile, integer[] outdbindex, integer index=at\_end, double precision nodataval=NULL);



```

        ARRAY[
            ROW(1, '1BB'::text, 0, NULL),
            ROW(2, '4BUI'::text, 0, NULL)
        ]::addbandarg[]
    )
);

-- &#xbaa8;&#xb4e0; &#xc791;&#xc5c5;&#xc774; &#xc815;&#xc0c1;&#xc778;&#xc9c0; &#xc655;&#xc778;&#xd558;&#xae30; &#xc704;&#xd574; &#xb798;&#xc2a4;&#xd130; &#xc34;&#xb4dc;&#xb4e4;&#xc758; &#xba54;&#xd0c0;&#xb370;&#xc774;&#xd130;&#xb97c; &#xcd9c;&#xb825; --
SELECT (bmd).*
FROM (SELECT ST_BandMetaData(rast,generate_series(1,2)) As bmd
      FROM dummy_rast WHERE rid = 10) AS foo;
-- &xacb0;&xacfc; --
pixeltype | nodatavalue | isoutdb | path
-----+-----+-----+-----
1BB      |              | f       |
4BUI     |              | f       |

-- &#xb798;&#xc2a4;&#xd130;&#xc758; &#xba54;&#xd0c0;&#xb370;&#xc774;&#xd130;&#xb97c; &#xcd9c;&#xb825; --
SELECT (rmd).width, (rmd).height, (rmd).numbands
FROM (SELECT ST_MetaData(rast) As rmd
      FROM dummy_rast WHERE rid = 10) AS foo;
-- &xacb0;&xacfc; --
upperleftx | upperlefty | width | height | scalex | scaley | skewx | skewy | srid | numbands
-----+-----+-----+-----+-----+-----+-----+-----+-----+-----
0 | 0 | 100 | 100 | 1 | -1 | 0 | 0 | 0 | 2

```

**&#xc608;&#xc2dc;: &#xbcf5;&#xc218;&#xc758; &#xc0c8;&#xb85c;&#xc6b4; &#xc34;&#xb4dc;**

```

SELECT
    *
FROM ST_BandMetadata(
    ST_AddBand(
        ST_MakeEmptyRaster(10, 10, 0, 0, 1, -1, 0, 0, 0),
        ARRAY[
            ROW(NULL, '8BUI', 255, 0),
            ROW(NULL, '16BUI', 1, 2),
            ROW(2, '32BUI', 100, 12),
            ROW(2, '32BF', 3.14, -1)
        ]::addbandarg[]
    ),
    ARRAY[]::integer[]
);

bandnum | pixeltype | nodatavalue | isoutdb | path
-----+-----+-----+-----+-----
1 | 8BUI      | 0           | f       |
2 | 32BF     | -1          | f       |
3 | 32BUI    | 12          | f       |
4 | 16BUI    | 2           | f       |

-- &#xb3d9;&#xc77c;&#xd55c; &#xc720;&#xd615;&#xc758; &#xb798;&#xc2a4;&#xd130;&#xb4e4;&#xc758; &#xd14c;&#xc774;&#xbe14;&#xc758; &#xccab; &#xc34;&#xb4dc;

```

```

-- &#xc88;&#xc9f8; &#xc34;&#xb4dc;&#xb97c; test_types&#xb9cc;&#xd07c; &#xb9ce;&#xc740; &#xc34;&#xb4dc;&#xb4e4;&#xacfc;
-- mice&#xb9cc;&#xd07c; &#xb9ce;&#xc740; &#xd589;&#xb4e4; (&#xc0c8; &#xb798;&#xc2a4;&#xd130;&#xb4e4;)&#xacfc; &#xd568;&#xae8; &#xb2e8;&#xc77c; &#xb798;&#xc2a4;&#xd130;&#xb85c; &#xc885;&#xd569;
-- &#xc8fc;&#xc758;: PostgreSQL 9.0 &#xc774;&#xc0c1; &#xc84;&#xc804;&#xb9cc; ORDER BY test_type&#xc744; &#xc9c0;&#xc6d0;&#xd569;&#xb2c8;&#xb2e4;.
-- 8.4 &#xc0f; &#xadf8; &#xc774;&#xc804; &#xc84;&#xc804;&#xc5d0;&#xc11c;&#xb294; &#xc0ac;&#xc6a9;&#xc790; &#xb370;&#xc774;&#xd130;&#xb97c; &#xd558;&#xc704; &#xc9d1;&#xd569;&#xc73c;&#xb85c; &#xc815;&#xb82c;&#xd558;&#xb294; &#xc2dd;&#xc73c;&#xb85c; (&#xb300;&#xbd80;&#xbd84;&#xc758; &#xacbd;&#xc6b0;)&#xb3d9;&#xc791;&#xd569;&#xb2c8;&#xb2e4;.
-- test_type&#xc758; &#xc54c;&#xd30c;&#xbcb3; &#xc21c;&#xc11c;&#xb85c; &#xac01; test_type &#xc5d0; &#xb300;&#xd55c; &#xc34;&#xb4dc;&#xb97c; &#xac00;&#xc9c4; &#xb798;&#xc2a4;&#xd130;&#xb97c; &#xcd9c;&#xb825;&#xd560; &#xac83;&#xc785;&#xb2c8;&#xb2e4;.
-- &#xb3d9;&#xbb3c;&#xbcf4;&#xd638;&#xb860;&#xc790; &#xbd84;&#xb4e4;&#xae8;: &#xc774; &#xc608;&#xc2dc;&#xc5d0;&#xc11c; &#xc5b4;&#xb5a4; &#xc950;(mice)&#xb3c4; &#xd76c;&#xc0dd;&#xb418;&#xc9c0; &#xc54a;&#xc558;&#xc2b5;&#xb2c8;&#xb2e4;.
SELECT
    mouse,
    ST_AddBand(NULL, array_agg(rast ORDER BY test_type), 1) As rast
FROM mice_studies
GROUP BY mouse;

```

**&#xc608;&#xc2dc;: &#xc0c8;&#xb85c;&#xc6b4; DB &#xc678;&#xbd80; &#xc34;&#xb4dc;**

```

SELECT
    *
FROM ST_BandMetadata (
    ST_AddBand(
        ST_MakeEmptyRaster(10, 10, 0, 0, 1, -1, 0, 0, 0),
        '/home/raster/mytestraster.tif'::text, NULL::int[]
    ),
    ARRAY[]::integer[]
);

```

| bandnum | pixeltype | nodatavalue | isoutdb | path                          |
|---------|-----------|-------------|---------|-------------------------------|
| 1       | 8BUI      |             | t       | /home/raster/mytestraster.tif |
| 2       | 8BUI      |             | t       | /home/raster/mytestraster.tif |
| 3       | 8BUI      |             | t       | /home/raster/mytestraster.tif |

**&#xcc38;&#xace0;**

[ST\\_BandMetaData](#), [ST\\_BandPixelType](#), [ST\\_MakeEmptyRaster](#), [ST\\_MetaData](#), [ST\\_NumBands](#), [ST\\_Reclass](#)

## 12.3.2 ST\_AsRaster

ST\_AsRaster — PostGIS &#xb3c4;&#xd615;&#xc744; PostGIS &#xb798;&#xc2a4;&#xd130;&#xb85c; &#xbcc0;&#xd658;&#xd569

### Synopsis

raster **ST\_AsRaster**(geometry geom, raster ref, text pixeltype, double precision value=1, double precision nodataval=0, boolean touched=false);

raster **ST\_AsRaster**(geometry geom, raster ref, text[] pixeltype=ARRAY['8BUI'], double precision[] value=ARRAY[1], double

```

precision[] nodataval=ARRAY[0], boolean touched=false);
raster ST_AsRaster(geometry geom, double precision scalex, double precision scaley, double precision gridx, double precision gridy, text pixeltype, double precision value=1, double precision nodataval=0, double precision skewx=0, double precision skewy=0, boolean touched=false);
raster ST_AsRaster(geometry geom, double precision scalex, double precision scaley, double precision gridx=NULL, double precision gridy=NULL, text[] pixeltype=ARRAY['8BIT'], double precision[] value=ARRAY[1], double precision[] nodataval=ARRAY[0], double precision skewx=0, double precision skewy=0, boolean touched=false);
raster ST_AsRaster(geometry geom, double precision scalex, double precision scaley, text pixeltype, double precision value=1, double precision nodataval=0, double precision upperleftx=NULL, double precision upperlefty=NULL, double precision skewx=0, double precision skewy=0, boolean touched=false);
raster ST_AsRaster(geometry geom, double precision scalex, double precision scaley, text[] pixeltype, double precision[] value=ARRAY[1], double precision[] nodataval=ARRAY[0], double precision upperleftx=NULL, double precision upperlefty=NULL, double precision skewx=0, double precision skewy=0, boolean touched=false);
raster ST_AsRaster(geometry geom, integer width, integer height, double precision gridx, double precision gridy, text pixeltype, double precision value=1, double precision nodataval=0, double precision skewx=0, double precision skewy=0, boolean touched=false);
raster ST_AsRaster(geometry geom, integer width, integer height, text pixeltype, double precision value=1, double precision nodataval=0, double precision upperleftx=NULL, double precision upperlefty=NULL, double precision skewx=0, double precision skewy=0, boolean touched=false);
raster ST_AsRaster(geometry geom, integer width, integer height, text[] pixeltype, double precision[] value=ARRAY[1], double precision[] nodataval=ARRAY[0], double precision upperleftx=NULL, double precision upperlefty=NULL, double precision skewx=0, double precision skewy=0, boolean touched=false);

```

## &#xc124;&#xba85;

```

PostGIS &#xb3c4;&#xd615;&#xc744; PostGIS &#xb798;&#xc2a4;&#xd130;&#xb85c; &#xbcc0;&#xd658;&#xd569;&#xb2c8;&#xb2e4;&#xc591;&#xd55c; &#xbcc0;&#xc885;&#xb4e4;&#xc744; &#xd1b5;&#xd574; &#xcd9c;&#xb825; &#xb798;&#xc2a4;&#xc815;&#xb82c; &#xbc29;&#xd5a5;&#xacfc; &#xd53d;&#xc140; &#xd06c;&#xae30;&#xb97c; &#xd06c;&#xac8c; &#xc138; &#xac00;&#xc9c0; &#xbc29;&#xbc95;&#xc73c;&#xb85c; &#xc124;&#xc815;&#xd560; &#xc218; &#xc788;&#xc2b5;&#xb2c8;&#xc98;&#xc74c; &#xb450; &#xbcc0;&#xc885;&#xc73c;&#xb85c; &#xc774;&#xb8e8;&#xc5b4;&#xc9c4; &#xccab; &#bc88;&#bc29;&#bc95;&#xc740; &#xc8fc;&#xc5b4;&#xc9c4; &#xcc38;&#xc870; &#xb798;&#xc2a4;&#xd130;&#xc640; &#xb3d9;&#xc815;&#xb82c; &#bc29;&#xd5a5;(scalex, scaley &#xbc0f; gridx, gridy), &#xd53d;&#xc140; &#xc720;&#xd615;, NODATA &#xac12;&#xc744; &#xac00;&#xc9c4; &#xb798;&#xc2a4;&#xd130;&#xb97c; &#xc0dd;&#xc131;&#xd569;&#xb2c8;&#xc77c;&#xbc18;&#xc801;&#xc73c;&#xb85c;, &#xcc38;&#xc870; &#xb798;&#xc2a4;&#xd130;&#xb97c; &#xb2f4;&#xace0; &#xc788;&#xb294; &#xd14c;&#xc774;&#xbe14;&#xacfc; &#xb3c4;&#xd615;&#xc744; &#xb2f4;&#xace0; &#xc788;&#xb294; &#xd14c;&#xc774;&#xbe14;&#xc744; &#xacb0;&#xd569;&#xd558;&#xb294; &#bc29;&#bc95;&#xc73c;&#xb85c; &#xd574;&#xc38;&#xc870; &#xb798;&#xc2a4;&#xd130;&#xb97c; &#xc785;&#xb825;&#xd569;&#xb2c8;&#xb2e4;.

&#xb124; &#xbcc0;&#xc885;&#xc73c;&#xb85c; &#xc774;&#xb8e8;&#xc5b4;&#xc9c4; &#xb450; &#bc88;&#xc9f8; &#bc29;&#xd53d;&#xc140; &#xd06c;&#xae30;(scalex & scaley &#xbc0f; skewx & skewy)&#xc758; &#xd30c;&#xb77c;&#xbbf8;&#xc81c;&#xacf5;, &#xc0ac;&#xc6a9;&#xc790;&#xac00; &#xb798;&#xc2a4;&#xd130;&#xc758; &#xcc28;&#xc6d0;&#xc744; &#xc124;&#xc815;&#xd560; &#xc218; &#xc788;&#xb3c4;&#xb85d; &#xd569;&#xb2c8;&#xb2e4;. &#xacb0;&#xacfc; &#xb798;&#xc2a4;&#xd130;&#xc758; &#xcc28;&#xc6d0;&#xc744; &#xc218;&#xc815;&#xd560; &#xc218; &#xc788;&#xb3c4;&#xd569;&#xb2c8;&#xb2e4;. &#xacb0;&#xacfc; &#xb798;&#xc2a4;&#xd130;&#xc758; &#xd53d;&#xc140; &#xd06c;&#xae30; &#xd30c;&#xb77c;&#xbbf8;&#xd130;(scalex & scaley &#xbc0f; skewx & skewy)&#xac00; &#xb3c4;&#xd615;&#xc758; &#bc94;&#xc704;&#xc5d0; &#xb9de;&#xb3c4;&#xb85d; &#xc870;&#xac83;&#xc785;&#xb2c8;&#xb2e4;. &#xb300;&#xbd80;&#xbd84;&#xc758; &#xacbd;&#xc6b0;, &#xc815;&#xc218;&#xd615; scalex & scaley &#xc778;&#xc218;&#xb4e4;&#xc744; PostgreSQL&#xc774; &#xc62c;&#xbc14;&#xb978; &#xbcc0;&#xc885; &#xc120;&#xd0dd;&#xd558;&#xb3c4;&#xb85d; &#xc774;&#xc911; &#xc815;&#xbcc0;&#xb3c4; &#xb370;&#xc774;&#xd130;&#xd615;&#xbcc0;&#xd658;&#xd574;&#xc57c;&#xb9cc; &#xd569;&#xb2c8;&#xb2e4;.

&#xb124; &#xbcc0;&#xc885;&#xc73c;&#xb85c; &#xc774;&#xb8e8;&#xc5b4;&#xc9c4; &#xc138; &#bc88;&#xc9f8; &#bc29;&#xb798;&#xc2a4;&#xd130;&#xc758; &#xcc28;&#xc6d0;(width & height)&#xc744; &#xc81c;&#xacf5;, &#xc0ac;&#xc6a9;&#xb798;&#xc2a4;&#xd130;&#xc758; &#xcc28;&#xc6d0;&#xc744; &#xc218;&#xc815;&#xd560; &#xc218; &#xc788;&#xb3c4;&#xd569;&#xb2c8;&#xb2e4;. &#xacb0;&#xacfc; &#xb798;&#xc2a4;&#xd130;&#xc758; &#xd53d;&#xc140; &#xd06c;&#xae30; &#xd30c;&#xb77c;&#xbbf8;&#xd130;(scalex & scaley &#xbc0f; skewx & skewy)&#xac00; &#xb3c4;&#xd615;&#xc758; &#bc94;&#xc704;&#xc5d0; &#xb9de;&#xb3c4;&#xb85d; &#xc870;&#xc815;&#xb420; &#xac83;&#xc785;&#xb2c8;&#xb2e4;.

```

&#xb4a4;&#xc758; &#xb450; &#xbc29;&#bc95; &#xac00;&#xc6b4;&#xb370; &#xac01; &#xbc29;&#bc95;&#xc758; &#xcc98;&#  
 &#xb450; &#xbcc0;&#xc885;&#xb4e4;&#xc740; &#xc0ac;&#xc6a9;&#xc790; &#xac00; &#xc815;&#xb82c; &#xadf8;&#xb9ac;&#  
 &#xc784;&#xc758;&#xc758; &#xbaa8;&#xc11c;&#xb9ac;(gridx & gridy)&#xb97c; &#xc124;&#xc815;&#xd560; &#xc218;  
 &#xc788;&#xb3c4;&#xb85d; &#xd569;&#xb2c8;&#xb2e4;. &#xb2e4;&#xb978; &#xb450; &#xbcc0;&#xc885;&#xb4e4;&#xc740;  
 &#xc88c;&#xc0c1;&#xb2e8;(upperleftx & upperlefty))&#xc744; &#xc785;&#xb825;&#xbc1b;&#xc2b5;&#xb2c8;&#xb2e4  
 &#xac01; &#bc29;&#bc95;&#xc758; &#bcc0;&#xc885;&#xc740; &#b2e8;&#xc77c; &#bc34;&#xb4dc; &#b798;&#xc2a4;&#  
 &#xb610;&#xb294; &#bcf5;&#xc218; &#bc34;&#xb4dc; &#b798;&#xc2a4;&#xd130;&#xb97c; &#xc0dd;&#xc131;&#xd560;  
 &#xc218; &#xc788;&#xac8c; &#xd574;&#xc90d;&#xb2c8;&#xb2e4;. &#bcf5;&#xc218; &#bc34;&#xb4dc; &#b798;&#xc2a4;&#  
 &#xc0dd;&#xc131;&#xd558;&#xb824;&#xba74;. &#xc0ac;&#xc6a9;&#xc790; &#xac00; &#xd53d;&#xc140; &#xc720;&#xd615;&#  
 &#bc30;&#xc5f4;(pixeltype[]), &#xcd08;&#xae30;&#xac12;&#xc758; &#bc30;&#xc5f4;(value) &#xadf8;&#xb9ac;&#xace  
 NODATA &#xac12;&#xc758; &#bc30;&#xc5f4;(nodataval)&#xc744; &#xc900;&#xbe44;&#xd574;&#xc57c;&#xb9cc;  
 &#xd569;&#xb2c8;&#xb2e4;. &#xc774;&#xb4e4;&#xc744; &#xc785;&#xb825;&#xd558;&#xc9c0; &#xc54a;&#xc73c;&#xba74.;&  
 &#xae30;&#bcf8;&#xc801;&#xc73c;&#xb85c; &#xd53d;&#xc140; &#xc720;&#xd615;&#xc740; 8BUI, &#xcd08;&#xae30;&#xac1  
 1, NODATA &#xac12;&#xc740; 0&#xc774; &#xb429;&#xb2c8;&#xb2e4;.

&#xcd9c;&#xb825; &#b798;&#xc2a4;&#xd130;&#xb294; &#xc18c;&#xc2a4; &#xb3c4;&#xd615;&#xacfc; &#xb3d9;&#xc77c;&#  
 &#xacf5;&#xac04; &#xcc38;&#xc870; &#xc2dc;&#xc2a4;&#xd15c;&#xc744; &#xac00;&#xc9c0;&#xac8c; &#xb429;&#xb2c8;&#  
 &#xc720;&#xc77c;&#xd55c; &#xc608;&#xc678;&#xb294; &#xcc38;&#xc870; &#b798;&#xc2a4;&#xd130;&#xb97c; &#bc1b;&#  
 &#bcc0;&#xc885;&#xb4e4;&#bfd0;&#xc785;&#xb2c8;&#xb2e4;. &#xc774;&#xb7f0; &#xacbd;&#xc6b0; &#xacb0;&#xacfc;  
 &#b798;&#xc2a4;&#xd130;&#xb294; &#xcc38;&#xc870; &#b798;&#xc2a4;&#xd130;&#xc640; &#xb3d9;&#xc77c;&#xd55c;  
 SRID&#xb97c; &#xac00;&#xc9c0;&#xac8c; &#xb429;&#xb2c8;&#xb2e4;.

&#xc120;&#xd0dd;&#xc801;&#xc778; touched &#xd30c;&#xb77c;&#xbbf8;&#xd130;&#xc758; &#xae30;&#bcf8;&#xac12;&#  
 &#xac70;&#xc9d3;&#xc73c;&#xb85c; GDAL\_ALL\_TOUCHED &#xb798;&#xc2a4;&#xd130;&#xd654; &#xc635;&#xc158;&#xc5d  
 &#xb9e4;&#xd551;&#xb418;&#xb294;&#xb370;. &#xc774; &#xc635;&#xc158;&#xc740; &#xb77c;&#xc778; &#xb610;&#xb294;  
 &#xd3f4;&#xb9ac;&#xace4;&#xc5d0; &#xb2ff;&#xc740; &#xd53d;&#xc140;&#xc744; &#xc5c6;&#xc568; &#xac83;&#xc778;&#  
 &#xb9d0; &#xac83;&#xc778;&#xc9c0; &#acb0;&#xc815;&#xd569;&#xb2c8;&#xb2e4;. &#b80c;&#xb354;&#xb9c1;&#xb41c;  
 &#xb77c;&#xc778; &#xacbd;&#xb85c; &#xc0c1;&#xc5d0; &#xc788;&#xb294; &#xd53d;&#xc140;&#xb9cc;&#xc774; &#xc544;&#  
 &#xadf8; &#xc911;&#xc2ec;&#xc810;&#xc774; &#xd3f4;&#xb9ac;&#xace4; &#xb0b4;&#xbd80;&#xc5d0; &#xc788;&#xb294;  
 &#xd53d;&#xc140; &#xb610;&#xd55c; &#xb9d0;&#xc774;&#xc8e0;.

&#xc774; &#xd568;&#xc218;&#xb294; **ST\_AsPNG** &#bc0f; &#xb2e4;&#xb978; **ST\_AsGDALRaster** &#xacc4;&#xc5f4;  
 &#xd568;&#xc218;&#xc640; &#acb0;&#xd569;&#xd574;&#xc11c; &#xc0ac;&#xc6a9;&#xd560; &#b54c; &#xb370;&#xc774;&#  
 &#xc9c1;&#xc811; &#xac00;&#xc838;&#xc628; &#xb3c4;&#xd615;&#xc758; JPEG &#xb610;&#xb294; PNG&#xb97c; &#xb80c;&  
 &#xb370; &#xd2b9;&#xd788; &#xc720;&#xc6a9;&#xd569;&#xb2c8;&#xb2e4;.

2.0.0 &#bc84;&#xc804;&#xbd80;&#xd130; &#xc0ac;&#xc6a9;&#xd560; &#xc218; &#xc788;&#xc2b5;&#xb2c8;&#xb2e4;.

GDAL 1.6.0 &#xc774;&#xc0c1; &#bc84;&#xc804;&#xc774; &#xd544;&#xc694;&#xd569;&#xb2c8;&#xb2e4;.

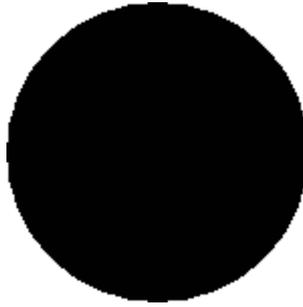
---

**Note**


&#xc544;&#xc9c1; &#xb9cc;&#xace1; &#xb3c4;&#xd615;, TIN, &#xb2e4;&#xba74;&#xccb4;  
 &#xd45c;&#xba74; &#xb4f1;&#xc758; &#bcf5;&#xc7a1; &#xb3c4;&#xd615; &#xc720;&#xd615;&#xc744;  
 &#xb80c;&#xb354;&#xb9c1;&#xd560; &#xc218;&#xb294; &#xc5c6;&#xc9c0;&#xb9cc;, GDAL&#xc774;  
 &#xad00;&#xb828; &#xae30;&#xb2a5;&#xc744; &#xc9c0;&#xc6d0;&#xd558;&#xac8c; &#xb418;&#xba74; &#xd560;  
 &#xc218; &#xc788;&#xac8c; &#xb420; &#xac83;&#xc785;&#xb2c8;&#xb2e4;.

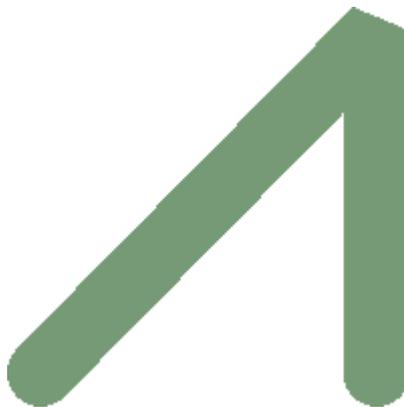
---

**ST\_Buffer**: PNG



**ST\_Buffer**

```
-- this will output a black circle taking up 150 x 150 pixels --
SELECT ST_AsPNG(ST_AsRaster(ST_Buffer(ST_Point(1,5),10),150, 150));
```



**ST\_Buffer**

```
-- RGB - (118,154,118) - ↔
-- ↔
SELECT ST_AsPNG(
  ST_AsRaster(
    ST_Buffer(
      ST_GeomFromText('LINESTRING(50 50,150 150,150 50)'), 10, 'join=bevel ↔
    ),
    200,200,ARRAY['8BUI', '8BUI', '8BUI'], ARRAY[118,154,118], ARRAY ↔
    [0,0,0]));
```

**ST\_Buffer**

[ST\\_BandPixelType](#), [ST\\_Buffer](#), [ST\\_GDALDrivers](#), [ST\\_AsGDALRaster](#), [ST\\_AsPNG](#), [ST\\_AsJPEG](#), [ST\\_SRID](#)

### 12.3.3 ST\_Band

**ST\_Band** — **ST\_Band**

## Synopsis

```
raster ST_Band(raster rast, integer[] nbands = ARRAY[1]);
raster ST_Band(raster rast, integer nband);
raster ST_Band(raster rast, text nbands, character delimiter=,);
```

### Warning

Returns one or more bands of an existing raster as a new raster. Useful for building new rasters from existing rasters or export of only selected bands of a raster or rearranging the order of bands in a raster. If no band is specified or any of specified bands does not exist in the raster, then all bands are returned. Used as a helper function in various functions such as for deleting a band.

#### Warning

 `ST_Band(rast, '1,2,3')` returns a raster with three bands. The first band is the original raster, the second band is the original raster with a value of 1, and the third band is the original raster with a value of 2. The function `ST_Band(rast, '1,2,3')` is equivalent to `ST_Band(rast, '1,2,3', ',')`.

2.0.0 `ST_Band(rast, '1,2,3')` returns a raster with three bands. The first band is the original raster, the second band is the original raster with a value of 1, and the third band is the original raster with a value of 2. The function `ST_Band(rast, '1,2,3')` is equivalent to `ST_Band(rast, '1,2,3', ',')`.

### Example

```
-- ST_Band(rast, '1,2,3')
SELECT ST_NumBands(rast1) As numb1, ST_BandPixelType(rast1) As pix1,
       ST_NumBands(rast2) As numb2, ST_BandPixelType(rast2) As pix2
FROM (
  SELECT ST_Band(rast) As rast1, ST_Reclass(ST_Band(rast,3), '100-200':1, [200-254:2', '2 ←
         BUI') As rast2
  FROM dummy_rast
  WHERE rid = 2) As foo;

numb1 | pix1 | numb2 | pix2
-----+-----+-----+-----
1 | 8BUI | 1 | 2BUI

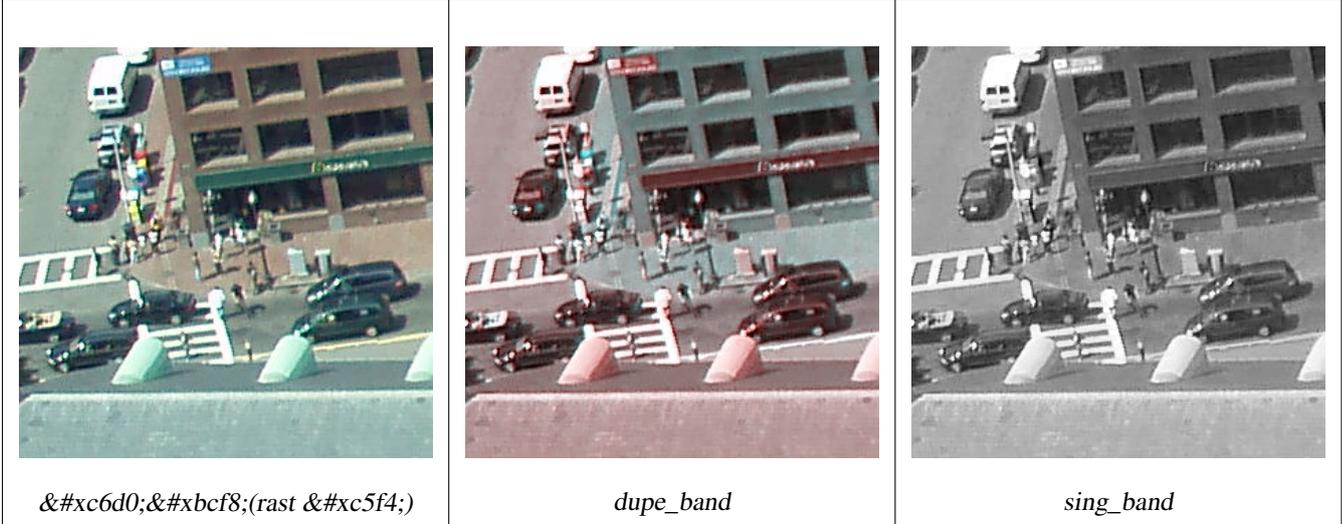
-- ST_Band(rast, '{2,3}::int[]')
SELECT ST_NumBands(ST_Band(rast, '{2,3}::int[]')) As num_bands
FROM dummy_rast WHERE rid=2;
```

```

num_bands
-----
2

-- &#x34; &#x4dc; 2 &#x0f; 3&#x744; &#xc18; &#xd658; &#xd569; &#b2c8; &#b2e4; . ←
  &#xc30; &#xc5f4; &#xc744; &#xc774; &#xc6a9; &#xd574; &#xc34; &#x4dc; &#b97c; ←
  &#xc815; &#xc758; &#xd569; &#b2c8; &#b2e4; .
SELECT ST_NumBands(ST_Band(rast, ARRAY[2,3])) As num_bands
FROM dummy_rast
WHERE rid=2;

```



```

-- &#xc6d0; &#xbcf8; &#xc758; &#xb450; &#xc88; &#xc9f8; &#xc34; &#x4dc; &#b97c; &#xd55c; ←
  &#xc88; , &#xcacab; &#xc88; &#xc9f8; &#xc34; &#x4dc; &#b97c; &#xb450; &#xc88; ←
  &#xc18; &#xbcf5; &#xd574; &#xc11c; &#xc0c8; &#xb798; &#xc2a4; &#xd130; &#b97c; ←
  &#xc791; &#xc131; &#xd558; &#xace0; , &#xc138; &#xc88; &#xc9f8; &#xc34; &#x4dc; &#b85c; ←
  &#xb610; &#b2e4; &#b978; &#xb798; &#xc2a4; &#xd130; &#b97c; ←
  &#xc791; &#xc131; &#xd569; &#b2c8; &#b2e4; .
SELECT rast, ST_Band(rast, ARRAY[2,1,1]) As dupe_band,
      ST_Band(rast, 3) As sing_band
FROM samples.than_chunked
WHERE rid=35;

```

**&#xcc38; &#xace0;**

[ST\\_AddBand](#), [ST\\_NumBands](#), [ST\\_Reclass](#), Chapter 12

### 12.3.4 ST\_MakeEmptyCoverage

ST\_MakeEmptyCoverage — Cover georeferenced area with a grid of empty raster tiles.

#### Synopsis

raster **ST\_MakeEmptyCoverage**(integer tilewidth, integer tileheight, integer width, integer height, double precision upperleftx, double precision upperlefty, double precision scalex, double precision scaley, double precision skewx, double precision skewy, integer srid=unknown);

**ST\_MakeEmptyRaster**

Create a set of raster tiles with **ST\_MakeEmptyRaster**. Grid dimension is `width` & `height`. Tile dimension is `tilewidth` & `tileheight`. The covered georeferenced area is from upper left corner (`upperleftx`, `upperlefty`) to lower right corner (`upperleftx + width * scalex`, `upperlefty + height * scaley`).



**Note**

Note that `scaley` is generally negative for rasters and `scalex` is generally positive. So lower right corner will have a lower `y` value and higher `x` value than the upper left corner.

Availability: 2.4.0

**ST\_MakeEmptyCoverage**

Create 16 tiles in a 4x4 grid to cover the WGS84 area from upper left corner (22, 77) to lower right corner (55, 33).

```
SELECT (ST_MetaData(tile)).* FROM ST_MakeEmptyCoverage(1, 1, 4, 4, 22, 33, (55 - 22)/(4)::float, (33 - 77)/(4)::float, 0., 0., 4326) tile;
```

| upperleftx | upperlefty | width | height | scalex | scaley | skewx | skewy | srid | numbands |
|------------|------------|-------|--------|--------|--------|-------|-------|------|----------|
| 22         | 33         | 1     | 1      | 8.25   | -11    | 0     | 0     | 4326 | 0        |
| 30.25      | 33         | 1     | 1      | 8.25   | -11    | 0     | 0     | 4326 | 0        |
| 38.5       | 33         | 1     | 1      | 8.25   | -11    | 0     | 0     | 4326 | 0        |
| 46.75      | 33         | 1     | 1      | 8.25   | -11    | 0     | 0     | 4326 | 0        |
| 22         | 22         | 1     | 1      | 8.25   | -11    | 0     | 0     | 4326 | 0        |
| 30.25      | 22         | 1     | 1      | 8.25   | -11    | 0     | 0     | 4326 | 0        |
| 38.5       | 22         | 1     | 1      | 8.25   | -11    | 0     | 0     | 4326 | 0        |
| 46.75      | 22         | 1     | 1      | 8.25   | -11    | 0     | 0     | 4326 | 0        |
| 22         | 11         | 1     | 1      | 8.25   | -11    | 0     | 0     | 4326 | 0        |
| 30.25      | 11         | 1     | 1      | 8.25   | -11    | 0     | 0     | 4326 | 0        |
| 38.5       | 11         | 1     | 1      | 8.25   | -11    | 0     | 0     | 4326 | 0        |
| 46.75      | 11         | 1     | 1      | 8.25   | -11    | 0     | 0     | 4326 | 0        |
| 22         | 0          | 1     | 1      | 8.25   | -11    | 0     | 0     | 4326 | 0        |
| 30.25      | 0          | 1     | 1      | 8.25   | -11    | 0     | 0     | 4326 | 0        |
| 38.5       | 0          | 1     | 1      | 8.25   | -11    | 0     | 0     | 4326 | 0        |
| 46.75      | 0          | 1     | 1      | 8.25   | -11    | 0     | 0     | 4326 | 0        |

**ST\_MakeEmptyRaster**

**ST\_MakeEmptyRaster**

### 12.3.5 ST\_MakeEmptyRaster

**ST\_MakeEmptyRaster** — `(integer width, integer height, float8 upperleftx, float8 upperlefty, float8 scalex, float8 scaley, float8 skewx, float8 skewy, integer srid=unknown)`

#### Synopsis

raster **ST\_MakeEmptyRaster**(raster rast);

raster **ST\_MakeEmptyRaster**(integer width, integer height, float8 upperleftx, float8 upperlefty, float8 scalex, float8 scaley, float8 skewx, float8 skewy, integer srid=unknown);

raster **ST\_MakeEmptyRaster**(integer width, integer height, float8 upperleftx, float8 upperlefty, float8 pixelsize);

#### Parameters

`width`: width of the raster in pixels.  
`height`: height of the raster in pixels.  
`upperleftx`: x coordinate of the upper-left corner of the raster.  
`upperlefty`: y coordinate of the upper-left corner of the raster.  
`scalex`: x-axis scale factor.  
`scaley`: y-axis scale factor.  
`skewx`: x-axis skew factor.  
`skewy`: y-axis skew factor.  
`srid`: SRID of the raster.

`pixelsize`: size of one pixel in the x and y directions.

`upperleftx`, `upperlefty`, `scalex`, `scaley`, `skewx`, `skewy`, `pixelsize`: if these parameters are not specified, the raster will be created with a width of 100, a height of 100, and a scale of 1.0 in both directions.

`SRID`: if not specified, the raster will be created with a SRID of 0 (unknown).

#### Examples

```
INSERT INTO dummy_rast (rid, rast)
VALUES (3, ST_MakeEmptyRaster( 100, 100, 0.0005, 0.0005, 1, 1, 0, 0, 4326) );
```

```
-- Create a raster with a width of 100, a height of 100, and a scale of 1.0 in both directions.
-- The raster will be created with a SRID of 0 (unknown).
-- The raster will be created with a width of 100, a height of 100, and a scale of 1.0 in both directions.
```

```
INSERT INTO dummy_rast (rid, rast)
SELECT 4, ST_MakeEmptyRaster(rast)
FROM dummy_rast WHERE rid = 3;
```

```
-- Create a raster with a width of 100, a height of 100, and a scale of 1.0 in both directions.
-- The raster will be created with a SRID of 0 (unknown).
```



2.1.0

### Example 2.1.0

```

WITH foo AS (
  SELECT ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(3, 3, 0, 0, 1, -1, 0, 0, 0), 1, '8 ←
    BUI', 1, 0), 2, '8BUI', 10, 0) AS rast UNION ALL
  SELECT ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(3, 3, 3, 0, 1, -1, 0, 0, 0), 1, '8 ←
    BUI', 2, 0), 2, '8BUI', 20, 0) AS rast UNION ALL
  SELECT ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(3, 3, 6, 0, 1, -1, 0, 0, 0), 1, '8 ←
    BUI', 3, 0), 2, '8BUI', 30, 0) AS rast UNION ALL

  SELECT ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(3, 3, 0, -3, 1, -1, 0, 0, 0), 1, '8 ←
    BUI', 4, 0), 2, '8BUI', 40, 0) AS rast UNION ALL
  SELECT ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(3, 3, 3, -3, 1, -1, 0, 0, 0), 1, '8 ←
    BUI', 5, 0), 2, '8BUI', 50, 0) AS rast UNION ALL
  SELECT ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(3, 3, 6, -3, 1, -1, 0, 0, 0), 1, '8 ←
    BUI', 6, 0), 2, '8BUI', 60, 0) AS rast UNION ALL

  SELECT ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(3, 3, 0, -6, 1, -1, 0, 0, 0), 1, '8 ←
    BUI', 7, 0), 2, '8BUI', 70, 0) AS rast UNION ALL
  SELECT ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(3, 3, 3, -6, 1, -1, 0, 0, 0), 1, '8 ←
    BUI', 8, 0), 2, '8BUI', 80, 0) AS rast UNION ALL
  SELECT ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(3, 3, 6, -6, 1, -1, 0, 0, 0), 1, '8 ←
    BUI', 9, 0), 2, '8BUI', 90, 0) AS rast
), bar AS (
  SELECT ST_Union(rast) AS rast FROM foo
), baz AS (
  SELECT ST_Tile(rast, 3, 3, TRUE) AS rast FROM bar
)
SELECT
  ST_DumpValues(rast)
FROM baz;

```

#### st\_dumpvalues

```

-----
(1,"{{1,1,1},{1,1,1},{1,1,1}}")
(2,"{{10,10,10},{10,10,10},{10,10,10}}")
(1,"{{2,2,2},{2,2,2},{2,2,2}}")
(2,"{{20,20,20},{20,20,20},{20,20,20}}")
(1,"{{3,3,3},{3,3,3},{3,3,3}}")
(2,"{{30,30,30},{30,30,30},{30,30,30}}")
(1,"{{4,4,4},{4,4,4},{4,4,4}}")
(2,"{{40,40,40},{40,40,40},{40,40,40}}")
(1,"{{5,5,5},{5,5,5},{5,5,5}}")
(2,"{{50,50,50},{50,50,50},{50,50,50}}")
(1,"{{6,6,6},{6,6,6},{6,6,6}}")
(2,"{{60,60,60},{60,60,60},{60,60,60}}")
(1,"{{7,7,7},{7,7,7},{7,7,7}}")
(2,"{{70,70,70},{70,70,70},{70,70,70}}")
(1,"{{8,8,8},{8,8,8},{8,8,8}}")
(2,"{{80,80,80},{80,80,80},{80,80,80}}")
(1,"{{9,9,9},{9,9,9},{9,9,9}}")
(2,"{{90,90,90},{90,90,90},{90,90,90}}")
(18 rows)

```

```

WITH foo AS (
  SELECT ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(3, 3, 0, 0, 1, -1, 0, 0, 0), 1, '8 ←
    BUI', 1, 0), 2, '8BUI', 10, 0) AS rast UNION ALL
  SELECT ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(3, 3, 3, 0, 1, -1, 0, 0, 0), 1, '8 ←
    BUI', 2, 0), 2, '8BUI', 20, 0) AS rast UNION ALL

```

```

SELECT ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(3, 3, 6, 0, 1, -1, 0, 0, 0), 1, '8 ←
  BUI', 3, 0), 2, '8BUI', 30, 0) AS rast UNION ALL

SELECT ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(3, 3, 0, -3, 1, -1, 0, 0, 0), 1, '8 ←
  BUI', 4, 0), 2, '8BUI', 40, 0) AS rast UNION ALL
SELECT ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(3, 3, 3, -3, 1, -1, 0, 0, 0), 1, '8 ←
  BUI', 5, 0), 2, '8BUI', 50, 0) AS rast UNION ALL
SELECT ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(3, 3, 6, -3, 1, -1, 0, 0, 0), 1, '8 ←
  BUI', 6, 0), 2, '8BUI', 60, 0) AS rast UNION ALL

SELECT ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(3, 3, 0, -6, 1, -1, 0, 0, 0), 1, '8 ←
  BUI', 7, 0), 2, '8BUI', 70, 0) AS rast UNION ALL
SELECT ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(3, 3, 3, -6, 1, -1, 0, 0, 0), 1, '8 ←
  BUI', 8, 0), 2, '8BUI', 80, 0) AS rast UNION ALL
SELECT ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(3, 3, 6, -6, 1, -1, 0, 0, 0), 1, '8 ←
  BUI', 9, 0), 2, '8BUI', 90, 0) AS rast
), bar AS (
  SELECT ST_Union(rast) AS rast FROM foo
), baz AS (
  SELECT ST_Tile(rast, 3, 3, 2) AS rast FROM bar
)
SELECT
  ST_DumpValues(rast)
FROM baz;

```

```

          st_dumpvalues
-----

```

```

(1, "{10,10,10},{10,10,10},{10,10,10}")
(1, "{20,20,20},{20,20,20},{20,20,20}")
(1, "{30,30,30},{30,30,30},{30,30,30}")
(1, "{40,40,40},{40,40,40},{40,40,40}")
(1, "{50,50,50},{50,50,50},{50,50,50}")
(1, "{60,60,60},{60,60,60},{60,60,60}")
(1, "{70,70,70},{70,70,70},{70,70,70}")
(1, "{80,80,80},{80,80,80},{80,80,80}")
(1, "{90,90,90},{90,90,90},{90,90,90}")
(9 rows)

```

**ST\_Union**

**ST\_Retile**

### 12.3.7 ST\_Retile

**ST\_Retile** — Retile a raster into a grid of smaller rasters. The output raster has the same extent as the input raster, but is divided into a grid of smaller rasters. The output raster is a grid of smaller rasters, each of which is a copy of the input raster, but with a different offset. The output raster is a grid of smaller rasters, each of which is a copy of the input raster, but with a different offset.

#### Synopsis

setof raster **ST\_Retile**(regclass tab, name col, geometry ext, float8 sfx, float8 sfy, int tw, int th, text algo='NearestNeighbor');

**ST\_Retile**

**ST\_Retile**(regclass tab, name col, geometry ext, float8 sfx, float8 sfy, int tw, int th, text algo='NearestNeighbor');

'NearestNeighbor', 'Bilinear', 'Cubic', 'Cubic-Spline', 'Lanczos' &#x00; &#xc788;&#xc2b5;&#xb2c8;&#xb2e4;. &#xc790;&#xc138;&#xd55c;&#xb0b4;&#xc6a9;&#xc740; **GDAL Warp resampling methods** &#xb97c; &#xcc38;&#xc870;&#xd558;&#xc2ed;&#xc2dc;&#xc624;. 2.2.0 &#xbc84;&#xc804;&#xbd80;&#xd130; &#xc0ac;&#xc6a9;&#xd560; &#xc218; &#xc788;&#xc2b5;&#xb2c8;&#xb2e4;.

**&#xcc38;&#xace0;**

[ST\\_CreateOverview](#)

### 12.3.8 ST\_FromGDALRaster

ST\_FromGDALRaster — &#xc9c0;&#xc6d0; GDAL &#xb798;&#xc2a4;&#xd130; &#xd30c;&#xc77c;&#xb85c;&#xbd80;&#xd130; &#xb798;&#xc2a4;&#xd130; &#xb97c; &#xbc18;&#xd658;&#xd569;&#xb2c8;&#xb2e4;.

#### Synopsis

raster **ST\_FromGDALRaster**(bytea gdaldata, integer srid=NULL);

**&#xc124;&#xba85;**

&#xc9c0;&#xc6d0; GDAL &#xb798;&#xc2a4;&#xd130; &#xd30c;&#xc77c;&#xb85c;&#xbd80;&#xd130; &#xb798;&#xc2a4;&#xd130; &#xb97c; &#xbc18;&#xd658;&#xd569;&#xb2c8;&#xb2e4;. gdaldata &#xb294; bytea &#xc720;&#xd615;&#xc73c;&#xb85c; GDAL &#xb798;&#xc2a4;&#xd130; &#xd30c;&#xc77c;&#xc758; &#xb0b4;&#xc6a9;&#xc744; &#xb2f4;&#xace0; &#xc788;&#xc5b4;&#xd569;&#xb2c8;&#xb2e4;.

srid &#xac00; NULL&#xc77c; &#xacbd;&#xc6b0;, &#xc774; &#xd568;&#xc218;&#xb294; GDAL &#xb798;&#xc2a4;&#xd130; &#xc790;&#xb3d9;&#xc801;&#xc73c;&#xb85c; SRID&#xb97c; &#xd560;&#xb2f9;&#xd558;&#xb824; &#xd560; &#xac83;&#xc774; srid &#xb97c; &#xc124;&#xc815;&#xd55c; &#xacbd;&#xc6b0;, &#xd574;&#xb2f9; &#xac12;&#xc774; &#xc790;&#xb3d9;&#xc774; &#xd560;&#xb2f9;&#xb41c; &#xc5b4;&#xb5a4; SRID&#xbcf4;&#xb2e4; &#xc6b0;&#xc120;&#xd560; &#xac83;&#xc785;&#xb2c8;&#xb2e4;.

2.1.0 &#xbc84;&#xc804;&#xbd80;&#xd130; &#xc0ac;&#xc6a9;&#xd560; &#xc218; &#xc788;&#xc2b5;&#xb2c8;&#xb2e4;.

**&#xc608;&#xc2dc;**

```
WITH foo AS (
    SELECT ST_AsPNG(ST_AddBand(ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(2, 2, 0, 0, ←
        0.1, -0.1, 0, 0, 4326), 1, '8BUI', 1, 0), 2, '8BUI', 2, 0), 3, '8BUI', 3, 0)) AS ←
        png
),
bar AS (
    SELECT 1 AS rid, ST_FromGDALRaster(png) AS rast FROM foo
    UNION ALL
    SELECT 2 AS rid, ST_FromGDALRaster(png, 3310) AS rast FROM foo
)
SELECT
    rid,
    ST_Metadatas(rast) AS metadata,
    ST_SummaryStats(rast, 1) AS stats1,
    ST_SummaryStats(rast, 2) AS stats2,
    ST_SummaryStats(rast, 3) AS stats3
FROM bar
ORDER BY rid;
```

| rid | metadata               | stats1        | stats2        | stats3         |
|-----|------------------------|---------------|---------------|----------------|
| 1   | (0,0,2,2,1,-1,0,0,0,3) | (4,4,1,0,1,1) | (4,8,2,0,2,2) | (4,12,3,0,3,3) |

```
 2 | (0,0,2,2,1,-1,0,0,3310,3) | (4,4,1,0,1,1) | (4,8,2,0,2,2) | (4,12,3,0,3,3)
(2 rows)
```

**ST\_AsGDALRaster**

**ST\_AsGDALRaster**

## 12.4 ST\_GeoReference

### 12.4.1 ST\_GeoReference

**ST\_GeoReference** — Returns the geographic information system (GIS) metadata for a raster. The metadata is returned as a text string. The metadata includes the coordinate system (CRS), the units, the datum, the projection, the scale, the skew, the scale, the upper left corner, and the upper left corner.

#### Synopsis

```
text ST_GeoReference(raster rast, text format=GDAL);
```

**ST\_GeoReference**

**ST\_GeoReference** (raster rast, text format=GDAL) returns the geographic information system (GIS) metadata for a raster. The metadata is returned as a text string. The metadata includes the coordinate system (CRS), the units, the datum, the projection, the scale, the skew, the scale, the upper left corner, and the upper left corner.

The metadata is returned as a text string. The metadata includes the coordinate system (CRS), the units, the datum, the projection, the scale, the skew, the scale, the upper left corner, and the upper left corner.

GDAL:

```
scalex
skewy
skewx
scaley
upperleftx
upperlefty
```

ESRI:

```
scalex
skewy
skewx
scaley
upperleftx + scalex*0.5
upperlefty + scaley*0.5
```

**Example**

```
SELECT ST_GeoReference(rast, 'ESRI') As esri_ref, ST_GeoReference(rast, 'GDAL') As gdal_ref
FROM dummy_rast WHERE rid=1;
```

| esri_ref     | gdal_ref     |
|--------------|--------------|
| 2.0000000000 | 2.0000000000 |
| 0.0000000000 | 0.0000000000 |
| 0.0000000000 | 0.0000000000 |
| 3.0000000000 | 3.0000000000 |
| 1.5000000000 | 0.5000000000 |
| 2.0000000000 | 0.5000000000 |

**See Also**

[ST\\_SetGeoReference](#), [ST\\_ScaleX](#), [ST\\_ScaleY](#)

**12.4.2 ST\_Height**

**ST\_Height** — Returns the height of a raster.

**Synopsis**

```
integer ST_Height(raster rast);
```

**Parameters**

**rast**: raster to be measured.

**Example**

```
SELECT rid, ST_Height(rast) As rastheight
FROM dummy_rast;
```

| rid | rastheight |
|-----|------------|
| 1   | 20         |
| 2   | 5          |

**See Also**

[ST\\_Width](#)

**12.4.3 ST\_IsEmpty**

**ST\_IsEmpty** — Returns true if a raster is empty (width = 0, height = 0).

## Synopsis

boolean **ST\_IsEmpty**(raster rast);

### 12.4.3 ST\_IsEmpty

ST\_IsEmpty(rast) — Returns true if the raster has no pixels (width = 0, height = 0). Returns false if the raster has one or more pixels.

2.0.0 Returns true if the raster has no pixels (width = 0, height = 0). Returns false if the raster has one or more pixels.

### 12.4.4 ST\_MemSize

```
SELECT ST_IsEmpty(ST_MakeEmptyRaster(100, 100, 0, 0, 0, 0, 0, 0))
```

```
st_isempty |
-----+
f          |
```

```
SELECT ST_IsEmpty(ST_MakeEmptyRaster(0, 0, 0, 0, 0, 0, 0, 0))
```

```
st_isempty |
-----+
t          |
```

### 12.4.5 ST\_HasNoBand

#### ST\_HasNoBand

## 12.4.4 ST\_MemSize

ST\_MemSize(rast) — Returns the number of bytes of memory used by the raster. Returns 0 if the raster has no pixels (width = 0, height = 0). Returns the number of bytes of memory used by the raster if the raster has one or more pixels.

## Synopsis

integer **ST\_MemSize**(raster rast);

### 12.4.5 ST\_MemSize

ST\_MemSize(rast) — Returns the number of bytes of memory used by the raster. Returns 0 if the raster has no pixels (width = 0, height = 0). Returns the number of bytes of memory used by the raster if the raster has one or more pixels.

2.0.0 Returns the number of bytes of memory used by the raster. Returns 0 if the raster has no pixels (width = 0, height = 0). Returns the number of bytes of memory used by the raster if the raster has one or more pixels. PostgreSQL: pg\_column\_size, pg\_size\_pretty, pg\_relation\_size, pg\_total\_relation\_size

**Note**



pg\_relation\_size, ST\_MemSize, TOAST, pg\_column\_size, pg\_total\_relation\_size

2.2.0

**SQL**

```
SELECT ST_MemSize(ST_AsRaster(ST_Buffer(ST_Point(1,5),10,1000),150, 150, '8BUI')) As
rast_mem;

rast_mem
-----
22568
```

**SQL**

**12.4.5 ST\_MetaData**

ST\_MetaData — (skew), (skewx | skewy | srid | numbands

**Synopsis**

record ST\_MetaData(raster rast);

**SQL**

upperleftx | upperlefty | width | height | scalex | scaley | skewx | skewy | srid | numbands

**SQL**

```
SELECT rid, (foo.md).*
FROM (SELECT rid, ST_MetaData(rast) As md
FROM dummy_rast) As foo;
```

| rid | upperleftx | upperlefty | width | height | scalex | scaley | skewx | skewy | srid | ← |
|-----|------------|------------|-------|--------|--------|--------|-------|-------|------|---|
|     | numbands   |            |       |        |        |        |       |       |      |   |
| 1   | 0.5        | 0.5        | 10    | 20     | 2      | 3      |       |       | 0    | ← |
| 2   | 3427927.75 | 5793244    | 5     | 5      | 0.05   | -0.05  |       |       | 0    | ← |

**ST\_BandMetaData**

**ST\_NumBands**

## 12.4.6 ST\_NumBands

**ST\_NumBands** — Returns the number of bands in a raster. The function returns an integer value.

### Synopsis

integer **ST\_NumBands**(raster rast);

**ST\_NumBands**

Returns the number of bands in a raster. The function returns an integer value.

**ST\_NumBands**

```
SELECT rid, ST_NumBands(rast) As numbands
FROM dummy_rast;
```

| rid | numbands |
|-----|----------|
| 1   | 0        |
| 2   | 3        |

**ST\_Value**

**ST\_Value**

## 12.4.7 ST\_PixelHeight

**ST\_PixelHeight** — Returns the pixel height of a raster. The function returns a double precision value.

### Synopsis

double precision **ST\_PixelHeight**(raster rast);

**ST\_Height**

ST\_Height(*rast*) — Returns the height of the raster in pixels. The height is the number of rows in the raster. The height is always a positive integer.

ST\_Height(*rast*, *width*) — Returns the height of the raster in pixels. The height is the number of rows in the raster. The height is always a positive integer. The width is the number of columns in the raster.

**ST\_PixelWidth**

```
SELECT ST_Height(rast) As rastheight, ST_PixelHeight(rast) As pixheight,
       ST_ScaleX(rast) As scalex, ST_ScaleY(rast) As scaley, ST_SkewX(rast) As skewx,
       ST_SkewY(rast) As skewy
FROM dummy_rast;
```

| rastheight | pixheight | scalex | scaley | skewx | skewy |
|------------|-----------|--------|--------|-------|-------|
| 20         | 3         | 2      | 3      | 0     | 0     |
| 5          | 0.05      | 0.05   | -0.05  | 0     | 0     |

**ST\_SetSkew**

```
SELECT ST_Height(rast) As rastheight, ST_PixelHeight(rast) As pixheight,
       ST_ScaleX(rast) As scalex, ST_ScaleY(rast) As scaley, ST_SkewX(rast) As skewx,
       ST_SkewY(rast) As skewy
FROM (SELECT ST_SetSkew(rast,0.5,0.5) As rast
      FROM dummy_rast) As skewed;
```

| rastheight | pixheight         | scalex | scaley | skewx | skewy |
|------------|-------------------|--------|--------|-------|-------|
| 20         | 3.04138126514911  | 2      | 3      | 0.5   | 0.5   |
| 5          | 0.502493781056044 | 0.05   | -0.05  | 0.5   | 0.5   |

**ST\_PixelWidth**

ST\_PixelWidth, ST\_ScaleX, ST\_ScaleY, ST\_SkewX, ST\_SkewY

**12.4.8 ST\_PixelWidth**

ST\_PixelWidth(*rast*) — Returns the pixel width of the raster in pixels. The pixel width is the number of columns in the raster. The pixel width is always a positive integer.

**Synopsis**

double precision **ST\_PixelWidth**(raster *rast*);

**Skewing:**

Skewing is a transformation that shears a shape. It is defined by a scale factor and a skew angle. The diagram shows a 2D coordinate system with X and Y axes. A dashed line represents the original shape, and a solid line represents the skewed shape. The skew angle is labeled  $\theta_i$ . The scale factor is labeled  $Y\_SCALE$  and  $X\_SCALE$ . The skew parameters are labeled  $X\_SKEW$  and  $Y\_SKEW$ . The diagram also shows vectors  $\vec{j}_b$  and  $\vec{i}_b$  and their magnitudes  $\|\vec{j}_b\|$  and  $\|\vec{i}_b\|$ .

Skewing is defined by a scale factor and a skew angle. The diagram shows a 2D coordinate system with X and Y axes. A dashed line represents the original shape, and a solid line represents the skewed shape. The skew angle is labeled  $\theta_i$ . The scale factor is labeled  $Y\_SCALE$  and  $X\_SCALE$ . The skew parameters are labeled  $X\_SKEW$  and  $Y\_SKEW$ . The diagram also shows vectors  $\vec{j}_b$  and  $\vec{i}_b$  and their magnitudes  $\|\vec{j}_b\|$  and  $\|\vec{i}_b\|$ .

**Skewing Parameters:**

```
SELECT ST_Width(rast) As rastwidth, ST_PixelWidth(rast) As pixwidth,
       ST_ScaleX(rast) As scalex, ST_ScaleY(rast) As scaley, ST_SkewX(rast) As skewx,
       ST_SkewY(rast) As skewy
FROM dummy_rast;
```

| rastwidth | pixwidth | scalex | scaley | skewx | skewy |
|-----------|----------|--------|--------|-------|-------|
| 10        | 2        | 2      | 3      | 0     | 0     |
| 5         | 0.05     | 0.05   | -0.05  | 0     | 0     |

**Skewing Parameters (Skewed):**

```
SELECT ST_Width(rast) As rastwidth, ST_PixelWidth(rast) As pixwidth,
       ST_ScaleX(rast) As scalex, ST_ScaleY(rast) As scaley, ST_SkewX(rast) As skewx,
       ST_SkewY(rast) As skewy
FROM (SELECT ST_SetSkew(rast,0.5,0.5) As rast
FROM dummy_rast) As skewed;
```

| rastwidth | pixwidth          | scalex | scaley | skewx | skewy |
|-----------|-------------------|--------|--------|-------|-------|
| 10        | 2.06155281280883  | 2      | 3      | 0.5   | 0.5   |
| 5         | 0.502493781056044 | 0.05   | -0.05  | 0.5   | 0.5   |



**ST\_ScaleY**

```
SELECT rid, ST_ScaleY(rast) As rastpixheight
FROM dummy_rast;
```

| rid | rastpixheight |
|-----|---------------|
| 1   | 3             |
| 2   | -0.05         |

**ST\_Height****ST\_Height****12.4.11 ST\_RasterToWorldCoord**

**ST\_RasterToWorldCoord** — Returns the world coordinates (X, Y) for a given pixel in a raster. The pixel is identified by its row and column indices (row, col). The raster is identified by its name.

**Synopsis**

record **ST\_RasterToWorldCoord**(raster rast, integer xcolumn, integer yrow);

**ST\_RasterToWorldCoord**

Returns the world coordinates (X, Y) for a given pixel in a raster. The pixel is identified by its row and column indices (row, col). The raster is identified by its name.

2.1.0 Returns the world coordinates (X, Y) for a given pixel in a raster. The pixel is identified by its row and column indices (row, col). The raster is identified by its name.

**ST\_RasterToWorldCoord**

```
-- Returns the world coordinates (X, Y) for a given pixel in a raster.
SELECT
    rid,
    (ST_RasterToWorldCoord(rast, 1, 1)).*,
    (ST_RasterToWorldCoord(rast, 2, 2)).*
FROM dummy_rast
```

| rid | longitude  | latitude | longitude | latitude   |
|-----|------------|----------|-----------|------------|
| 1   | 0.5        | 0.5      | 2.5       | 3.5        |
| 2   | 3427927.75 | 5793244  | 3427927.8 | 5793243.95 |



2.1.0; ST\_RasterToWorldCoordY, ST\_ScaleX, ST\_SetSkew, ST\_SkewX

### Example 1

```
-- ST_RasterToWorldCoordX, ST_ScaleX
SELECT rid, ST_RasterToWorldCoordX(rast,1) As x1coord,
        ST_RasterToWorldCoordX(rast,2) As x2coord,
        ST_ScaleX(rast) As pixelx
FROM dummy_rast;
```

| rid | x1coord    | x2coord   | pixelx |
|-----|------------|-----------|--------|
| 1   | 0.5        | 2.5       | 2      |
| 2   | 3427927.75 | 3427927.8 | 0.05   |

```
-- ST_RasterToWorldCoordY, ST_SetSkew, ST_SkewX
SELECT rid, ST_RasterToWorldCoordX(rast, 1, 1) As x1coord,
        ST_RasterToWorldCoordX(rast, 2, 3) As x2coord,
        ST_ScaleX(rast) As pixelx
FROM (SELECT rid, ST_SetSkew(rast, 100.5, 0) As rast FROM dummy_rast) As foo;
```

| rid | x1coord    | x2coord   | pixelx |
|-----|------------|-----------|--------|
| 1   | 0.5        | 203.5     | 2      |
| 2   | 3427927.75 | 3428128.8 | 0.05   |

### Example 2

ST\_ScaleX, ST\_RasterToWorldCoordY, ST\_SetSkew, ST\_SkewX

## 12.4.13 ST\_RasterToWorldCoordY

ST\_RasterToWorldCoordY — Returns the world coordinate Y of the pixel at the given row and column of the raster. The raster must be in SRID 4326. The function returns a float8 value.

### Synopsis

```
float8 ST_RasterToWorldCoordY(raster rast, integer yrow);
float8 ST_RasterToWorldCoordY(raster rast, integer xcolumn, integer yrow);
```

### Example 1

```
-- ST_RasterToWorldCoordY
SELECT rid, ST_RasterToWorldCoordY(rast, 1) As y1,
        ST_RasterToWorldCoordY(rast, 2) As y2,
        ST_ScaleX(rast) As pixelx
FROM dummy_rast;
```



**ST\_Rotation**

ST\_Rotation(*rast*, *angle*) — Returns a raster with the same geometry as *rast*, but rotated by *angle* degrees. *angle* is in degrees and can be negative. The rotation is performed around the center of mass of the raster. The output raster has the same resolution and extent as the input raster. The output raster is a float8 raster.

**ST\_Rotation**

```
SELECT rid, ST_Rotation(ST_SetScale(ST_SetSkew(rast, sqrt(2)), sqrt(2))) as rot FROM dummy_rast;
```

| rid | rot               |
|-----|-------------------|
| 1   | 0.785398163397448 |
| 2   | 0.785398163397448 |

**ST\_Rotation**

[ST\\_Rotation](#), [ST\\_SetScale](#), [ST\\_SetSkew](#)

**12.4.15 ST\_SkewX**

ST\_SkewX(*rast*, *skew*) — Returns a raster with the same geometry as *rast*, but skewed by *skew* units. The skewing is performed around the center of mass of the raster. The output raster has the same resolution and extent as the input raster. The output raster is a float8 raster.

**Synopsis**

float8 ST\_SkewX(raster rast);

**ST\_SkewX**

ST\_SkewX(*rast*, *skew*) — Returns a raster with the same geometry as *rast*, but skewed by *skew* units. The skewing is performed around the center of mass of the raster. The output raster has the same resolution and extent as the input raster. The output raster is a float8 raster.

**ST\_SkewX**

```
SELECT rid, ST_SkewX(rast) As skewx, ST_SkewY(rast) As skewy,
       ST_GeoReference(rast) as georef
FROM dummy_rast;
```

| rid | skewx | skewy | georef  |
|-----|-------|-------|---|
| 1   | 0     | 0     | 2.0000000000<br>: 0.0000000000<br>: 0.0000000000<br>: 3.0000000000<br>: 0.5000000000<br>: 0.5000000000<br>: |
| 2   | 0     | 0     | 0.0500000000<br>: 0.0000000000<br>: 0.0000000000  |

```

: -0.0500000000
: 3427927.7500000000
: 5793244.0000000000

```

**ST\_SkewY**

[ST\\_GeoReference](#), [ST\\_SkewY](#), [ST\\_SetSkew](#)

## 12.4.16 ST\_SkewY

**ST\_SkewY** — Y skew of a 2D geometry. The skew is defined as the angle between the X and Y axes of the geometry's local coordinate system. The skew is measured in degrees.

### Synopsis

float8 **ST\_SkewY**(raster rast);

**ST\_SkewY**

Y skew of a 2D geometry. The skew is defined as the angle between the X and Y axes of the geometry's local coordinate system. The skew is measured in degrees.

**ST\_SkewY**

```

SELECT rid, ST_SkewX(rast) As skewx, ST_SkewY(rast) As skewy,
       ST_GeoReference(rast) as georef
FROM dummy_rast;

```

| rid | skewx | skewy | georef  |
|-----|-------|-------|---|
| 1   | 0     | 0     | 2.0000000000<br>: 0.0000000000<br>: 0.0000000000<br>: 3.0000000000<br>: 0.5000000000<br>: 0.5000000000<br>:         |
| 2   | 0     | 0     | 0.0500000000<br>: 0.0000000000<br>: 0.0000000000<br>: -0.0500000000<br>: 3427927.7500000000<br>: 5793244.0000000000 |

**ST\_SkewX**

[ST\\_GeoReference](#), [ST\\_SkewX](#), [ST\\_SetSkew](#)

## 12.4.17 ST\_SRID

**ST\_SRID** — spatial\_ref\_sys

## Synopsis

integer **ST\_SRID**(raster rast);

### Notes

spatial\_ref\_sys table; SRID; -1; 0



#### Note

PostGIS 2.0; SRID; -1; 0

### Example

```
SELECT ST_SRID(rast) As srid
FROM dummy_rast WHERE rid=1;
```

```
srid
-----
0
```

### References

Section 4.5, **ST\_SRID**

## 12.4.18 ST\_Summary

**ST\_Summary** — Returns a text summary of the raster's metadata.

## Synopsis

text **ST\_Summary**(raster rast);

### Notes

2.1.0; SRID; -1; 0

### Example

```
SELECT ST_Summary(
  ST_AddBand(
    ST_AddBand(
      ST_AddBand(
        ST_MakeEmptyRaster(10, 10, 0, 0, 1, -1, 0, 0, 0)
```

```

        , 1, '8BUI', 1, 0
    )
    , 2, '32BF', 0, -9999
)
, 3, '16BSI', 0, NULL
)
);

-----
st_summary
-----
Raster of 10x10 pixels has 3 bands and extent of BOX(0 -10,10 0)+
band 1 of pixtype 8BUI is in-db with NODATA value of 0      +
band 2 of pixtype 32BF is in-db with NODATA value of -9999 +
band 3 of pixtype 16BSI is in-db with no NODATA value
(1 row)

```

**ST\_MetaData**

[ST\\_MetaData](#), [ST\\_BandMetaData](#), [ST\\_Summary](#) [ST\\_Extent](#)

### 12.4.19 ST\_UpperLeftX

**ST\_UpperLeftX** — X coordinate of the upper-left corner of the raster.

#### Synopsis

```
float8 ST_UpperLeftX(raster rast);
```

**Parameters**

**rast** raster: The raster to be processed.

**Returns**

```
SELECT rid, ST_UpperLeftX(rast) As ulx
FROM dummy_rast;
```

```

rid |      ulx
-----+-----
  1 |      0.5
  2 | 3427927.75

```

**ST\_UpperLeftY**

[ST\\_UpperLeftY](#), [ST\\_GeoReference](#), [Box3D](#)

### 12.4.20 ST\_UpperLeftY

**ST\_UpperLeftY** — Y coordinate of the upper-left corner of the raster.

## Synopsis

float8 **ST\_UpperLeftY**(raster rast);

**&#xc124;&#xba85;**

&#xb798;&#xc2a4;&#xd130;&#xc758; &#xc88c;&#xc0c1;&#xb2e8; Y &#xc88c;&#xd45c;&#xb97c; &#xd22c;&#xc601;&#xb41c; &#xacf5;&#xac04; &#xcc38;&#xc870; &#xb2e8;&#xc704;&#xb85c; &#xbc18;&#xd658;&#xd569;&#xb2c8;&#xb2e4;.

**&#xc608;&#xc2dc;**

```
SELECT rid, ST_UpperLeftY(rast) As uly
FROM dummy_rast;
```

```
rid |    uly
-----+-----
  1 |     0.5
  2 | 5793244
```

**&#xcc38;&#xace0;**

[ST\\_UpperLeftX](#), [ST\\_GeoReference](#), [Box3D](#)

## 12.4.21 ST\_Width

**ST\_Width** — &#xb798;&#xc2a4;&#xd130;&#xc758; &#xb108;&#xbe44;&#xb97c; &#xd53d;&#xc140; &#xac1c;&#xc218;&#xb85c; &#xbc18;&#xd658;&#xd569;&#xb2c8;&#xb2e4;.

## Synopsis

integer **ST\_Width**(raster rast);

**&#xc124;&#xba85;**

&#xb798;&#xc2a4;&#xd130;&#xc758; &#xb108;&#xbe44;&#xb97c; &#xd53d;&#xc140; &#xac1c;&#xc218;&#xb85c; &#xbc18;&#xd658;&#xd569;&#xb2c8;&#xb2e4;.

**&#xc608;&#xc2dc;**

```
SELECT ST_Width(rast) As rastwidth
FROM dummy_rast WHERE rid=1;
```

```
rastwidth
-----
10
```

**&#xcc38;&#xace0;**

[ST\\_Height](#)



## Synopsis

integer **ST\_WorldToRasterCoordX**(raster rast, geometry pt);  
integer **ST\_WorldToRasterCoordX**(raster rast, double precision xw);  
integer **ST\_WorldToRasterCoordX**(raster rast, double precision xw, double precision yw);

### ST\_WorldToRasterCoordX

**ST\_WorldToRasterCoordX**(rast, pt) returns the x coordinate of the point in the raster coordinate system. The raster coordinate system is defined by the raster's `Srid` and `Scale` properties. The raster coordinate system is defined by the raster's `Srid` and `Scale` properties. The raster coordinate system is defined by the raster's `Srid` and `Scale` properties.

**ST\_WorldToRasterCoordX**(rast, xw) returns the x coordinate of the point in the raster coordinate system. The raster coordinate system is defined by the raster's `Srid` and `Scale` properties.

### ST\_WorldToRasterCoordX

```
SELECT rid, ST_WorldToRasterCoordX(rast,3427927.8) As xcoord,
       ST_WorldToRasterCoordX(rast,3427927.8,20.5) As xcoord_xwyw,
       ST_WorldToRasterCoordX(rast,ST_GeomFromText('POINT(3427927.8 20.5)',ST_SRID ←
       (rast))) As ptxcoord
FROM dummy_rast;
```

| rid | xcoord  | xcoord_xwyw | ptxcoord |
|-----|---------|-------------|----------|
| 1   | 1713964 | 1713964     | 1713964  |
| 2   | 1       | 1           | 1        |

### ST\_RasterToWorldCoordX

**ST\_RasterToWorldCoordX**, **ST\_RasterToWorldCoordY**, **ST\_SRID**

## 12.4.24 ST\_WorldToRasterCoordY

**ST\_WorldToRasterCoordY**(rast, pt) returns the y coordinate of the point in the raster coordinate system. The raster coordinate system is defined by the raster's `Srid` and `Scale` properties. The raster coordinate system is defined by the raster's `Srid` and `Scale` properties. The raster coordinate system is defined by the raster's `Srid` and `Scale` properties.

## Synopsis

integer **ST\_WorldToRasterCoordY**(raster rast, geometry pt);  
integer **ST\_WorldToRasterCoordY**(raster rast, double precision xw);  
integer **ST\_WorldToRasterCoordY**(raster rast, double precision xw, double precision yw);

**ST\_RasterToWorldCoordY**

ST\_RasterToWorldCoordY(rast, ycoord) — Returns the world coordinate Y of the raster cell at the given row and column. The raster must be in SRID 4326. The function returns a double precision floating point number. The function is commutative with ST\_RasterToWorldCoordX. The function is commutative with ST\_RasterToWorldCoordZ. The function is commutative with ST\_RasterToWorldCoordM. The function is commutative with ST\_RasterToWorldCoordS. The function is commutative with ST\_RasterToWorldCoordT. The function is commutative with ST\_RasterToWorldCoordU. The function is commutative with ST\_RasterToWorldCoordV. The function is commutative with ST\_RasterToWorldCoordW. The function is commutative with ST\_RasterToWorldCoordX. The function is commutative with ST\_RasterToWorldCoordY. The function is commutative with ST\_RasterToWorldCoordZ. The function is commutative with ST\_RasterToWorldCoordM. The function is commutative with ST\_RasterToWorldCoordS. The function is commutative with ST\_RasterToWorldCoordT. The function is commutative with ST\_RasterToWorldCoordU. The function is commutative with ST\_RasterToWorldCoordV. The function is commutative with ST\_RasterToWorldCoordW.

ST\_RasterToWorldCoordY(rast, ycoord, srid) — Returns the world coordinate Y of the raster cell at the given row and column in the given SRID. The function returns a double precision floating point number. The function is commutative with ST\_RasterToWorldCoordX. The function is commutative with ST\_RasterToWorldCoordZ. The function is commutative with ST\_RasterToWorldCoordM. The function is commutative with ST\_RasterToWorldCoordS. The function is commutative with ST\_RasterToWorldCoordT. The function is commutative with ST\_RasterToWorldCoordU. The function is commutative with ST\_RasterToWorldCoordV. The function is commutative with ST\_RasterToWorldCoordW.

**ST\_RasterToWorldCoordZ**

```
SELECT rid, ST_WorldToRasterCoordY(rast,20.5) As ycoord,
       ST_WorldToRasterCoordY(rast,3427927.8,20.5) As ycoord_xwyw,
       ST_WorldToRasterCoordY(rast,ST_GeomFromText('POINT(3427927.8 20.5)',ST_SRID ←
       (rast))) As ptycoord
FROM dummy_rast;
```

| rid | ycoord    | ycoord_xwyw | ptycoord  |
|-----|-----------|-------------|-----------|
| 1   | 7         | 7           | 7         |
| 2   | 115864471 | 115864471   | 115864471 |

**ST\_RasterToWorldCoordM**

[ST\\_RasterToWorldCoordX](#), [ST\\_RasterToWorldCoordY](#), [ST\\_SRID](#)

**12.5 ST\_RasterToWorldCoordZ****12.5.1 ST\_BandMetaData**

ST\_BandMetaData(rast, band) — Returns the basic meta data about a raster band. The function returns a table with the following columns: pixeltype, nodatavalue, isoutdb, path, outdbbandnum, filesize, filetimestamp. The function is commutative with ST\_BandMetaData(rast, integer[] band).

**Synopsis**

- record `ST_BandMetaData(raster rast, integer band=1);`
- record `ST_BandMetaData(raster rast, integer[] band);`

**ST\_BandMetaData**

Returns basic meta data about a raster band. Columns returned: pixeltype, nodatavalue, isoutdb, path, outdbbandnum, filesize, filetimestamp.

**Note**

If `isoutdb` is `False`, `path`, `outdbbandnum`, `filesize` and `filetimestamp` are `NULL`. If `outdb` access is disabled, `filesize` and `filetimestamp` will also be `NULL`.

**Note**

If band has no `NODATA` value, `nodatavalue` are `NULL`.

**Note**

If `isoutdb` is `False`, `path`, `outdbbandnum`, `filesize` and `filetimestamp` are `NULL`. If `outdb` access is disabled, `filesize` and `filetimestamp` will also be `NULL`.

Enhanced: 2.5.0 to include `outdbbandnum`, `filesize` and `filetimestamp` for `outdb` rasters.

**Example 1**

```
-- Example 1
SELECT
    rid,
    (ST_RasterToWorldCoord(rast, 1, 1)).*,
    (ST_RasterToWorldCoord(rast, 2, 3)).*
FROM (
    SELECT
        rid,
        ST_SetSkew(rast, 100.5, 0) As rast
    FROM dummy_rast
) As foo

rid | longitude | latitude | longitude | latitude
---+-----+-----+-----+-----
1 | 0.5 | 0.5 | 203.5 | 6.5
2 | 3427927.75 | 5793244 | 3428128.8 | 5793243.9
```

**Example 2**

```
WITH foo AS (
    SELECT
        ST_AddBand(NULL::raster, '/home/pele/devel/geo/postgis-git/raster/test/regress/
        loader/Projected.tif', NULL::int[]) AS rast
)
SELECT
    *
FROM ST_BandMetadata(
    (SELECT rast FROM foo),
    ARRAY[1,3,2]::int[]
);

bandnum | pixeltype | nodatavalue | isoutdb | path ←
        | outdbbandnum | filesize | filetimestamp |
```







**ST\_BandFileSize**

### 12.5.5 ST\_BandFileSize

**ST\_BandFileSize** — Returns the file size of a band stored in file system. If no bandnum specified, 1 is assumed.

#### Synopsis

**bigint ST\_BandFileSize**(raster rast, integer bandnum=1);

**ST\_BandFileSize**

Returns the file size of a band stored in file system. Throws an error if called with an in db band, or if outdb access is not enabled.

This function is typically used in conjunction with **ST\_BandPath()** and **ST\_BandFileTimestamp()** so a client can determine if the filename of a outdb raster as seen by it is the same as the one seen by the server.

Availability: 2.5.0

**ST\_BandFileSize**

```
SELECT ST_BandFileSize(rast,1) FROM dummy_rast WHERE rid = 1;
```

```
st_bandfilesize
-----
                240574
```

### 12.5.6 ST\_BandFileTimestamp

**ST\_BandFileTimestamp** — Returns the file timestamp of a band stored in file system. If no bandnum specified, 1 is assumed.

#### Synopsis

**bigint ST\_BandFileTimestamp**(raster rast, integer bandnum=1);

**ST\_BandFileTimestamp**

Returns the file timestamp (number of seconds since Jan 1st 1970 00:00:00 UTC) of a band stored in file system. Throws an error if called with an in db band, or if outdb access is not enabled.

This function is typically used in conjunction with **ST\_BandPath()** and **ST\_BandFileSize()** so a client can determine if the filename of a outdb raster as seen by it is the same as the one seen by the server.

Availability: 2.5.0

**ST\_BandFileTimestamp**

```
SELECT ST_BandFileTimestamp(rast,1) FROM dummy_rast WHERE rid = 1;
```

```
st_bandfiletimestamp
-----
                1521807257
```

## 12.5.7 ST\_BandPixelType

`ST_BandPixelType` — Returns name describing data type and size of values stored in each cell of given band.

### Synopsis

text `ST_BandPixelType`(raster rast, integer bandnum=1);

### Parameters

Returns name describing data type and size of values stored in each cell of given band.

bandnum — Band number to query. Default is 1.

- 1BB - 1 byte, 1 band
- 2BUI - 2 bytes, unsigned integer, 1 band
- 4BUI - 4 bytes, unsigned integer, 1 band
- 8BSI - 8 bytes, signed integer, 1 band
- 8BUI - 8 bytes, unsigned integer, 1 band
- 16BSI - 16 bytes, signed integer, 1 band
- 16BUI - 16 bytes, unsigned integer, 1 band
- 32BSI - 32 bytes, signed integer, 1 band
- 32BUI - 32 bytes, unsigned integer, 1 band
- 32BF - 32 bytes, float, 1 band
- 64BF - 64 bytes, float, 1 band

### Examples

```
SELECT ST_BandPixelType(rast,1) As btype1,
       ST_BandPixelType(rast,2) As btype2, ST_BandPixelType(rast,3) As btype3
FROM dummy_rast
WHERE rid = 2;
```

```
btype1 | btype2 | btype3
-----+-----+-----
8BUI   | 8BUI   | 8BUI
```

### See Also

[ST\\_NumBands](#)

## 12.5.8 ST\_PixelOfValue

`ST_PixelOfValue` — Returns pixel value of given band.

## Synopsis

integer **ST\_MemSize**(raster rast);

**&#xc124;&#xba85;**

&#xb798;&#xc2a4;&#xd130; &#xac1d;&#xccb4; &#xb0b4;&#xbd80;&#xc5d0; &#xc788;&#xb294; &#xbc34;&#xb4dc;&#xb4e4;&#xac1c;&#xc218;&#xb97c; &#xbc18;&#xd658;&#xd569;&#xb2c8;&#xb2e4;.

**&#xc608;&#xc2dc;**

```
SELECT ST_MinPossibleValue('16BSI');
```

```
  st_minpossiblevalue
-----
                -32768
```

```
SELECT ST_MinPossibleValue('8BUI');
```

```
  st_minpossiblevalue
-----
                    0
```

**&#xcc38;&#xace0;**

**ST\_BandPixelType**

## 12.5.9 ST\_HasNoBand

**ST\_HasNoBand** — &#xc785;&#xb825;&#xb41c; &#xbc34;&#xb4dc; &#xbc88;&#xd638;&#xc5d0; &#xbc34;&#xb4dc;&#xac00; &#xc5c6;&#xc744; &#xacbd;&#xc6b0; &#xcc38;&#xc744; &#xbc18;&#xd658;&#xd569;&#xb2c8;&#xb2e4;. &#xbc34;&#xb4dc; &#xbc88;&#xd638;&#xb97c; &#xc124;&#xc815;&#xd558;&#xc9c0; &#xc54a;&#xc744; &#xacbd;&#xc6b0;, &#xbc34;&#xb4dc; 1&#xb85c; &#xac00;&#xc815;&#xd569;&#xb2c8;&#xb2e4;.

## Synopsis

boolean **ST\_HasNoBand**(raster rast, integer bandnum=1);

**&#xc124;&#xba85;**

&#xc785;&#xb825;&#xb41c; &#xbc34;&#xb4dc; &#xbc88;&#xd638;&#xc5d0; &#xbc34;&#xb4dc;&#xac00; &#xc5c6;&#xc744; &#xacbd;&#xc6b0; &#xcc38;&#xc744; &#xbc18;&#xd658;&#xd569;&#xb2c8;&#xb2e4;. &#xbc34;&#xb4dc; &#xbc88;&#xd638;&#xc124;&#xc815;&#xd558;&#xc9c0; &#xc54a;&#xc744; &#xacbd;&#xc6b0;, &#xbc34;&#xb4dc; 1&#xb85c; &#xac00;&#xc815;&#xd569;&#xb2c8;&#xb2e4;.

2.0.0 &#xbc84;&#xc804;&#xbd80;&#xd130; &#xc0ac;&#xc6a9;&#xd560; &#xc218; &#xc788;&#xc2b5;&#xb2c8;&#xb2e4;.

**&#xc608;&#xc2dc;**

```
SELECT rid, ST_HasNoBand(rast) As hb1, ST_HasNoBand(rast,2) as hb2,
ST_HasNoBand(rast,4) as hb4, ST_NumBands(rast) As numbands
FROM dummy_rast;
```

| rid | hb1 | hb2 | hb4 | numbands |
|-----|-----|-----|-----|----------|
| 1   | t   | t   | t   | 0        |
| 2   | f   | f   | t   | 3        |

**ST\_NumBands**

[ST\\_NumBands](#)

## 12.6 ST\_PixelAsPolygon

### 12.6.1 ST\_PixelAsPolygon

**ST\_PixelAsPolygon** — Returns a polygon geometry for each pixel in a raster. The geometry is defined by the pixel's bounding box and the pixel's value.

#### Synopsis

geometry **ST\_PixelAsPolygon**(raster rast, integer columnx, integer rowy);

**ST\_PixelAsPolygon**

Returns a polygon geometry for each pixel in a raster. The geometry is defined by the pixel's bounding box and the pixel's value.

2.0.0 Returns a polygon geometry for each pixel in a raster. The geometry is defined by the pixel's bounding box and the pixel's value.

**ST\_PixelAsPolygon**

```
-- ST_PixelAsPolygon; ST_PixelAsPolygons; ST_PixelAsPoint; ST_PixelAsPoints; ST_PixelAsCentroid; ST_PixelAsCentroids;
SELECT i, j, ST_AsText(ST_PixelAsPolygon(foo.rast, i, j)) As blpgeom
FROM dummy_rast As foo
      CROSS JOIN generate_series(1,2) As i
      CROSS JOIN generate_series(1,1) As j
WHERE rid=2;
```

| i | j | blpgeom   |
|---|---|---|
| 1 | 1 | POLYGON((3427927.75 5793244,3427927.8 5793244,3427927.8 5793243.95,...  |
| 2 | 1 | POLYGON((3427927.8 5793244,3427927.85 5793244,3427927.85 5793243.95, .. |

**ST\_PixelAsPolygon**

[ST\\_DumpAsPolygons](#), [ST\\_PixelAsPolygons](#), [ST\\_PixelAsPoint](#), [ST\\_PixelAsPoints](#), [ST\\_PixelAsCentroid](#), [ST\\_PixelAsCentroids](#), [ST\\_Intersection](#), [ST\\_AsText](#)

## 12.6.2 ST\_PixelAsPolygons

`ST_PixelAsPolygons` — Returns a set of records representing the pixels of a raster as polygons. The returned polygons are in the same order as the pixels in the raster. The polygons are in the same order as the pixels in the raster. The polygons are in the same order as the pixels in the raster.

### Synopsis

setof record `ST_PixelAsPolygons`(raster rast, integer band=1, boolean exclude\_nodata\_value=TRUE);

### Parameters

`rast` raster: The raster to be processed.  
`band` integer: The band number to be processed.  
`exclude_nodata_value` boolean: If TRUE, only those pixels whose values are not NODATA are returned as points.

Return record format: *geom* geometry, *val* double precision, *x* integer, *y* integers.



#### Note

When `exclude_nodata_value = TRUE`, only those pixels whose values are not NODATA are returned as points.



#### Note

`ST_PixelAsPolygons` returns a set of records representing the pixels of a raster as polygons. The returned polygons are in the same order as the pixels in the raster. The polygons are in the same order as the pixels in the raster. The polygons are in the same order as the pixels in the raster.

2.0.0 Returns a set of records representing the pixels of a raster as polygons. The returned polygons are in the same order as the pixels in the raster. The polygons are in the same order as the pixels in the raster. The polygons are in the same order as the pixels in the raster.

2.1.0 Returns a set of records representing the pixels of a raster as polygons. The returned polygons are in the same order as the pixels in the raster. The polygons are in the same order as the pixels in the raster. The polygons are in the same order as the pixels in the raster.

2.1.1 Returns a set of records representing the pixels of a raster as polygons. The returned polygons are in the same order as the pixels in the raster. The polygons are in the same order as the pixels in the raster. The polygons are in the same order as the pixels in the raster.

### Examples

```
-- Returns a set of records representing the pixels of a raster as polygons.
SELECT (gv).x, (gv).y, (gv).val, ST_AsText((gv).geom) geom
FROM (SELECT ST_PixelAsPolygons (
    ST_SetValue(ST_SetValue(ST_AddBand(ST_MakeEmptyRaster(2, 2, 0, 0, 0.001,
    -0.001, 0.001, 0.001, 4269),
    '8BUI'::text, 1, 0),
    2, 2, 10),
    1, 1, NULL)
) gv
) foo;

x | y | val |
geom
```



**ST\_AsText**

ST\_AsText(*geom*) returns the text representation of the geometry. The text representation is a string of characters that can be used to store the geometry in a text field. The text representation is a string of characters that can be used to store the geometry in a text field. The text representation is a string of characters that can be used to store the geometry in a text field.

Return record format: *geom* geometry, *val* double precision, *x* integer, *y* integers.

**Note**

When `exclude_nodata_value = TRUE`, only those pixels whose values are not NODATA are returned as points.

2.1.0 ST\_AsText(*geom*, *val*, *x*, *y*): 2.1.1 ST\_AsText(*geom*, *val*, *x*, *y*, *exclude\_nodata\_value*):

2.1.1 ST\_AsText(*geom*, *val*, *x*, *y*, *exclude\_nodata\_value*): 2.1.1 ST\_AsText(*geom*, *val*, *x*, *y*, *exclude\_nodata\_value*):

**ST\_DumpAsPolygons**

```
SELECT x, y, val, ST_AsText(geom) FROM (SELECT (ST_PixelAsPoints(rast, 1)).* FROM
dummy_rast WHERE rid = 2) foo;
```

| x | y | val | st_astext                    |
|---|---|-----|------------------------------|
| 1 | 1 | 253 | POINT(3427927.75 5793244)    |
| 2 | 1 | 254 | POINT(3427927.8 5793244)     |
| 3 | 1 | 253 | POINT(3427927.85 5793244)    |
| 4 | 1 | 254 | POINT(3427927.9 5793244)     |
| 5 | 1 | 254 | POINT(3427927.95 5793244)    |
| 1 | 2 | 253 | POINT(3427927.75 5793243.95) |
| 2 | 2 | 254 | POINT(3427927.8 5793243.95)  |
| 3 | 2 | 254 | POINT(3427927.85 5793243.95) |
| 4 | 2 | 253 | POINT(3427927.9 5793243.95)  |
| 5 | 2 | 249 | POINT(3427927.95 5793243.95) |
| 1 | 3 | 250 | POINT(3427927.75 5793243.9)  |
| 2 | 3 | 254 | POINT(3427927.8 5793243.9)   |
| 3 | 3 | 254 | POINT(3427927.85 5793243.9)  |
| 4 | 3 | 252 | POINT(3427927.9 5793243.9)   |
| 5 | 3 | 249 | POINT(3427927.95 5793243.9)  |
| 1 | 4 | 251 | POINT(3427927.75 5793243.85) |
| 2 | 4 | 253 | POINT(3427927.8 5793243.85)  |
| 3 | 4 | 254 | POINT(3427927.85 5793243.85) |
| 4 | 4 | 254 | POINT(3427927.9 5793243.85)  |
| 5 | 4 | 253 | POINT(3427927.95 5793243.85) |
| 1 | 5 | 252 | POINT(3427927.75 5793243.8)  |
| 2 | 5 | 250 | POINT(3427927.8 5793243.8)   |
| 3 | 5 | 254 | POINT(3427927.85 5793243.8)  |
| 4 | 5 | 254 | POINT(3427927.9 5793243.8)   |
| 5 | 5 | 254 | POINT(3427927.95 5793243.8)  |

**ST\_DumpAsPolygons**

ST\_DumpAsPolygons, ST\_PixelAsPolygon, ST\_PixelAsPolygons, ST\_PixelAsPoint, ST\_PixelAsCentroid, ST\_PixelAsCentroids

## 12.6.5 ST\_PixelAsCentroid

`ST_PixelAsCentroid` — Returns the centroid of a pixel in a raster.

### Synopsis

geometry `ST_PixelAsCentroid`(raster rast, integer x, integer y);

#### Parameters

`rast`: raster to process.  
`x`: x coordinate of the pixel.  
`y`: y coordinate of the pixel.

2.1.0: Returns the centroid of a pixel in a raster. The centroid is the center of mass of the pixel.

2.1.0: Returns the centroid of a pixel in a raster. The centroid is the center of mass of the pixel.

#### Example

```
SELECT ST_AsText(ST_PixelAsCentroid(rast, 1, 1)) FROM dummy_rast WHERE rid = 1;
```

```
st_astext
-----
POINT(1.5 2)
```

#### See Also

[ST\\_DumpAsPolygons](#), [ST\\_PixelAsPolygon](#), [ST\\_PixelAsPolygons](#), [ST\\_PixelAsPoint](#), [ST\\_PixelAsPoints](#), [ST\\_PixelAsCentroids](#)

## 12.6.6 ST\_PixelAsCentroids

`ST_PixelAsCentroids` — Returns the centroids of all pixels in a raster.

### Synopsis

setof record `ST_PixelAsCentroids`(raster rast, integer band=1, boolean exclude\_nodata\_value=TRUE);

#### Parameters

`rast`: raster to process.  
`band`: band number to process.  
`exclude_nodata_value`: boolean flag to exclude no data values.

Return record format: `geom geometry`, `val double precision`, `x integer`, `y integers`.



```

ST_Value(rast, geometry pt, boolean exclude_nodata_value=true);
ST_Value(rast, integer band, geometry pt, boolean exclude_nodata_value=true);
ST_Value(rast, integer x, integer y, boolean exclude_nodata_value=true);
ST_Value(rast, integer band, integer x, integer y, boolean exclude_nodata_value=true);

```

## Synopsis

```

double precision ST_Value(rast, geometry pt, boolean exclude_nodata_value=true);
double precision ST_Value(rast, integer band, geometry pt, boolean exclude_nodata_value=true);
double precision ST_Value(rast, integer x, integer y, boolean exclude_nodata_value=true);
double precision ST_Value(rast, integer band, integer x, integer y, boolean exclude_nodata_value=true);

```

### Parameters

```

rast: raster to be sampled.
geometry pt: point geometry to sample.
integer band: band number to sample.
integer x, integer y: x and y coordinates to sample.
boolean exclude_nodata_value: if true, nodata values are excluded.
integer band, integer x, integer y: band number, x and y coordinates to sample.

```

The allowed values of the `resample` parameter are "nearest" which performs the default nearest-neighbor resampling, and "bilinear" which performs a [bilinear interpolation](#) to estimate the value between pixel centers.

```

resample: 2.1.0 nearest;
exclude_nodata_value: true;

```

```

resample: 2.0.0 bilinear;
exclude_nodata_value: true;

```

### Examples

```

-- Create a raster and a point geometry.
CREATE TABLE dummy_rast (rid integer);
INSERT INTO dummy_rast VALUES (2);
CREATE TABLE foo (rid integer, pt geometry);
INSERT INTO foo VALUES (2, ST_Point(342792.77, 5793243.76));
-- Create a raster with a value of 252 at the point location.
CREATE TABLE sometable (rid integer, geom geometry);
INSERT INTO sometable VALUES (2, ST_GeomFromText('POINT(342792.77 5793243.76)'));
-- Query to get the value of the raster at the point location.
SELECT rid, ST_Value(rast, foo.pt_geom) As b1pval, ST_Value(rast, 2, foo.pt_geom) As b2pval
FROM dummy_rast CROSS JOIN (SELECT ST_SetSRID(ST_Point(342792.77, 5793243.76), 0) As
pt_geom) As foo
WHERE rid=2;

rid | b1pval | b2pval
----+-----+-----
  2 |    252 |     79

-- Query to get the value of the raster at the point location using a different SRID.
SELECT rid, ST_Value(rast, 3, sometable.geom) As b3pval
FROM sometable
WHERE ST_Intersects(rast, sometable.geom);

```

```
SELECT rid, ST_Value(rast, 1, 1, 1) As b1pval,
       ST_Value(rast, 2, 1, 1) As b2pval, ST_Value(rast, 3, 1, 1) As b3pval
FROM dummy_rast
WHERE rid=2;
```

| rid | b1pval | b2pval | b3pval |
|-----|--------|--------|--------|
| 2   | 253    | 78     | 70     |

```
-- &#xc01; &#xd53d;&#xc140;&#xc758; &#xbc34;&#xb4dc; 1, 2, 3&#xc758; &#xaa8;&#xb4e0; &#xc12;&#xb4e4;&#xc744; &#xc5bb;&#xc2b5;&#xb2c8;&#xb2e4;. --
SELECT x, y, ST_Value(rast, 1, x, y) As b1val,
       ST_Value(rast, 2, x, y) As b2val, ST_Value(rast, 3, x, y) As b3val
FROM dummy_rast CROSS JOIN
generate_series(1, 1000) As x CROSS JOIN generate_series(1, 1000) As y
WHERE rid = 2 AND x <= ST_Width(rast) AND y <= ST_Height(rast);
```

| x | y | b1val | b2val | b3val |
|---|---|-------|-------|-------|
| 1 | 1 | 253   | 78    | 70    |
| 1 | 2 | 253   | 96    | 80    |
| 1 | 3 | 250   | 99    | 90    |
| 1 | 4 | 251   | 89    | 77    |
| 1 | 5 | 252   | 79    | 62    |
| 2 | 1 | 254   | 98    | 86    |
| 2 | 2 | 254   | 118   | 108   |
| : |   |       |       |       |
| : |   |       |       |       |

```
-- &#xc55e;&#xc758; &#xc608;&#xc2dc;&#xcc98;&#xb7fc; &#xc01; &#xd53d;&#xc140;&#xc758; &#xbc34;&#xb4dc; 1, 2, 3&#xc758; &#xaa8;&#xb4e0; &#xc12;&#xb4e4;&#xc744; &#xc5bb;&#xc9c0;&#xb9cc; &#xc01; &#xd53d;&#xc140;&#xc758; &#xc88c;&#xc0c1;&#xb2e8; &#xd3ec;&#xc778;&#xd2b8;&#xb3c4; &#xbc18;&#xd658;&#xd569;&#xb2c8;&#xb2e4;. --
SELECT ST_AsText(ST_SetSRID(
       ST_Point(ST_UpperLeftX(rast) + ST_ScaleX(rast)*x,
               ST_UpperLeftY(rast) + ST_ScaleY(rast)*y),
               ST_SRID(rast))) As uplpt
, ST_Value(rast, 1, x, y) As b1val,
  ST_Value(rast, 2, x, y) As b2val, ST_Value(rast, 3, x, y) As b3val
FROM dummy_rast CROSS JOIN
generate_series(1,1000) As x CROSS JOIN generate_series(1,1000) As y
WHERE rid = 2 AND x <= ST_Width(rast) AND y <= ST_Height(rast);
```

| uplpt                       | b1val | b2val | b3val |
|-----------------------------|-------|-------|-------|
| POINT(3427929.25 5793245.5) | 253   | 78    | 70    |
| POINT(3427929.25 5793247)   | 253   | 96    | 80    |
| POINT(3427929.25 5793248.5) | 250   | 99    | 90    |
| :                           |       |       |       |

```
-- &#xd2b9;&#xc815; &#xc12; &#xbc94;&#xc704;&#xc5d0; &#xb4e4;&#xc5b4;&#xc624;&#xba70; &#xd2b9;&#xc815; &#xd3f4;&#xb9ac;&#xace4;&#xcfc; &#xad50;&#xcc28;&#xd558;&#xb294; &#xaa8;&#xb4e0; &#xd53d;&#xc140;&#xb4e4;&#xc758; &#xd569;&#xc9d1;&#xd569;&#xc73c;&#xb85c; &#xc774;&#xb8e8;&#xc5b4;&#xc9c4; &#xd3f4;&#xb9ac;&#xace4;&#xc744; &#xc5bb;&#xc2b5;&#xb2c8;&#xb2e4;. --
SELECT ST_AsText(ST_Union(pixpolyg)) As shadow
FROM (SELECT ST_Translate(ST_MakeEnvelope(
       ST_UpperLeftX(rast), ST_UpperLeftY(rast),
       ST_UpperLeftX(rast) + ST_ScaleX(rast),
       ST_UpperLeftY(rast) + ST_ScaleY(rast), 0
```

```

        ), ST_ScaleX(rast)*x, ST_ScaleY(rast)*y
    ) As pixpolyg, ST_Value(rast, 2, x, y) As b2val
FROM dummy_rast CROSS JOIN
generate_series(1,1000) As x CROSS JOIN generate_series(1,1000) As y
WHERE rid = 2
    AND x <= ST_Width(rast) AND y <= ST_Height(rast)) As foo
WHERE
    ST_Intersects(
        pixpolyg,
        ST_GeomFromText('POLYGON((3427928 5793244,3427927.75 5793243.75,3427928
        5793243.75,3427928 5793244))',0)
    ) AND b2val != 254;

-----
shadow
-----
MULTIPOLYGON(((3427928 5793243.9,3427928 5793243.85,3427927.95 5793243.85,3427927.95
5793243.9,
3427927.95 5793243.95,3427928 5793243.95,3427928.05 5793243.95,3427928.05
5793243.9,3427928 5793243.9)),((3427927.95 5793243.9,3427927.95 579324
3.85,3427927.9 5793243.85,3427927.85 5793243.85,3427927.85 5793243.9,3427927.9
5793243.9,3427927.9 5793243.95,
3427927.95 5793243.95,3427927.95 5793243.9)),((3427927.85 5793243.75,3427927.85
5793243.7,3427927.8 5793243.7,3427927.8 5793243.75
,3427927.8 5793243.8,3427927.8 5793243.85,3427927.85 5793243.85,3427927.85
5793243.8,3427927.85 5793243.75)),
((3427928.05 5793243.75,3427928.05 5793243.7,3427928 5793243.7,3427927.95
5793243.7,3427927.95 5793243.75,3427927.95 5793243.8,3427
927.95 5793243.85,3427928 5793243.85,3427928 5793243.8,3427928.05 5793243.8,
3427928.05 5793243.75)),((3427927.95 5793243.75,3427927.95 5793243.7,3427927.9
5793243.7,3427927.85 5793243.7,
3427927.85 5793243.75,3427927.85 5793243.8,3427927.85 5793243.85,3427927.9 5793243.85,
3427927.95 5793243.85,3427927.95 5793243.8,3427927.95 5793243.75)))

-- &#xb300;&#xc6a9;&#xb7c9; &#xb798;&#xc2a4;&#xd130; &#xd0c0;&#xc77c;&#xc758;
&#xbaa8;&#xb4e0; &#xd53d;&#xc140;&#xc744; &#xd655;&#xc778;&#xd558;&#xb294;
&#xc791;&#xc5c5;&#xc740; &#xc624;&#xb798; &#xac78;&#xb9b4; &#xc218;
&#xc788;&#xc2b5;&#xb2c8;&#xb2e4;.
-- &#xb2e8;&#xacc4;&#xbcc4;&#xb85c; &#xc120;&#xd0dd;&#xd560; &#xc218; &#xc788;&#xb294;
generate_series &#xd30c;&#xb77c;&#xbbf8;&#xd130;&#xb97c; &#xc774;&#xc6a9;&#xd574;&#xc11c;
-- &#xc790;&#xb9bf;&#xc218;&#xb97c; &#xd1b5;&#xd574; &#xc815;&#xd655;&#xb3c4;&#xb97c;
&#xc870;&#xae08; &#xc904;&#xc774; &#xba74; &#xc18d;&#xb3c4;&#xb97c; &#xd06c;&#xac8c;
&#xd5a5;&#xc0c1;&#xc2dc;&#xd0ac; &#xc218; &#xc788;&#xc2b5;&#xb2c8;&#xb2e4;.
-- &#xb2e4;&#xc74c; &#xc608;&#xc2dc;&#xb294; &#xc774;&#xc804; &#xc608;&#xc2dc;&#xc640;
&#xac19;&#xc740; &#xc791;&#xc5c5;&#xc744; &#xd558;&#xc9c0;&#xb9cc; &#xb9e4; 4(2x2)
&#xd53d;&#xc140;&#xb9c8;&#xb2e4; &#xd53d;&#xc140; 1&#xac1c;&#xb97c;
&#xd655;&#xc778;&#xd574;&#xc11c; &#xd655;&#xc778;&#xd55c; &#xac83;&#xc73c;&#xb85c;
&#xcc98;&#xb9ac;&#xd569;&#xb2c8;&#xb2e4;.
-- &#xd655;&#xc778;&#xb41c; &#xd53d;&#xc140;&#xc758; &#xac12;&#xc744; &#xadf8;
&#xb2e4;&#xc74c; &#xd53d;&#xc140; 4&#xac1c;&#xc758; &#xac12;&#xc73c;&#xb85c;
&#xc800;&#xc7a5;&#xd569;&#xb2c8;&#xb2e4;.

SELECT ST_AsText(ST_Union(pixpolyg)) As shadow
FROM (SELECT ST_Translate(ST_MakeEnvelope(
    ST_UpperLeftX(rast), ST_UpperLeftY(rast),
    ST_UpperLeftX(rast) + ST_ScaleX(rast)*2,
    ST_UpperLeftY(rast) + ST_ScaleY(rast)*2, 0
    ), ST_ScaleX(rast)*x, ST_ScaleY(rast)*y
    ) As pixpolyg, ST_Value(rast, 2, x, y) As b2val
FROM dummy_rast CROSS JOIN
generate_series(1,1000,2) As x CROSS JOIN generate_series(1,1000,2) As y
WHERE rid = 2

```

```

AND x <= ST_Width(rast) AND y <= ST_Height(rast) ) As foo
WHERE
  ST_Intersects(
    pixpolyg,
    ST_GeomFromText('POLYGON((3427928 5793244,3427927.75 5793243.75,3427928 5793243.75,3427928 5793243.75,3427928 5793244))',0)
  ) AND b2val != 254;

-----
MULTIPOLYGON(((3427927.9 5793243.85,3427927.8 5793243.85,3427927.8 5793243.95,
3427927.9 5793243.95,3427928 5793243.95,3427928.1 5793243.95,3427928.1 5793243.85,3427928 5793243.85,3427927.9 5793243.85)),
((3427927.9 5793243.65,3427927.8 5793243.65,3427927.8 5793243.75,3427927.8 5793243.85,3427927.9 5793243.85,
3427928 5793243.85,3427928 5793243.75,3427928.1 5793243.75,3427928.1 5793243.65,3427928 5793243.65,3427927.9 5793243.65)))

```

## ST\_SetValue, ST\_DumpAsPolygons, ST\_NumBands, ST\_PixelAsPolygon, ST\_ScaleX, ST\_ScaleY, ST\_UpperLeftX, ST\_UpperLeftY, ST\_SRID, ST\_AsText, ST\_Point, ST\_MakeEnvelope, ST\_Intersects, ST\_Intersection

[ST\\_SetValue](#), [ST\\_DumpAsPolygons](#), [ST\\_NumBands](#), [ST\\_PixelAsPolygon](#), [ST\\_ScaleX](#), [ST\\_ScaleY](#), [ST\\_UpperLeftX](#), [ST\\_UpperLeftY](#), [ST\\_SRID](#), [ST\\_AsText](#), [ST\\_Point](#), [ST\\_MakeEnvelope](#), [ST\\_Intersects](#), [ST\\_Intersection](#)

### 12.6.8 ST\_NearestValue

**ST\_NearestValue** — columnx, rowy, bandnum, geometry pt, boolean exclude\_nodata\_value=true; columnx, rowy, bandnum, geometry pt, boolean exclude\_nodata\_value=true; columnx, rowy, bandnum, integer columnx, integer rowy, boolean exclude\_nodata\_value=true; columnx, rowy, bandnum, integer columnx, integer rowy, boolean exclude\_nodata\_value=true;

#### Synopsis

double precision **ST\_NearestValue**(raster rast, integer bandnum, geometry pt, boolean exclude\_nodata\_value=true);  
double precision **ST\_NearestValue**(raster rast, geometry pt, boolean exclude\_nodata\_value=true);  
double precision **ST\_NearestValue**(raster rast, integer bandnum, integer columnx, integer rowy, boolean exclude\_nodata\_value=true);  
double precision **ST\_NearestValue**(raster rast, integer columnx, integer rowy, boolean exclude\_nodata\_value=true);

#### ST\_NearestValue(raster rast, integer bandnum, geometry pt, boolean exclude\_nodata\_value=true);

columnx, rowy, bandnum, geometry pt, boolean exclude\_nodata\_value=true; columnx, rowy, bandnum, integer columnx, integer rowy, boolean exclude\_nodata\_value=true; columnx, rowy, bandnum, integer columnx, integer rowy, boolean exclude\_nodata\_value=true; columnx, rowy, bandnum, integer columnx, integer rowy, boolean exclude\_nodata\_value=true;

columnx, rowy, bandnum, integer columnx, integer rowy, boolean exclude\_nodata\_value=true; columnx, rowy, bandnum, integer columnx, integer rowy, boolean exclude\_nodata\_value=true; columnx, rowy, bandnum, integer columnx, integer rowy, boolean exclude\_nodata\_value=true; columnx, rowy, bandnum, integer columnx, integer rowy, boolean exclude\_nodata\_value=true;

2.1.0

**Note**

ST\_NearestValue

**ST\_NearestValue**

```
-- 2x2
SELECT
  ST_Value(rast, 2, 2) AS value,
  ST_NearestValue(rast, 2, 2) AS nearestvalue
FROM (
  SELECT
    ST_SetValue(
      ST_SetValue(
        ST_SetValue(
          ST_SetValue(
            ST_SetValue(
              ST_AddBand(
                ST_MakeEmptyRaster(5, 5,
                  -2, 2, 1, -1, 0, 0, 0),
                '8BUI'::text, 1, 0
              ),
              1, 1, 0.
            ),
            2, 3, 0.
          ),
          3, 5, 0.
        ),
        4, 2, 0.
      ),
      5, 4, 0.
    ) AS rast
) AS foo

value | nearestvalue
-----+-----
1 | 1
```

```
-- 2x3
SELECT
  ST_Value(rast, 2, 3) AS value,
  ST_NearestValue(rast, 2, 3) AS nearestvalue
FROM (
  SELECT
    ST_SetValue(
      ST_SetValue(
        ST_SetValue(
          ST_SetValue(
            ST_SetValue(
              ST_AddBand(
                ST_MakeEmptyRaster(5, 5,
                  -2, 2, 1, -1, 0, 0, 0),
```



```

        skewx => 0, skewy => 0, srid => 4326),
        index => 1, pixeltype => '16BSI',
        initialvalue => 0,
        nodataval => -999),
        1,1,1,
        newvalueset =>ARRAY[ARRAY[10.0::float8, 50.0::float8], ARRAY[40.0::float8, 20.0::float8 ←
        ]] AS rast
    )
SELECT
ST_AsText (
    ST_SetZ (
        rast,
        band => 1,
        geom => 'SRID=4326;LINESTRING(1.0 1.9, 1.0 0.2)::geometry,
        resample => 'bilinear'
    ))
FROM test_raster

        st_astext
-----
LINESTRING Z (1 1.9 38,1 0.2 27)

```

**ST\_SetZ**

[ST\\_Value](#), [ST\\_SetSRID](#)

## 12.6.10 ST\_SetSkew

**ST\_SetSkew** — Returns a geometry with the same X/Y coordinates as the input geometry, and values from the raster copied into the Z dimension using the requested resample algorithm.

### Synopsis

bytea **ST\_AsGDALRaster**(raster rast, text format, text[] options=NULL, integer srid=sameassource);

**ST\_SetValues**

Returns a geometry with the same X/Y coordinates as the input geometry, and values from the raster copied into the Z dimensions using the requested resample algorithm.

The `resample` parameter can be set to "nearest" to copy the values from the cell each vertex falls within, or "bilinear" to use [bilinear interpolation](#) to calculate a value that takes neighboring cells into account also.

2.2.0 [ST\\_SetValues](#)

**ST\_SetValues**

```

--
-- 2x2 test raster with values
--
-- 10 50
-- 40 20
--
WITH test_raster AS (
SELECT
ST_SetValues(

```

```

ST_AddBand(
  ST_MakeEmptyRaster(width => 2, height => 2,
    upperleftx => 0, upperlefty => 2,
    scalex => 1.0, scaley => -1.0,
    skewx => 0, skewy => 0, srid => 4326),
  index => 1, pixeltype => '16BSI',
  initialvalue => 0,
  nodataval => -999),
  1,1,1,
  newvalueset =>ARRAY[ARRAY[10.0::float8, 50.0::float8], ARRAY[40.0::float8, 20.0::float8 ←
  ]]) AS rast
)
SELECT
ST_AsText (
  ST_SetM(
    rast,
    band => 1,
    geom => 'SRID=4326;LINESTRING(1.0 1.9, 1.0 0.2)::geometry,
    resample => 'bilinear'
  ))
FROM test_raster

          st_astext
-----
LINESTRING M (1 1.9 38,1 0.2 27)

```

**ST\_SetSRID**

**ST\_Value**, **ST\_SetSRID**

## 12.6.11 ST\_Neighborhood

**ST\_Neighborhood** — columnx, rowy, bandnum, distanceX, distanceY, exclude\_nodata\_value

**ST\_Neighborhood**(raster rast, integer bandnum, integer columnX, integer rowY, integer distanceX, integer distanceY, boolean exclude\_nodata\_value=true);

**ST\_Neighborhood**(raster rast, integer columnX, integer rowY, integer distanceX, integer distanceY, boolean exclude\_nodata\_value=true);

**ST\_Neighborhood**(raster rast, integer bandnum, geometry pt, integer distanceX, integer distanceY, boolean exclude\_nodata\_value=true);

**ST\_Neighborhood**(raster rast, geometry pt, integer distanceX, integer distanceY, boolean exclude\_nodata\_value=true);

### Synopsis

double precision[][] **ST\_Neighborhood**(raster rast, integer bandnum, integer columnX, integer rowY, integer distanceX, integer distanceY, boolean exclude\_nodata\_value=true);

double precision[][] **ST\_Neighborhood**(raster rast, integer columnX, integer rowY, integer distanceX, integer distanceY, boolean exclude\_nodata\_value=true);

double precision[][] **ST\_Neighborhood**(raster rast, integer bandnum, geometry pt, integer distanceX, integer distanceY, boolean exclude\_nodata\_value=true);

double precision[][] **ST\_Neighborhood**(raster rast, geometry pt, integer distanceX, integer distanceY, boolean exclude\_nodata\_value=true);

**ST\_Neighborhood**

columnx, rowy, bandnum, distanceX, distanceY, exclude\_nodata\_value

**ST\_Neighborhood**(raster rast, integer bandnum, integer columnX, integer rowY, integer distanceX, integer distanceY, boolean exclude\_nodata\_value=true);

**ST\_Neighborhood**(raster rast, integer columnX, integer rowY, integer distanceX, integer distanceY, boolean exclude\_nodata\_value=true);

**ST\_Neighborhood**(raster rast, integer bandnum, geometry pt, integer distanceX, integer distanceY, boolean exclude\_nodata\_value=true);

**ST\_Neighborhood**(raster rast, geometry pt, integer distanceX, integer distanceY, boolean exclude\_nodata\_value=true);

```

distanceX &#xc0f; distanceY &#xd30c;&#xb77c;&#xbbbf8;&#xd130;&#xa
&#xc9c0;&#xc815;&#xd55c;&#xd53d;&#xc140;&#xc8fc;&#xc704;&#xc758;&#xd53d;&#xc140;&#xac1c;&#xc218;&#xb97c;
X &#bc0f; Y &#xcd95;&#xc73c;&#xb85c;&#xc815;&#xc758;&#xd569;&#xb2c8;&#xb2e4;. &#xc608;&#xb97c;&#xb4e4;&#xc5b
&#xc0ac;&#xc6a9;&#xc790;&#xc124;&#xc815;&#xd53d;&#xc140;&#xc8fc;&#xc704;&#xb85c; X&#xcd95;&#xc744;&#xb530;&#
3&#xd53d;&#xc140;&#xac70;&#xb9ac;&#xc548;&#xadf8;&#xb9ac;&#xace0; Y&#xcd95;&#xc744;&#xb530;&#xb77c;
2&#xd53d;&#xc140;&#xac70;&#xb9ac;&#xc548;&#xc758;&#xbaa8;&#xb4e0;&#xac12;&#xc744;&#xc6d0;&#xd558;&#xb294;
&#xacbd;&#xc6b0;&#xb9d0;&#xc785;&#xb2c8;&#xb2e4;. 2&#xcc28;&#xc6d0;&#bc30;&#xc5f4;&#xc758;&#xc911;&#xc2ec;&#
columnx &#bc0f; rowy &#b610;&#xb294;&#xae30;&#xd558;&#xd559;&#xc801;&#xd3ec;&#xc778;&#xd2b8;&#xb85c;
&#xc9c0;&#xc815;&#xb41c;&#xd53d;&#xc140;&#xc758;&#xac12;&#xc774;&#xb420;&#xac83;&#xc785;&#xb2c8;&#xb2e4;.
&#bc34;&#xb4dc;&#bc88;&#xd638;&#b294; 1&#bd80;&#xd130;&#xc2dc;&#xc791;&#xd558;&#ba70;,&#b530;&#b85c;
&#xc9c0;&#xc815;&#xd558;&#xc9c0;&#xc54a;&#xc744;&#xacbd;&#xc6b0; bandnum &#xc744; 1&#xb85c;&#xc00;&#xc815;&#
exclude_nodata_value &#xb97c;&#xac70;&#xc9d3;&#xc73c;&#xb85c;&#xc124;&#xc815;&#xd560;&#xacbd;&#xc6b0;,&#
nodata &#xd53d;&#xc140;&#xc744;&#xd3ec;&#xd568;&#xd55c;&#baa8;&#b4e0;&#xd53d;&#xc140;&#xc774;&#xad50;&#
&#xc00;&#xc815;&#xd558;&#xace0;&#xac12;&#xc744;&#bc18;&#xd658;&#xd569;&#b2c8;&#b2e4;. exclude_nodata_
&#xb97c;&#xc124;&#xc815;&#xd558;&#xc9c0;&#xc54a;&#xc740;&#xacbd;&#xc6b0;,&#b798;&#xc2a4;&#xd130;&#xc758;
&#ba54;&#xd0c0;&#b370;&#xc774;&#xd130;&#xc5d0;&#xc11c;&#xac12;&#xc744;&#xc77d;&#xc5b4;&#b4e4;&#xc785;&#

```

**Note**

```

&#bc18;&#xd658;&#b418;&#b294; 2&#xcc28;&#xc6d0;&#bc30;&#xc5f4;&#xc758;&#xc01;&#xcd95;&#xc758;
&#xad6c;&#xc131;&#xc694;&#xc18c;&#xac1c;&#xc218;&#b294; 2 * (distanceX|distanceY) +
1&#xc785;&#xb2c8;&#b2e4;. &#b530;&#b77c;&#xc11c; distanceX &#xc640; distanceY &#xc00;
&#xc01;&#xc01; 1&#xc778;&#xacbd;&#xc6b0;,&#bc18;&#xd658;&#b418;&#b294;&#bc30;&#xc5f4;&#xc740;
3x3&#xc774;&#b420;&#xac83;&#xc785;&#b2c8;&#b2e4;.

```

**Note**

```

ST_Min4ma, ST_Sum4ma, ST_Mean4ma &#xac19;&#xc740;&#xc5b4;&#b5a4;&#b798;&#xc2a4;&#xd130;
&#xacf5;&#xc04;&#xcc98;&#b9ac;&#b0b4;&#xc7a5;&#xd568;&#xc218;&#b3c4;&#xc774; 2&#xcc28;&#xc6d0;
&#bc30;&#xc5f4;&#xcd9c;&#b825;&#bb3c;&#xc744;&#xc785;&#b825;&#bc1b;&#xc744;&#xc218;
&#xc788;&#xc2b5;&#b2c8;&#b2e4;.

```

2.1.0 &#bc84;&#xc804;&#bd80;&#xd130;&#xc0ac;&#xc6a9;&#xd560;&#xc218;&#xc788;&#xc2b5;&#b2c8;&#b2e4;.

**&#xc608;&#xc2dc;**

```

-- 2x2 &#xd53d;&#xc140;&#xc774;&#xac12;&#xc744;&#xc00;&#xc9c0;&#xace0; ↔
&#xc788;&#xc2b5;&#b2c8;&#b2e4;.
SELECT
  ST_Neighborhood(rast, 2, 2, 1, 1)
FROM (
  SELECT
    ST_SetValues(
      ST_AddBand(
        ST_MakeEmptyRaster(5, 5, -2, 2, 1, -1, 0, 0, 0),
        '8BUI'::text, 1, 0
      ),
      1, 1, 1, ARRAY[
        [0, 1, 1, 1, 1],
        [1, 1, 1, 0, 1],
        [1, 0, 1, 1, 1],
        [1, 1, 1, 1, 0],
        [1, 1, 0, 1, 1]
      ]::double precision[],
      1

```

```

        ) AS rast
) AS foo

-----
st_neighborhood
-----
{{NULL,1,1},{1,1,NULL},{1,1,1}}

-- 2x3 &#xd53d;&#xc140;&#xc774; NODATA&#xc785;&#xb2c8;&#xb2e4;.
SELECT
  ST_Neighborhood(rast, 2, 3, 1, 1)
FROM (
  SELECT
    ST_SetValues(
      ST_AddBand(
        ST_MakeEmptyRaster(5, 5, -2, 2, 1, -1, 0, 0, 0),
        '8BUI'::text, 1, 0
      ),
      1, 1, 1, ARRAY[
        [0, 1, 1, 1, 1],
        [1, 1, 1, 0, 1],
        [1, 0, 1, 1, 1],
        [1, 1, 1, 1, 0],
        [1, 1, 0, 1, 1]
      ]::double precision[],
      1
    ) AS rast
) AS foo

-----
st_neighborhood
-----
{{1,1,1},{1,NULL,1},{1,1,1}}

```

```

-- 3x3 &#xd53d;&#xc140;&#xc774; &#xac12;&#xc744; &#xac00;&#xc9c0;&#xace0; ↔
&#xc788;&#xc2b5;&#xb2c8;&#xb2e4;.
-- exclude_nodata_value = FALSE
SELECT
  ST_Neighborhood(rast, 3, 3, 1, 1, false)
FROM (
  SELECT
    ST_SetValues(
      ST_AddBand(
        ST_MakeEmptyRaster(5, 5, -2, 2, 1, -1, 0, 0, 0),
        '8BUI'::text, 1, 0
      ),
      1, 1, 1, ARRAY[
        [0, 1, 1, 1, 1],
        [1, 1, 1, 0, 1],
        [1, 0, 1, 1, 1],
        [1, 1, 1, 1, 0],
        [1, 1, 0, 1, 1]
      ]::double precision[],
      1
    ) AS rast
) AS foo

-----
st_neighborhood
-----
{{1,0,1},{1,1,1},{0,1,1}}

```

**ST\_SetValue**

[ST\\_NearestValue](#), [ST\\_Min4ma](#), [ST\\_Max4ma](#), [ST\\_Sum4ma](#), [ST\\_Mean4ma](#), [ST\\_Range4ma](#), [ST\\_Distinct4ma](#), [ST\\_StdDev4ma](#)

## 12.6.12 ST\_SetValue

**ST\_SetValue** — `columnx, rowy`; `geometry geom, double precision newvalue`;  
`geometry geom, double precision newvalue`;  
`integer columnx, integer rowy, double precision newvalue`;  
`integer columnx, integer rowy, double precision newvalue`;

### Synopsis

raster **ST\_SetValue**(raster rast, integer bandnum, geometry geom, double precision newvalue);  
raster **ST\_SetValue**(raster rast, geometry geom, double precision newvalue);  
raster **ST\_SetValue**(raster rast, integer bandnum, integer columnx, integer rowy, double precision newvalue);  
raster **ST\_SetValue**(raster rast, integer columnx, integer rowy, double precision newvalue);

**ST\_SetValue**

Returns modified raster resulting from setting the specified pixels' values to new value for the designated band given the raster's row and column or a geometry. If no band is specified, then band 1 is assumed.

`2.1.0`; `ST_SetValue()`;  
`ST_SetValue()`;  
`ST_SetValues()`; `geomval[]`;

**ST\_SetValue**

```
-- SELECT
SELECT (foo.geomval).val, ST_AsText(ST_Union((foo.geomval).geom))
FROM (SELECT ST_DumpAsPolygons(
        ST_SetValue(rast,1,
                    ST_Point(3427927.75, 5793243.95),
                    50)
      ) As geomval
FROM dummy_rast
where rid = 2) As foo
WHERE (foo.geomval).val < 250
GROUP BY (foo.geomval).val;
```

| val | st_astext  |
|-----|--|
| 50  | POLYGON((3427927.75 5793244,3427927.75 5793243.95,3427927.8 579324 ... |
| 249 | POLYGON((3427927.95 5793243.95,3427927.95 5793243.85,3427928 57932 ... |

```
-- UPDATE
UPDATE dummy_rast SET rast = ST_SetValue(rast,1, ST_Point(3427927.75, 5793243.95)
,100)
WHERE rid = 2 ;
```

**ST\_SetValues**

**ST\_Value, ST\_DumpAsPolygons**

### 12.6.13 ST\_SetValues

**ST\_SetValues** — Set the values of a raster. The raster is updated in place. The new values are specified by the `newvalueset` parameter. The `noiset` parameter controls whether the existing values are preserved. The `keepnodata` parameter controls whether the existing no-data values are preserved.

#### Synopsis

```
raster ST_SetValues(raster rast, integer nband, integer columnx, integer rowy, double precision[] newvalueset, boolean[] noiset=NULL, boolean keepnodata=FALSE);
```

```
raster ST_SetValues(raster rast, integer nband, integer columnx, integer rowy, double precision[] newvalueset, double precision noisetvalue, boolean keepnodata=FALSE);
```

```
raster ST_SetValues(raster rast, integer nband, integer columnx, integer rowy, integer width, integer height, double precision newvalue, boolean keepnodata=FALSE);
```

```
raster ST_SetValues(raster rast, integer columnx, integer rowy, integer width, integer height, double precision newvalue, boolean keepnodata=FALSE);
```

```
raster ST_SetValues(raster rast, integer nband, geomval[] geomvalset, boolean keepnodata=FALSE);
```

**ST\_SetValues**

```
ST_SetValues(raster rast, integer nband, integer columnx, integer rowy, double precision[] newvalueset, boolean[] noiset=NULL, boolean keepnodata=FALSE);
```

```
ST_SetValues(raster rast, integer nband, integer columnx, integer rowy, double precision[] newvalueset, double precision noisetvalue, boolean keepnodata=FALSE);
```

```
ST_SetValues(raster rast, integer nband, integer columnx, integer rowy, integer width, integer height, double precision newvalue, boolean keepnodata=FALSE);
```

```
ST_SetValues(raster rast, integer columnx, integer rowy, integer width, integer height, double precision newvalue, boolean keepnodata=FALSE);
```

```
ST_SetValues(raster rast, integer nband, geomval[] geomvalset, boolean keepnodata=FALSE);
```

```
ST_SetValues(raster rast, integer nband, integer columnx, integer rowy, double precision[] newvalueset, boolean[] noiset=NULL, boolean keepnodata=FALSE);
```

```
ST_SetValues(raster rast, integer nband, integer columnx, integer rowy, double precision[] newvalueset, double precision noisetvalue, boolean keepnodata=FALSE);
```

```
ST_SetValues(raster rast, integer nband, integer columnx, integer rowy, integer width, integer height, double precision newvalue, boolean keepnodata=FALSE);
```

```
ST_SetValues(raster rast, integer columnx, integer rowy, integer width, integer height, double precision newvalue, boolean keepnodata=FALSE);
```

```

+ - + - + - +      + - + - + - +
| 1 | 1 | 1 |      | 1 | 1 | 1 |
+ - + - + - +      + - + - + - +
| 1 | 1 | 1 |      =
> | 1 | 9 | 9 |
+ - + - + - +      + - + - + - +
| 1 | 1 | 1 |      | 1 | 9 | 9 |
+ - + - + - +      + - + - + - +
*/

```

2.1.0

### ST\_SetValues: ST\_PixelAsPolygons

```

/*
ST_SetValues() <math>[x, y, val]</math> <math>[x, y, val]</math> <math>[x, y, val]</math> <math>[x, y, val]</math> <math>[x, y, val]</math>
*/

```

```

+ - + - + - +      + - + - + - +
| 1 | 1 | 1 |      | 1 | 1 | 1 |
+ - + - + - +      + - + - + - +
| 1 | 1 | 1 |      =
> | 1 | 9 | 9 |
+ - + - + - +      + - + - + - +
| 1 | 1 | 1 |      | 1 | 9 | 9 |
+ - + - + - +      + - + - + - +
*/
SELECT
    (poly).x,
    (poly).y,
    (poly).val
FROM (
SELECT
    ST_PixelAsPolygons(
        ST_SetValues(
            ST_AddBand(
                ST_MakeEmptyRaster(3, 3, 0, 0, 1, -1, 0, 0, 0),
                1, '8BUI', 1, 0
            ),
            1, 2, 2, ARRAY[[9, 9], [9, 9]]::double precision[]
        )
    ) AS poly
) foo
ORDER BY 1, 2;

```

```

x | y | val
---+---+---
1 | 1 | 1
1 | 2 | 1
1 | 3 | 1
2 | 1 | 1
2 | 2 | 9
2 | 3 | 9
3 | 1 | 1
3 | 2 | 9
3 | 3 | 9

```

```

/*
ST_SetValues() <math>[x, y, val]</math> <math>[x, y, val]</math> <math>[x, y, val]</math> <math>[x, y, val]</math> <math>[x, y, val]</math>
*/

```

```

+ - + - + - +      + - + - + - +
| 1 | 1 | 1 |      | 9 | 9 | 9 |
+ - + - + - +      + - + - + - +
| 1 | 1 | 1 |      =
> | 9 | 9 | 9 |

```

```

+ - + - + - +      + - + - + - +
| 1 | 1 | 1 |      | 9 | 9 | 9 |
+ - + - + - +      + - + - + - +
*/
SELECT
    (poly).x,
    (poly).y,
    (poly).val
FROM (
SELECT
    ST_PixelAsPolygons(
        ST_SetValues(
            ST_AddBand(
                ST_MakeEmptyRaster(3, 3, 0, 0, 1, -1, 0, 0, 0),
                1, '8BUI', 1, 0
            ),
            1, 1, 1, ARRAY[[9, 9, 9], [9, NULL, 9], [9, 9, 9]]::double ←
                precision[][]
        )
    ) AS poly
) foo
ORDER BY 1, 2;

```

```

x | y | val
---+---+---
1 | 1 | 9
1 | 2 | 9
1 | 3 | 9
2 | 1 | 9
2 | 2 |
2 | 3 | 9
3 | 1 | 9
3 | 2 | 9
3 | 3 | 9

```

```

/*
ST_SetValues() &#xd568;&#xc218;&#xac00; &#xb2e4;&#xc74c; &#xc791;&#xc5c5;&#xc744; ←
&#xd569;&#xb2c8;&#xb2e4; .

+ - + - + - +      + - + - + - +
| 1 | 1 | 1 |      | 9 | 9 | 9 |
+ - + - + - +      + - + - + - +
| 1 | 1 | 1 |      =
> | 1 | | 9 |
+ - + - + - +      + - + - + - +
| 1 | 1 | 1 |      | 9 | 9 | 9 |
+ - + - + - +      + - + - + - +
*/
SELECT
    (poly).x,
    (poly).y,
    (poly).val
FROM (
SELECT
    ST_PixelAsPolygons(
        ST_SetValues(
            ST_AddBand(
                ST_MakeEmptyRaster(3, 3, 0, 0, 1, -1, 0, 0, 0),
                1, '8BUI', 1, 0
            ),
            1, 1, 1,
            ARRAY[[9, 9, 9], [9, NULL, 9], [9, 9, 9]]::double precision ←

```

```

        [],
        ARRAY[[false], [true]]::boolean[][])
    ) AS poly
) foo
ORDER BY 1, 2;

```

| x | y | val |
|---|---|-----|
| 1 | 1 | 9   |
| 1 | 2 | 1   |
| 1 | 3 | 9   |
| 2 | 1 | 9   |
| 2 | 2 |     |
| 2 | 3 | 9   |
| 3 | 1 | 9   |
| 3 | 2 | 9   |
| 3 | 3 | 9   |

```

/*
ST_SetValues() &#xd568;&#xc218;&#xac00; &#xb2e4;&#xc74c; &#xc791;&#xc5c5;&#xc744; <-
&#xd569;&#xb2c8;&#xb2e4;.

+ - + - + - +          + - + - + - +
| 1 | 1 | 1 |          | 9 | 9 | 9 |
+ - + - + - +          + - + - + - +
| 1 | 1 | 1 |          =
> | 1 | | | 9 |
+ - + - + - +          + - + - + - +
| 1 | 1 | 1 |          | 9 | 9 | 9 |
+ - + - + - +          + - + - + - +
*/
SELECT
    (poly).x,
    (poly).y,
    (poly).val
FROM (
SELECT
    ST_PixelAsPolygons(
        ST_SetValues(
            ST_AddBand(
                ST_MakeEmptyRaster(3, 3, 0, 0, 1, -1, 0, 0, 0),
                1, '8BUI', 1, 0
            ),
            1, 1, 1,
            ARRAY[[9, 9, 9], [9, NULL, 9], [9, 9, 9]]::double precision <-
                [],
            ARRAY[[false], [true]]::boolean[][])
        ) AS poly
) foo
ORDER BY 1, 2;

```

| x | y | val |
|---|---|-----|
| 1 | 1 | 9   |
| 1 | 2 | 1   |
| 1 | 3 | 9   |
| 2 | 1 | 9   |
| 2 | 2 |     |
| 2 | 3 | 9   |
| 3 | 1 | 9   |

```
3 | 2 | 9
3 | 3 | 9
```

**ST\_SetValues() with a 3x3 grid and a 3x3 grid of values**

```
/*
ST_SetValues() with a 3x3 grid and a 3x3 grid of values
  &#xd568;&#xc218;&#xac00; &#xb2e4;&#xc74c; &#xc791;&#xc5c5;&#xc744; &#x2190
  &#xd569;&#xb2c8;&#xb2e4;.

+ - + - + - +          + - + - + - +
| 1 | 1 | 1 |          | 1 | 1 | 1 |
+ - + - + - +          + - + - + - +
| 1 | 1 | 1 |      =
> | 1 | 9 | 9 |
+ - + - + - +          + - + - + - +
| 1 | 1 | 1 |          | 1 | 9 | 9 |
+ - + - + - +          + - + - + - +
*/
SELECT
    (poly).x,
    (poly).y,
    (poly).val
FROM (
SELECT
    ST_PixelAsPolygons(
        ST_SetValues(
            ST_AddBand(
                ST_MakeEmptyRaster(3, 3, 0, 0, 1, -1, 0, 0, 0),
                1, '8BUI', 1, 0
            ),
            1, 1, 1, ARRAY[[-1, -1, -1], [-1, 9, 9], [-1, 9, 9]]::double &#x2190
                precision[[]], -1
        )
    ) AS poly
) foo
ORDER BY 1, 2;

x | y | val
---+---+-----
1 | 1 | 1
1 | 2 | 1
1 | 3 | 1
2 | 1 | 1
2 | 2 | 9
2 | 3 | 9
3 | 1 | 1
3 | 2 | 9
3 | 3 | 9
```

```
/*
&#xc774; &#xc608;&#xc2dc;&#xb294; &#xc774;&#xc804; &#xc608;&#xc2dc;&#xc640; &#x2190
&#xbe44;&#xc2b7;&#xd558;&#xc9c0;&#xb9cc;, nosetvalue = -1 &#xb300;&#xc2e0; nosetvalue = &#x2190
NULL &#xc744; &#xc501;&#xb2c8;&#xb2e4;.

ST_SetValues() with a 3x3 grid and a 3x3 grid of values
  &#xd568;&#xc218;&#xac00; &#xb2e4;&#xc74c; &#xc791;&#xc5c5;&#xc744; &#x2190
  &#xd569;&#xb2c8;&#xb2e4;.

+ - + - + - +          + - + - + - +
| 1 | 1 | 1 |          | 1 | 1 | 1 |
+ - + - + - +          + - + - + - +
```

```

| 1 | 1 | 1 |   =
>   | 1 | 9 | 9 |
+ - + - + - +   + - + - + - +
| 1 | 1 | 1 |   | 1 | 9 | 9 |
+ - + - + - +   + - + - + - +
*/
SELECT
    (poly).x,
    (poly).y,
    (poly).val
FROM (
SELECT
    ST_PixelAsPolygons(
        ST_SetValues(
            ST_AddBand(
                ST_MakeEmptyRaster(3, 3, 0, 0, 1, -1, 0, 0, 0),
                1, '8BUI', 1, 0
            ),
            1, 1, 1, ARRAY[[NULL, NULL, NULL], [NULL, 9, 9], [NULL, 9, 9]]::↵
                double precision[], NULL::double precision
        )
    ) AS poly
) foo
ORDER BY 1, 2;

 x | y | val
---+---+-----
 1 | 1 |   1
 1 | 2 |   1
 1 | 3 |   1
 2 | 1 |   1
 2 | 2 |   9
 2 | 3 |   9
 3 | 1 |   1
 3 | 2 |   9
 3 | 3 |   9

```

### ST\_SetValues(geom, val, x, y, z)

```

/*
ST_SetValues(geom, val, x, y, z)
    Returns a copy of the input geometry with the specified value
    assigned to the pixel at the specified x, y, and z coordinates.
    The input geometry must be a 2D or 3D raster.

+ - + - + - +   + - + - + - +
| 1 | 1 | 1 |   | 1 | 1 | 1 |
+ - + - + - +   + - + - + - +
| 1 | 1 | 1 |   =
>   | 1 | 9 | 9 |
+ - + - + - +   + - + - + - +
| 1 | 1 | 1 |   | 1 | 9 | 9 |
+ - + - + - +   + - + - + - +
*/
SELECT
    (poly).x,
    (poly).y,
    (poly).val
FROM (
SELECT
    ST_PixelAsPolygons(
        ST_SetValues(

```

```

        ST_AddBand(
            ST_MakeEmptyRaster(3, 3, 0, 0, 1, -1, 0, 0, 0),
            1, '8BUI', 1, 0
        ),
        1, 2, 2, 2, 2, 9
    )
) AS poly
) foo
ORDER BY 1, 2;

```

| x | y | val |
|---|---|-----|
| 1 | 1 | 1   |
| 1 | 2 | 1   |
| 1 | 3 | 1   |
| 2 | 1 | 1   |
| 2 | 2 | 9   |
| 2 | 3 | 9   |
| 3 | 1 | 1   |
| 3 | 2 | 9   |
| 3 | 3 | 9   |

```

/*
ST_SetValues() &#xd568;&#xc218;&#xac00; &#xb2e4;&#xc74c; &#xc791;&#xc5c5;&#xc744; ←
&#xd569;&#xb2c8;&#xb2e4;.

```

```

+ - + - + - +          + - + - + - +
| 1 | 1 | 1 |          | 1 | 1 | 1 |
+ - + - + - +          + - + - + - +
| 1 | 1 | 1 |          =
> | 1 | 9 | 9 |
+ - + - + - +          + - + - + - +
| 1 | 1 | 1 |          | 1 | 9 | 9 |
+ - + - + - +          + - + - + - +
*/

```

```

SELECT
    (poly).x,
    (poly).y,
    (poly).val
FROM (
SELECT
    ST_PixelAsPolygons(
        ST_SetValues(
            ST_AddBand(
                ST_MakeEmptyRaster(3, 3, 0, 0, 1, -1, 0, 0, 0),
                1, '8BUI', 1, 0
            ),
            1, 2, 2, 2, 2, 9
        )
    ) AS poly
) foo
ORDER BY 1, 2;

```

| x | y | val |
|---|---|-----|
| 1 | 1 | 1   |
| 1 | 2 | 1   |
| 1 | 3 | 1   |
| 2 | 1 | 1   |
| 2 | 2 | 9   |
| 2 | 3 | 9   |
| 3 | 1 | 1   |

```
3 | 2 | 9
3 | 3 | 9
```

### ST\_DumpValues

```
WITH foo AS (
  SELECT 1 AS rid, ST_AddBand(ST_MakeEmptyRaster(5, 5, 0, 0, 1, -1, 0, 0, 0), 1, '8 ←
    BUI', 0, 0) AS rast
), bar AS (
  SELECT 1 AS gid, 'SRID=0;POINT(2.5 -2.5)::geometry geom UNION ALL
  SELECT 2 AS gid, 'SRID=0;POLYGON((1 -1, 4 -1, 4 -4, 1 -4, 1 -1))::geometry geom ←
    UNION ALL
  SELECT 3 AS gid, 'SRID=0;POLYGON((0 0, 5 0, 5 -1, 1 -1, 1 -4, 0 -4, 0 0))':: ←
    geometry geom UNION ALL
  SELECT 4 AS gid, 'SRID=0;MULTIPOINT(0 0, 4 4, 4 -4)::geometry
)
SELECT
  rid, gid, ST_DumpValues(ST_SetValue(rast, 1, geom, gid))
FROM foo t1
CROSS JOIN bar t2
ORDER BY rid, gid;
```

| rid | gid | st_dumpvalues   |
|-----|-----|---|
| 1   | 1   | (1, "{ {NULL, NULL, NULL, NULL, NULL}, {NULL, NULL, NULL, NULL, NULL}, {NULL, NULL, 1, NULL, ←<br>NULL}, {NULL, NULL, NULL, NULL, NULL}, {NULL, NULL, NULL, NULL, NULL} }") |
| 1   | 2   | (1, "{ {NULL, NULL, NULL, NULL, NULL}, {NULL, 2, 2, 2, NULL}, {NULL, 2, 2, 2, NULL}, {NULL ←<br>, 2, 2, 2, NULL}, {NULL, NULL, NULL, NULL, NULL} }")                        |
| 1   | 3   | (1, "{ {3, 3, 3, 3, 3}, {3, NULL, NULL, NULL, NULL}, {3, NULL, NULL, NULL, NULL}, {3, NULL, NULL, ←<br>NULL, NULL}, {NULL, NULL, NULL, NULL, NULL} }")                      |
| 1   | 4   | (1, "{ {4, NULL, NULL, NULL, NULL}, {NULL, NULL, NULL, NULL, NULL}, {NULL, NULL, NULL, NULL, ←<br>NULL}, {NULL, NULL, NULL, NULL, NULL}, {NULL, NULL, NULL, NULL, 4} }")    |

(4 rows)

ST\_DumpValues

```
WITH foo AS (
  SELECT 1 AS rid, ST_AddBand(ST_MakeEmptyRaster(5, 5, 0, 0, 1, -1, 0, 0, 0), 1, '8 ←
    BUI', 0, 0) AS rast
), bar AS (
  SELECT 1 AS gid, 'SRID=0;POINT(2.5 -2.5)::geometry geom UNION ALL
  SELECT 2 AS gid, 'SRID=0;POLYGON((1 -1, 4 -1, 4 -4, 1 -4, 1 -1))::geometry geom ←
    UNION ALL
  SELECT 3 AS gid, 'SRID=0;POLYGON((0 0, 5 0, 5 -1, 1 -1, 1 -4, 0 -4, 0 0))':: ←
    geometry geom UNION ALL
  SELECT 4 AS gid, 'SRID=0;MULTIPOINT(0 0, 4 4, 4 -4)::geometry
)
SELECT
  t1.rid, t2.gid, t3.gid, ST_DumpValues(ST_SetValues(rast, 1, ARRAY[ROW(t2.geom, t2. ←
    gid), ROW(t3.geom, t3.gid)]::geomval[]))
FROM foo t1
CROSS JOIN bar t2
CROSS JOIN bar t3
WHERE t2.gid = 1
      AND t3.gid = 2
ORDER BY t1.rid, t2.gid, t3.gid;
```

```

rid | gid | gid | st_dumpvalues
-----+-----+-----+-----
1 | 1 | 2 | (1, "{NULL,NULL,NULL,NULL,NULL},{NULL,2,2,2,NULL},{NULL,2,2,2,NULL},{
NULL,2,2,2,NULL},{NULL,NULL,NULL,NULL,NULL}")
(1 row)

```

&#xc774;&#xc804; &#xc608;&#xc2dc;&#xc640; &#xbc18;&#xb300;&#xb418;&#xb294; &#xc0c1;&#xd669;&#xc744; &#xbcf4;&#xc608;&#xc2dc;&#xc785;&#xb2c8;&#xb2e4;

```

WITH foo AS (
    SELECT 1 AS rid, ST_AddBand(ST_MakeEmptyRaster(5, 5, 0, 0, 1, -1, 0, 0, 0), 1, '8 ←
    BUI', 0, 0) AS rast
), bar AS (
    SELECT 1 AS gid, 'SRID=0;POINT(2.5 -2.5)::geometry geom UNION ALL
    SELECT 2 AS gid, 'SRID=0;POLYGON((1 -1, 4 -1, 4 -4, 1 -4, 1 -1))::geometry geom ←
    UNION ALL
    SELECT 3 AS gid, 'SRID=0;POLYGON((0 0, 5 0, 5 -1, 1 -1, 1 -4, 0 -4, 0 0)):: ←
    geometry geom UNION ALL
    SELECT 4 AS gid, 'SRID=0;MULTIPOINT(0 0, 4 4, 4 -4)::geometry
)
SELECT
    t1.rid, t2.gid, t3.gid, ST_DumpValues(ST_SetValues(rast, 1, ARRAY[ROW(t2.geom, t2. ←
    gid), ROW(t3.geom, t3.gid)]::geomval[]))
FROM foo t1
CROSS JOIN bar t2
CROSS JOIN bar t3
WHERE t2.gid = 2
    AND t3.gid = 1
ORDER BY t1.rid, t2.gid, t3.gid;

```

```

rid | gid | gid | st_dumpvalues
-----+-----+-----+-----
1 | 2 | 1 | (1, "{NULL,NULL,NULL,NULL,NULL},{NULL,2,2,2,NULL},{NULL,2,1,2,NULL},{ ←
NULL,2,2,2,NULL},{NULL,NULL,NULL,NULL,NULL}")
(1 row)

```

&#xcc38;&#xace0;

[ST\\_Value](#), [ST\\_SetValue](#), [ST\\_PixelAsPolygons](#)

## 12.6.14 ST\_DumpValues

`ST_DumpValues` — &#xc9c0;&#xc815;&#xb41c; &#xbc34;&#xb4dc;&#xc758; &#xac12;&#xb4e4;&#xc744; 2&#xcc28;&#xc6d0; &#xbc30;&#xc5f4;&#xb85c; &#xbc18;&#xd658;&#xd569;&#xb2c8;&#xb2e4;

### Synopsis

```

setof record ST_DumpValues( raster rast , integer[] nband=NULL , boolean exclude_nodata_value=true );
double precision[][] ST_DumpValues( raster rast , integer nband , boolean exclude_nodata_value=true );

```

&#xc124;&#xba85;

&#xc9c0;&#xc815;&#xb41c; &#xbc34;&#xb4dc;&#xc758; &#xac12;&#xb4e4;&#xc744; 2&#xcc28;&#xc6d0; &#xbc30;&#xc5f4;&#xb85c; &#xbc18;&#xd658;&#xd569;&#xb2c8;&#xb2e4;(&#xccab; &#xbc88;&#xc9f8; &#xc778;&#xb371;&#xc2a4;&#xac00; &#xd589;

band, nband, NULL, and ST\_AddBand. The ST\_AddBand function is used to add a new band to a raster. The ST\_SetValue function is used to set the value of a pixel in a raster. The ST\_SetValues function is used to set the values of a range of pixels in a raster.

2.1.0. The ST\_AddBand function is used to add a new band to a raster.

### ST\_AddBand

```
WITH foo AS (
  SELECT ST_AddBand(ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(3, 3, 0, 0, 1, -1, 0, 0, 0), ←
    1, '8BUI'::text, 1, 0), 2, '32BF'::text, 3, -9999), 3, '16BSI', 0, 0) AS rast
)
SELECT
  (ST_DumpValues(rast)).*
FROM foo;
```

| nband | valarray   |
|-------|--|
| 1     | {{1,1,1},{1,1,1},{1,1,1}}                            |
| 2     | {{3,3,3},{3,3,3},{3,3,3}}                            |
| 3     | {{NULL,NULL,NULL},{NULL,NULL,NULL},{NULL,NULL,NULL}} |

(3 rows)

```
WITH foo AS (
  SELECT ST_SetValue(ST_AddBand(ST_MakeEmptyRaster(3, 3, 0, 0, 1, -1, 0, 0, 0), 1, '8 ←
    BUI', 1, 0), 1, 2, 5) AS rast
)
SELECT
  (ST_DumpValues(rast, 1))[2][1]
FROM foo;
```

| st_dumpvalues |
|---------------|
| 5             |

(1 row)

```
WITH foo AS (
  SELECT ST_SetValue(ST_AddBand(ST_MakeEmptyRaster(3, 3, 0, 0, 1, -1, 0, 0, 0), 1, '8 ←
    BUI', 1, 0), 1, 2, 5) AS rast
)
SELECT
  (ST_DumpValues(rast, 1))[2][1]
FROM foo;
```

| st_dumpvalues |
|---------------|
| 5             |

(1 row)

### ST\_SetValue

[ST\\_Value](#), [ST\\_SetValue](#), [ST\\_SetValues](#)

## 12.6.15 ST\_PixelOfValue

ST\_PixelOfValue — Returns the value of a pixel in a raster. The ST\_PixelOfValue function is used to retrieve the value of a pixel in a raster. The ST\_PixelOfValue function is used to retrieve the value of a pixel in a raster.

## Synopsis

```
setof record ST_PixelOfValue( raster rast , integer nband , double precision[] search , boolean exclude_nodata_value=true );
setof record ST_PixelOfValue( raster rast , double precision[] search , boolean exclude_nodata_value=true );
setof record ST_PixelOfValue( raster rast , integer nband , double precision search , boolean exclude_nodata_value=true );
setof record ST_PixelOfValue( raster rast , double precision search , boolean exclude_nodata_value=true );
```

## Parameters

`rast`: raster to be processed.  
`nband`: number of bands to be processed.  
`search`: array of double precision values to search for.  
`exclude_nodata_value`: boolean flag to exclude nodata values from the search results.

2.1.0: `search` is a double precision value.

## Examples

```
SELECT
  (pixels).*
FROM (
  SELECT
    ST_PixelOfValue (
      ST_SetValue (
        ST_SetValue (
          ST_SetValue (
            ST_SetValue (
              ST_SetValue (
                ST_AddBand (
                  ST_MakeEmptyRaster ←
                    (5, 5, -2, 2, 1, ←
                    -1, 0, 0, 0),
                  '8BUI'::text, 1, 0
                ),
                1, 1, 0
              ),
              2, 3, 0
            ),
            3, 5, 0
          ),
          4, 2, 0
        ),
        5, 4, 255
      ),
      1, ARRAY[1, 255]) AS pixels
) AS foo
```

| val | x | y |
|-----|---|---|
| 1   | 1 | 2 |
| 1   | 1 | 3 |
| 1   | 1 | 4 |
| 1   | 1 | 5 |
| 1   | 2 | 1 |
| 1   | 2 | 2 |
| 1   | 2 | 4 |
| 1   | 2 | 5 |
| 1   | 3 | 1 |
| 1   | 3 | 2 |

```

1 | 3 | 3
1 | 3 | 4
1 | 4 | 1
1 | 4 | 3
1 | 4 | 4
1 | 4 | 5
1 | 5 | 1
1 | 5 | 2
1 | 5 | 3
255 | 5 | 4
1 | 5 | 5
    
```

## 12.7 ST\_SetGeoReference

### 12.7.1 ST\_SetGeoReference

ST\_SetGeoReference — raster, text georefcoords, text format=GDAL);

#### Synopsis

raster ST\_SetGeoReference(raster rast, text georefcoords, text format=GDAL);  
 raster ST\_SetGeoReference(raster rast, double precision upperleftx, double precision upperlefty, double precision scalex, double precision scaley, double precision skewx, double precision skewy);

#### Parameters

GDAL: `scalex skewx scaley upperleftx upperlefty`

ESRI: `scalex skewy skewx scaley upperleftx + scalex*0.5 upperlefty + scaley*0.5`

#### Note



DB `&#x678;&#xbd80;` `&#xbc34;&#xb4dc;&#xb97c;`  
`&#xac00;&#xc9c0;&#xace0;` `&#xc788;&#xc744;` `&#xacbd;&#xc6b0;` `&#xc9c0;&#xb9ac;&#xcc38;&#xc870;`  
`&#xd30c;&#xb77c;&#xbbf8;&#xd130;&#xb97c;` `&#xbcc0;&#xacbd;&#xd558;&#xba74;` `&#xbc34;&#xb4dc;&#xc758;`  
`&#xc678;&#xbd80;` `&#xc800;&#xc7a5;` `&#xb370;&#xc774;&#xd130;&#xc5d0;` `&#xc811;&#xadfc;&#xd558;&#xb294;`  
`&#xb370;` `&#xc624;&#xb958;&#xac00;` `&#xbc1c;&#xc0dd;&#xd560;` `&#xc218;&#xb3c4;`  
`&#xc788;&#xc2b5;&#xb2c8;&#xb2e4;`

2.1.0; ST\_SetGeoReference(raster, double precision, ...) ;

**;**

```
WITH foo AS (
    SELECT ST_MakeEmptyRaster(5, 5, 0, 0, 1, -1, 0, 0, 0) AS rast
)
SELECT
    0 AS rid, (ST_Metadata(rast)).*
FROM foo
UNION ALL
SELECT
    1, (ST_Metadata(ST_SetGeoReference(rast, '10 0 0 -10 0.1 0.1', 'GDAL'))).*
FROM foo
UNION ALL
SELECT
    2, (ST_Metadata(ST_SetGeoReference(rast, '10 0 0 -10 5.1 -4.9', 'ESRI'))).*
FROM foo
UNION ALL
SELECT
    3, (ST_Metadata(ST_SetGeoReference(rast, 1, 1, 10, -10, 0.001, 0.001))).*
FROM foo
```

| rid | upperleftx<br>skewy | srid | numbands | upperlefty          | width | height | scalex | scaley | skewx | skewy |
|-----|---------------------|------|----------|---------------------|-------|--------|--------|--------|-------|-------|
| 0   | 0                   | 0    | 0        | 0                   | 5     | 5      | 1      | -1     | 0     | 0     |
| 1   | 0.1                 | 0    | 0        | 0.1                 | 5     | 5      | 10     | -10    | 0     | 0     |
| 2   | 0.09999999999999996 | 0    | 0        | 0.09999999999999996 | 5     | 5      | 10     | -10    | 0     | 0     |
| 3   | 0.001               | 0    | 0        | 1                   | 5     | 5      | 10     | -10    | 0.001 | 0     |

**;**

[ST\\_GeoReference](#), [ST\\_ScaleX](#), [ST\\_ScaleY](#), [ST\\_UpperLeftX](#), [ST\\_UpperLeftY](#)

**12.7.2 ST\_SetRotation**

ST\_SetRotation — ;

**Synopsis**

float8 ST\_SetRotation(raster rast, float8 rotation);

**;**

;



```
WKT Raster <math>2.0.0</math> WKTRaster <math>84.804.5d0.11c.294</math>
ST_SetPixelSize <math>77c.294.85.6d.774.5c8.2b5.2c8.2e4.2.0.84.804.604.7ac.85.6d.73c.85c.8410.2b5.2c8.2e4.</math>
```

### <math>608.2dc;</math>

```
UPDATE dummy_rast
    SET rast = ST_SetScale(rast, 1.5)
WHERE rid = 2;

SELECT ST_ScaleX(rast) As pixx, ST_ScaleY(rast) As pixy, Box3D(rast) As newbox
FROM dummy_rast
WHERE rid = 2;
```

```
pixx | pixy | newbox
-----+-----+-----
1.5 | 1.5 | BOX(3427927.75 5793244 0, 3427935.25 5793251.5 0)
```

```
UPDATE dummy_rast
    SET rast = ST_SetScale(rast, 1.5, 0.55)
WHERE rid = 2;

SELECT ST_ScaleX(rast) As pixx, ST_ScaleY(rast) As pixy, Box3D(rast) As newbox
FROM dummy_rast
WHERE rid = 2;
```

```
pixx | pixy | newbox
-----+-----+-----
1.5 | 0.55 | BOX(3427927.75 5793244 0, 3427935.25 5793247 0)
```

### <math>ccc38.xae0;</math>

[ST\\_ScaleX](#), [ST\\_ScaleY](#), [Box3D](#)

## 12.7.4 ST\_SetSkew

`ST_SetSkew` —  $<math>9c0.9ac.ccc38.c870</math>; X  $<math>0f</math>; Y  $<math>30.6b8.30.</math>(skew)  $<math>b10.294.68c.804.01.30c.77c.85.97c.124.815.569.2c8.2e4.12.558.098.9cc.785.825.560.84bd.6b0.</math>, X  $<math>640</math>; Y  $<math>97c.3d9.77c.55c.12.73c.85.</math>$$$$$$

### Synopsis

```
raster ST_SetSkew(raster rast, float8 skewxy);
raster ST_SetSkew(raster rast, float8 skewx, float8 skewy);
```

### <math>124.85;</math>

$<math>9c0.9ac.ccc38.c870</math>; X  $<math>0f</math>; Y  $<math>30.6b8.30.</math>(skew)  $<math>b10.294.68c.804.01.30c.77c.85.97c.124.815.569.2c8.2e4.12.558.098.9cc.785.825.560.84bd.6b0.</math>, X  $<math>640</math>; Y  $<math>97c.3d9.77c.55c.12.73c.85.</math>$  [790.138.55c.840b4.6a9.740.6d4.84dc.30c.77c.744.84c38.c870.558.2ed.2dc.624.</math>](#)$$$$$

**Example 1**

```
-- Example 1
UPDATE dummy_rast SET rast = ST_SetSkew(rast,1,2) WHERE rid = 1;
SELECT rid, ST_SkewX(rast) As skewx, ST_SkewY(rast) As skewy,
       ST_GeoReference(rast) as georef
FROM dummy_rast WHERE rid = 1;
```

| rid | skewx | skewy | georef   |
|-----|-------|-------|--|
| 1   | 1     | 2     | 2.0000000000<br>: 2.0000000000<br>: 1.0000000000<br>: 3.0000000000<br>: 0.5000000000<br>: 0.5000000000 |

```
-- Example 2: ST_SetSkew(rast,0)
UPDATE dummy_rast SET rast = ST_SetSkew(rast,0) WHERE rid = 1;
SELECT rid, ST_SkewX(rast) As skewx, ST_SkewY(rast) As skewy,
       ST_GeoReference(rast) as georef
FROM dummy_rast WHERE rid = 1;
```

| rid | skewx | skewy | georef   |
|-----|-------|-------|--|
| 1   | 0     | 0     | 2.0000000000<br>: 0.0000000000<br>: 0.0000000000<br>: 3.0000000000<br>: 0.5000000000<br>: 0.5000000000 |

**Example 2**

[ST\\_GeoReference](#), [ST\\_SetGeoReference](#), [ST\\_SkewX](#), [ST\\_SkewY](#)

**12.7.5 ST\_SetSRID**

**ST\_SetSRID** — Set the SRID of a raster. `ST_SetSRID(raster rast, integer srid);`

**Synopsis**

raster **ST\_SetSRID**(raster rast, integer srid);

**Example 1**

```
-- Example 1
UPDATE dummy_rast SET rast = ST_SetSRID(rast, 3146) WHERE rid = 1;
SELECT rid, ST_SkewX(rast) As skewx, ST_SkewY(rast) As skewy,
       ST_GeoReference(rast) as georef
FROM dummy_rast WHERE rid = 1;
```

**Note**

Note: This section contains a list of functions and their descriptions. The text is partially obscured by a large, faint watermark or background image that appears to be a map or a technical diagram. The visible text includes:

**ST\_SetUpperLeft**

Section 4.5, [ST\\_SRID](#)

## 12.7.6 ST\_SetUpperLeft

**ST\_SetUpperLeft** — Sets the value of the upper left corner of the pixel of the raster to projected X and Y coordinates.

### Synopsis

raster **ST\_SetUpperLeft**(raster rast, double precision x, double precision y);

**ST\_SetUpperLeft**

Set the value of the upper left corner of raster to the projected X and Y coordinates

**ST\_SetUpperLeft**

```
SELECT ST_SetUpperLeft (rast, -71.01, 42.37)
FROM dummy_rast
WHERE rid = 2;
```

**ST\_SetUpperLeft**

[ST\\_SetUpperLeftX](#), [ST\\_SetUpperLeftY](#)

## 12.7.7 ST\_Resample

**ST\_Resample** — Resamples a raster to a new size and resolution. The new raster is created by interpolating the values of the original raster. The new raster is created by interpolating the values of the original raster. The new raster is created by interpolating the values of the original raster.

### Synopsis

raster **ST\_Resample**(raster rast, integer width, integer height, double precision gridx=NULL, double precision gridy=NULL, double precision skewx=0, double precision skewy=0, text algorithm=NearestNeighbour, double precision maxerr=0.125);  
 raster **ST\_Resample**(raster rast, double precision scalex=0, double precision scaley=0, double precision gridx=NULL, double precision gridy=NULL, double precision skewx=0, double precision skewy=0, text algorithm=NearestNeighbor, double precision maxerr=0.125);  
 raster **ST\_Resample**(raster rast, raster ref, text algorithm=NearestNeighbour, double precision maxerr=0.125, boolean usescale=true);  
 raster **ST\_Resample**(raster rast, raster ref, boolean usescale, text algorithm=NearestNeighbour, double precision maxerr=0.125);

**ST\_Resize**

`ST_Resize(geometry, width, height)`, `ST_Resize(geometry, width, height, gridx, gridy)`, `ST_Resize(geometry, width, height, scalex, skewx, skewy)`, `ST_Resize(geometry, width, height, gridx, gridy, scalex, skewx, skewy)`, `ST_Resize(geometry, width, height, gridx, gridy, scalex, skewx, skewy, SRID)`, `ST_Resize(geometry, width, height, gridx, gridy, scalex, skewx, skewy, SRID, nearest_neighbor)`, `ST_Resize(geometry, width, height, gridx, gridy, scalex, skewx, skewy, SRID, nearest_neighbor, lanczos)`, `ST_Resize(geometry, width, height, gridx, gridy, scalex, skewx, skewy, SRID, nearest_neighbor, lanczos, maxerr)`

`nearest_neighbor`: Bilinear, Cubic, Cubic-Spline, Lanczos, NearestNeighbor

`maxerr`: 0.125

**Note**

GDAL Warp resampling methods

GDAL 2.0.0, GDAL 1.6.1

ST\_Resize(geometry, width, height, SRID, nearest\_neighbor, lanczos, maxerr)

**ST\_ReduceResolution**

```
SELECT
  ST_Width(orig) AS orig_width,
  ST_Width(reduce_100) AS new_width
FROM (
  SELECT
    rast AS orig,
    ST_Resample(rast,100,100) AS reduce_100
  FROM aerials.boston
  WHERE ST_Intersects(rast,
    ST_Transform(
      ST_MakeEnvelope(-71.128, 42.2392, -71.1277, 42.2397, 4326), 26986)
    )
  LIMIT 1
) AS foo;
```

| orig_width | new_width |
|------------|-----------|
| 200        | 100       |

**ST\_Rescale**

[ST\\_Rescale](#), [ST\\_Resize](#), [ST\\_Transform](#)

## 12.7.8 ST\_Rescale

`ST_Rescale` — Rescale a raster to a new pixel size. The new pixel size is defined by the `scalex` and `scaley` parameters. The `algorithm` parameter specifies the resampling method to use. The `maxerr` parameter specifies the threshold for transformation approximation by the resampling algorithm (in pixel units). A default of 0.125 is used if no `maxerr` is specified, which is the same value used in GDAL `gdalwarp` utility. If set to zero, no approximation takes place.

### Synopsis

```
raster ST_Rescale(raster rast, double precision scalex, text algorithm=NearestNeighbour, double precision maxerr=0.125);
raster ST_Rescale(raster rast, double precision scalex, double precision scaley, text algorithm=NearestNeighbour, double precision maxerr=0.125);
```

**ST\_Rescale**

`ST_Rescale` — Rescale a raster to a new pixel size. The new pixel size is defined by the `scalex` and `scaley` parameters. The `algorithm` parameter specifies the resampling method to use. The `maxerr` parameter specifies the threshold for transformation approximation by the resampling algorithm (in pixel units). A default of 0.125 is used if no `maxerr` is specified, which is the same value used in GDAL `gdalwarp` utility. If set to zero, no approximation takes place.

`scalex` and `scaley` define the new pixel size. `scaley` must often be negative to get well oriented raster.

```
scalex <double>, scaley <double>, algorithm <text>=NearestNeighbour, maxerr <double>=0.125);
scalex <double>, scaley <double>, algorithm <text>=NearestNeighbour, maxerr <double>=0.125);
```

`maxerr` is the threshold for transformation approximation by the resampling algorithm (in pixel units). A default of 0.125 is used if no `maxerr` is specified, which is the same value used in GDAL `gdalwarp` utility. If set to zero, no approximation takes place.



### Note

GDAL Warp resampling methods

**Note**



ST\_Rescale  
 ST\_SetScale  
 ST\_SetScale  
 ST\_Rescale  
 ST\_Rescale  
 ST\_SetScale

GDAL 1.6.1

SRID

**ST\_SetScale**

ST\_SetScale

```
-- ST_SetScale
SELECT ST_PixelWidth(ST_AddBand(ST_MakeEmptyRaster(100, 100, 0, 0, 0.001, -0.001, 0, 0, 4269), '8BUI'::text, 1, 0)) width

width
-----
0.001

-- ST_Rescale
SELECT ST_PixelWidth(ST_Rescale(ST_AddBand(ST_MakeEmptyRaster(100, 100, 0, 0, 0.001, -0.001, 0, 0, 4269), '8BUI'::text, 1, 0), 0.0015)) width

width
-----
0.0015
```

**ST\_SetScale**

ST\_Resize, ST\_Resample, ST\_SetScale, ST\_ScaleX, ST\_ScaleY, ST\_Transform

**12.7.9 ST\_Reskew**

ST\_Reskew — NearestNeighbor, Bilinear, Cubic, Cubic-Spline, Lanczos

**Synopsis**

raster **ST\_Reskew**(raster rast, double precision skewxy, text algorithm=NearestNeighbour, double precision maxerr=0.125);  
 raster **ST\_Reskew**(raster rast, double precision skewx, double precision skewy, text algorithm=NearestNeighbour, double precision maxerr=0.125);

**Parameters**

**skewxy**: skew in the y direction. **skewx**: skew in the x direction. **algorithm**: NearestNeighbor, Bilinear, Cubic-Spline, Lanczos. **maxerr**: maximum error allowed. Default is 0.125.



**Note**

GDAL Warp resampling methods

**Note**



ST\_Reskew(**skewxy**, **skewx**, **skewy**, **algorithm**, **maxerr**)  
 ST\_SetSkew(**skewxy**, **skewx**, **skewy**, **algorithm**, **maxerr**)

GDAL 1.6.1  
 SRID

**Examples**

0.0015

```

-- the original raster non-rotated
SELECT ST_Rotation(ST_AddBand(ST_MakeEmptyRaster(100, 100, 0, 0, 0.001, -0.001, 0, 0, 4269) ←
, '8BUI'::text, 1, 0));

-- result
0

-- the reskewed raster raster rotation
SELECT ST_Rotation(ST_Reskew(ST_AddBand(ST_MakeEmptyRaster(100, 100, 0, 0, 0.001, -0.001, ←
0, 0, 4269), '8BUI'::text, 1, 0), 0.0015));

-- result
-0.982793723247329

```

**ST\_Resample, ST\_Rescale, ST\_SetSkew, ST\_SetRotation, ST\_SkewX, ST\_SkewY, ST\_Transform**

**ST\_Resample, ST\_Rescale, ST\_SetSkew, ST\_SetRotation, ST\_SkewX, ST\_SkewY, ST\_Transform**

## 12.7.10 ST\_SnapToGrid

**ST\_SnapToGrid** — Snap a raster to a grid. The grid is defined by the `gridx` and `gridy` parameters. The `algorithm` parameter can be `NearestNeighbor`, `Bilinear`, `Cubic`, `CubicSpline`, `Lanczos`, or `NearestNeighbor8`. The `maxerr` parameter is the maximum error in pixels. The `scalex` and `scaley` parameters are the scale factors for the `gridx` and `gridy` parameters. The `algorithm` parameter can be `NearestNeighbor`, `Bilinear`, `Cubic`, `CubicSpline`, `Lanczos`, or `NearestNeighbor8`.

### Synopsis

raster **ST\_SnapToGrid**(raster rast, double precision gridx, double precision gridy, text algorithm=NearestNeighbour, double precision maxerr=0.125, double precision scalex=DEFAULT 0, double precision scaley=DEFAULT 0);

raster **ST\_SnapToGrid**(raster rast, double precision gridx, double precision gridy, double precision scalex, double precision scaley, text algorithm=NearestNeighbour, double precision maxerr=0.125);

raster **ST\_SnapToGrid**(raster rast, double precision gridx, double precision gridy, double precision scalexy, text algorithm=NearestNeighbour, double precision maxerr=0.125);

**ST\_SnapToGrid**

**ST\_SnapToGrid**(raster rast, double precision gridx, double precision gridy, text algorithm=NearestNeighbour, double precision maxerr=0.125, double precision scalex=DEFAULT 0, double precision scaley=DEFAULT 0);

**ST\_SnapToGrid**(raster rast, double precision gridx, double precision gridy, double precision scalex, double precision scaley, text algorithm=NearestNeighbour, double precision maxerr=0.125);

**ST\_SnapToGrid**(raster rast, double precision gridx, double precision gridy, double precision scalexy, text algorithm=NearestNeighbour, double precision maxerr=0.125);

You can optionally define the pixel size of the new grid with `scalex` and `scaley`.





```
)
SELECT rid, (meta).* FROM bar

  rid | upperleftx | upperlefty | width | height | scalex | scaley | skewx | skewy | srid | ←
-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
   1 |          0 |           0 |    500 |    500 |        1 |       -1 |        0 |        0 |    0 | ←
     |          1 |           |        |        |        |        |        |        |        |        |
   2 |          0 |           0 |    500 |    100 |        1 |       -1 |        0 |        0 |    0 | ←
     |          1 |           |        |        |        |        |        |        |        |        |
   3 |          0 |           0 |    250 |    900 |        1 |       -1 |        0 |        0 |    0 | ←
     |          1 |           |        |        |        |        |        |        |        |        |
(3 rows)
```

**ST\_Resample, ST\_Rescale, ST\_Reskew, ST\_SnapToGrid**

### 12.7.12 ST\_Transform

ST\_Transform — Reprojects a raster to a different coordinate system. The raster is reprojected to the specified SRS (srid) using the specified algorithm. The output raster has the same dimensions as the input raster, but the pixel values are reprojected to the new coordinate system. The output raster is in the same format as the input raster. The output raster is in the same format as the input raster. The output raster is in the same format as the input raster.

#### Synopsis

```
raster ST_Transform(raster rast, integer srid, text algorithm=NearestNeighbor, double precision maxerr=0.125, double precision
scalex, double precision scaley);
raster ST_Transform(raster rast, integer srid, double precision scalex, double precision scaley, text algorithm=NearestNeighbor,
double precision maxerr=0.125);
raster ST_Transform(raster rast, raster alignto, text algorithm=NearestNeighbor, double precision maxerr=0.125);
```

**ST\_Transform**

ST\_Transform(raster rast, integer srid, text algorithm=NearestNeighbor, double precision maxerr=0.125, double precision scalex, double precision scaley);

ST\_Transform(raster rast, integer srid, double precision scalex, double precision scaley, text algorithm=NearestNeighbor, double precision maxerr=0.125);

ST\_Transform(raster rast, raster alignto, text algorithm=NearestNeighbor, double precision maxerr=0.125);

ST\_Transform(raster rast, integer srid, text algorithm=NearestNeighbor, double precision maxerr=0.125, double precision scalex, double precision scaley);

ST\_Transform(raster rast, integer srid, double precision scalex, double precision scaley, text algorithm=NearestNeighbor, double precision maxerr=0.125);

ST\_Transform(raster rast, raster alignto, text algorithm=NearestNeighbor, double precision maxerr=0.125);

ST\_Transform(raster rast, integer srid, text algorithm=NearestNeighbor, double precision maxerr=0.125, double precision scalex, double precision scaley);

ST\_Transform(raster rast, integer srid, double precision scalex, double precision scaley, text algorithm=NearestNeighbor, double precision maxerr=0.125);

ST\_Transform(raster rast, raster alignto, text algorithm=NearestNeighbor, double precision maxerr=0.125);

```
ST_SetSRID(  
SRID  
alignto  
= TRUE  
(ST_SameAlign  
= TRUE
```

**Note**

```
PROJJO  
PostGIS  
so  
PROJJO  
libproj.dll  
PostgreSQL
```



**Warning**

When transforming a coverage of tiles, you almost always want to use a reference raster to insure same alignment and no gaps in your tiles as demonstrated in example: Variant 3.

```
GDAL 1.6.1  
ST_Transform(rast, alignto)
```

**Example**

```
SELECT ST_Width(mass_stm) As w_before, ST_Width(wgs_84) As w_after,  
ST_Height(mass_stm) As h_before, ST_Height(wgs_84) As h_after  
FROM  
( SELECT rast As mass_stm, ST_Transform(rast,4326) As wgs_84  
, ST_Transform(rast,4326, 'Bilinear') AS wgs_84_bilin  
FROM aerials.o_2_boston  
WHERE ST_Intersects(rast,  
ST_Transform(ST_MakeEnvelope(-71.128, 42.2392,-71.1277, ←  
42.2397, 4326),26986) )  
LIMIT 1) As foo;  
  
w_before | w_after | h_before | h_after  
-----+-----+-----+-----  
200 | 228 | 200 | 170
```

|   |  |   |
|---|--|---|
|  <p>             ST_MakeEmptyRaster(2, 2, -500000, 600000, 100, -100, 0, 0, 2163), 1, '16BUI', 1, 0) AS rast UNION ALL<br/>             ST_MakeEmptyRaster(2, 2, -499800, 600000, 100, -100, 0, 0, 2163), 1, '16BUI', 2, 0) AS rast UNION ALL<br/>             ST_MakeEmptyRaster(2, 2, -499600, 600000, 100, -100, 0, 0, 2163), 1, '16BUI', 3, 0) AS rast UNION ALL<br/>             ST_MakeEmptyRaster(2, 2, -500000, 599800, 100, -100, 0, 0, 2163), 1, '16BUI', 10, 0) AS rast UNION ALL<br/>             ST_MakeEmptyRaster(2, 2, -499800, 599800, 100, -100, 0, 0, 2163), 1, '16BUI', 20, 0) AS rast UNION ALL<br/>             ST_MakeEmptyRaster(2, 2, -499600, 599800, 100, -100, 0, 0, 2163), 1, '16BUI', 30, 0) AS rast UNION ALL<br/>             ST_MakeEmptyRaster(2, 2, -500000, 599600, 100, -100, 0, 0, 2163), 1, '16BUI', 100, 0) AS rast UNION ALL<br/>             ST_MakeEmptyRaster(2, 2, -499800, 599600, 100, -100, 0, 0, 2163), 1, '16BUI', 200, 0) AS rast UNION ALL<br/>             ST_MakeEmptyRaster(2, 2, -499600, 599600, 100, -100, 0, 0, 2163), 1, '16BUI', 300, 0) AS rast         </p> |  <p>             WGS84<br/>             ST_Transform(rast, 4269) AS alignto         </p> |  <p>             NN<br/>             Bilinear<br/>             WGS84<br/>             ST_Transform(rast, 4269) AS alignto<br/>             ST_Transform(rast, alignto) AS aligned         </p> |
|---|--|---|

**ST\_Transform(rast, srid)**

ST\_Transform(rast, alignto)

```

WITH foo AS (
  SELECT 0 AS rid, ST_AddBand(ST_MakeEmptyRaster(2, 2, -500000, 600000, 100, -100, 0, 0, 2163), 1, '16BUI', 1, 0) AS rast UNION ALL
  SELECT 1, ST_AddBand(ST_MakeEmptyRaster(2, 2, -499800, 600000, 100, -100, 0, 0, 2163), 1, '16BUI', 2, 0) AS rast UNION ALL
  SELECT 2, ST_AddBand(ST_MakeEmptyRaster(2, 2, -499600, 600000, 100, -100, 0, 0, 2163), 1, '16BUI', 3, 0) AS rast UNION ALL

  SELECT 3, ST_AddBand(ST_MakeEmptyRaster(2, 2, -500000, 599800, 100, -100, 0, 0, 2163), 1, '16BUI', 10, 0) AS rast UNION ALL
  SELECT 4, ST_AddBand(ST_MakeEmptyRaster(2, 2, -499800, 599800, 100, -100, 0, 0, 2163), 1, '16BUI', 20, 0) AS rast UNION ALL
  SELECT 5, ST_AddBand(ST_MakeEmptyRaster(2, 2, -499600, 599800, 100, -100, 0, 0, 2163), 1, '16BUI', 30, 0) AS rast UNION ALL

  SELECT 6, ST_AddBand(ST_MakeEmptyRaster(2, 2, -500000, 599600, 100, -100, 0, 0, 2163), 1, '16BUI', 100, 0) AS rast UNION ALL
  SELECT 7, ST_AddBand(ST_MakeEmptyRaster(2, 2, -499800, 599600, 100, -100, 0, 0, 2163), 1, '16BUI', 200, 0) AS rast UNION ALL
  SELECT 8, ST_AddBand(ST_MakeEmptyRaster(2, 2, -499600, 599600, 100, -100, 0, 0, 2163), 1, '16BUI', 300, 0) AS rast
), bar AS (
  SELECT
    ST_Transform(rast, 4269) AS alignto
  FROM foo
  LIMIT 1
), baz AS (
  SELECT
    rid,
    rast,
    ST_Transform(rast, 4269) AS not_aligned,
    ST_Transform(rast, alignto) AS aligned

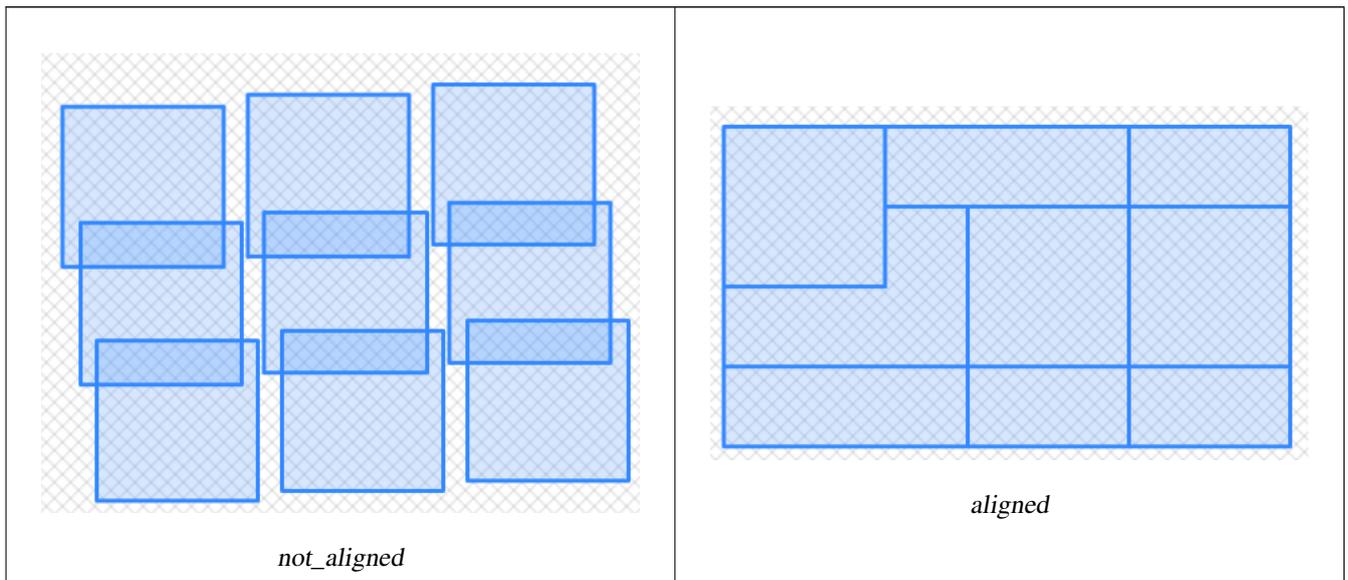
```

```

FROM foo
CROSS JOIN bar
)
SELECT
  ST_SameAlignment(rast) AS rast,
  ST_SameAlignment(not_aligned) AS not_aligned,
  ST_SameAlignment(aligned) AS aligned
FROM baz

rast | not_aligned | aligned
-----+-----+-----
t   | f           | t

```



**ST\_SameAlignment**

[ST\\_Transform](#), [ST\\_SetSRID](#)

## 12.8 [ST\\_SameAlignment](#)

### 12.8.1 [ST\\_SetBandNoDataValue](#)

**ST\_SetBandNoDataValue** — NODATA value for a raster band. The function sets the NODATA value for a raster band. The function returns the raster with the NODATA value set to the specified value. The function is implemented as follows: `ST_SetBandNoDataValue(raster rast, double precision nodatavalue);` `ST_SetBandNoDataValue(raster rast, integer band, double precision nodatavalue, boolean forcechecking=false);`

#### Synopsis

raster **ST\_SetBandNoDataValue**(raster rast, double precision nodatavalue);  
raster **ST\_SetBandNoDataValue**(raster rast, integer band, double precision nodatavalue, boolean forcechecking=false);

**ST\_SetBandNoDataValue**

`ST_SetBandNoDataValue(rast, 1, 254)`  
 Updates the specified band of the raster to NoData. The raster must be of type `Integer` or `Real`. The band number is 1-based. The NoData value is 254 for `Integer` and `NaN` for `Real`.

**ST\_SetBandNoDataValue**

```
-- ST_SetBandNoDataValue(rast, 1, 254);
UPDATE dummy_rast
  SET rast = ST_SetBandNoDataValue(rast,1, 254)
WHERE rid = 2;

-- ST_SetBandNoDataValue(rast, 1, 2, 3, 108);
UPDATE dummy_rast
  SET rast =
    ST_SetBandNoDataValue(
      ST_SetBandNoDataValue(
        ST_SetBandNoDataValue(
          rast,1, 254)
        ,2,99),
      3,108)
  WHERE rid = 2;

-- ST_SetBandNoDataValue(rast, 1, NULL);
UPDATE dummy_rast
  SET rast = ST_SetBandNoDataValue(rast,1, NULL)
WHERE rid = 2;
```

**ST\_SetBandNoDataValue**

[ST\\_BandNoDataValue](#), [ST\\_NumBands](#)

**12.8.2 ST\_SetBandIsNoData**

`ST_SetBandIsNoData(rast, 1)`  
 Returns a boolean raster where the specified band is NoData. The raster must be of type `Integer` or `Real`. The band number is 1-based.

**Synopsis**

raster `ST_SetBandIsNoData`(raster rast, integer band=1);

**ST\_SetBandIsNoData**

`ST_SetBandIsNoData(rast, 1)`  
 Returns a boolean raster where the specified band is NoData. The raster must be of type `Integer` or `Real`. The band number is 1-based.

ST\_BandIsNoData  
 &#xd568;&#xc218;&#xb97c;&#xd638;&#xcd9c;&#xd574;&#xc11c;&#xb098;&#xc628;&#xacb0;&#xacfc;&#xbb3c;&#xc774;&#xb2ec;&#xb77c;&#xc9c8;&#xb54c;&#xb9d0;&#xc785;&#xb2c8;&#xb2e4;.

2.0.0 &#xbc84;&#xc804;&#xbd80;&#xd130;&#xc0ac;&#xc6a9;&#xd560;&#xc218;&#xc788;&#xc2b5;&#xb2c8;&#xb2e4;.

## &#xc608;&#xc2dc;

```
-- &#xb798;&#xc2a4;&#xd130;&#xc5f4;&#xd55c;&#xa1c;&#xb97c;&#xac00;&#xc9c4;&#xc00;&#xc9dc;&#xd14c;&#xc774;&#xbe14;&#xc744;&#xc0dd;&#xc131;&#xd569;&#xb2c8;&#xb2e4;.
create table dummy_rast (rid integer, rast raster);

-- &#xbc34;&#xb4dc;&#xb450;&#xa1c;&#xc640;&#xd53d;&#xc140;/&#bc34;&#xb4dc;&#xd558;&#xb098;&#xb97c;&#xac00;&#xc9c4;&#xb798;&#xc2a4;&#xd130;&#xb97c;&#xcd94;&#xac00;&#xd569;&#xb2c8;&#xb2e4;. &#xccab;&#bc88;&#xc9f8;&#bc34;&#xb4dc;&#xc758; nodatavalue = pixel value = 3&#xc785;&#xb2c8;&#xb2e4;.
-- &#xb450;&#bc88;&#xc9f8;&#bc34;&#xb4dc;&#xc758; nodatavalue = 13, pixel value = 4 &#xc785;&#xb2c8;&#xb2e4;.
insert into dummy_rast values(1,
(
'01' -- little endian (uint8 ndr)
||
'0000' -- version (uint16 0)
||
'0200' -- nBands (uint16 0)
||
'17263529ED684A3F' -- scaleX (float64 0.000805965234044584)
||
'F9253529ED684ABF' -- scaleY (float64 -0.00080596523404458)
||
'1C9F33CE69E352C0' -- ipX (float64 -75.5533328537098)
||
'718F0E9A27A44840' -- ipY (float64 49.2824585505576)
||
'ED50EB853EC32B3F' -- skewX (float64 0.000211812383858707)
||
'7550EB853EC32B3F' -- skewY (float64 0.000211812383858704)
||
'E6100000' -- SRID (int32 4326)
||
'0100' -- width (uint16 1)
||
'0100' -- height (uint16 1)
||
'4' -- hasnodatavalue &#xac12;&#xc774;&#xcc38;&#xc73c;&#xb85c;, isnodata &#xac12;&#xc774;&#xcc38;&#xc774;&#xc5b4;&#xc57c;&#xd558;&#xb294;&#xb370;) &#xac70;&#xc9d3;&#xc73c;&#xb85c;&#xc124;&#xc815;&#xb410;&#xc2b5;&#xb2c8;&#xb2e4;.
||
'2' -- &#xccab;&#bc88;&#xc9f8;&#bc34;&#xb4dc;&#xc720;&#xd615;(4BUI)
||
'03' -- novalue==3
||
'03' -- pixel(0,0)==3 (same that nodata)
||
'0' -- hasnodatavalue &#xac12;&#xc774;&#xac70;&#xc9d3;&#xc73c;&#xb85c;&#xc124;&#xc815;&#xb410;&#xc2b5;&#xb2c8;&#xb2e4;.
||
'5' -- &#xb450;&#bc88;&#xc9f8;&#bc34;&#xb4dc;&#xc720;&#xd615;(16BSI)
||
'0D00' -- novalue==13
```

```

||
'0400' -- pixel(0,0)==4
)::raster
);

select st_bandisnodata(rast, 1) from dummy_rast where rid = 1; -- Expected false
select st_bandisnodata(rast, 1, TRUE) from dummy_rast where rid = 1; -- Expected true

-- isnodata &#xd50c;&#xb798;&#xadf8;&#xac00; ↔
  &#xc9c0;&#xc800;&#xbd84;&#xd569;&#xb2c8;&#xb2e4;. &#xc774; ↔
  &#xd50c;&#xb798;&#xadf8;&#xb97c; &#xcc38;&#xc73c;&#xb85c; ↔
  &#xc124;&#xc815;&#xd558;&#xaca0;&#xc2b5;&#xb2c8;&#xb2e4;.
update dummy_rast set rast = st_setbandisnodata(rast, 1) where rid = 1;

select st_bandisnodata(rast, 1) from dummy_rast where rid = 1; -- Expected true

```

**&#xcc38;&#xace0;**

[ST\\_BandNoDataValue](#), [ST\\_NumBands](#), [ST\\_SetBandNoDataValue](#), [ST\\_BandIsNoData](#)

### 12.8.3 ST\_SetBandPath

ST\_SetBandPath — Update the external path and band number of an out-db band

#### Synopsis

raster **ST\_SetBandPath**(raster rast, integer band, text outdbpath, integer outdbindex, boolean force=false);

**&#xc124;&#xba85;**

Updates an out-db band's external raster file path and external band number.



#### Note

If `force` is set to true, no tests are done to ensure compatibility (e.g. alignment, pixel support) between the external raster file and the PostGIS raster. This mode is intended for file system changes where the external raster resides.



#### Note

Internally, this method replaces the PostGIS raster's band at index `band` with a new band instead of updating the existing path information.

Availability: 2.5.0

**&#xc608;&#xc2dc;**

```

WITH foo AS (
  SELECT
    ST_AddBand(NULL::raster, '/home/pele/devel/geo/postgis-git/raster/test/regress/
      loader/Projected.tif', NULL::int[]) AS rast
)
SELECT
  1 AS query,
  *
FROM ST_BandMetadata(
  (SELECT rast FROM foo),
  ARRAY[1,3,2]::int[]
)
UNION ALL
SELECT
  2,
  *
FROM ST_BandMetadata(
  (
    SELECT
      ST_SetBandPath(
        rast,
        2,
        '/home/pele/devel/geo/postgis-git/raster/test/regress/loader/Projected2.tif'
      ) AS rast
    FROM foo
  ),
  ARRAY[1,3,2]::int[]
)
ORDER BY 1, 2;

```

| query | bandnum | pixeltype | nodatavalue | isoutdb | path   | outdbbandnum |
|-------|---------|-----------|-------------|---------|--|--------------|
| 1     | 1       | 8BUI      |             | t       | /home/pele/devel/geo/postgis-git/raster/test/regress/loader/Projected.tif  | 1            |
| 1     | 2       | 8BUI      |             | t       | /home/pele/devel/geo/postgis-git/raster/test/regress/loader/Projected.tif  | 2            |
| 1     | 3       | 8BUI      |             | t       | /home/pele/devel/geo/postgis-git/raster/test/regress/loader/Projected.tif  | 3            |
| 2     | 1       | 8BUI      |             | t       | /home/pele/devel/geo/postgis-git/raster/test/regress/loader/Projected.tif  | 1            |
| 2     | 2       | 8BUI      |             | t       | /home/pele/devel/geo/postgis-git/raster/test/regress/loader/Projected2.tif | 1            |
| 2     | 3       | 8BUI      |             | t       | /home/pele/devel/geo/postgis-git/raster/test/regress/loader/Projected.tif  | 3            |

**ST\_BandMetadata, ST\_SetBandIndex**

**ST\_BandMetadata, ST\_SetBandIndex**

## 12.8.4 ST\_SetBandIndex

ST\_SetBandIndex — Update the external band number of an out-db band





2.2.0 ST\_Count(rastertable, rastercolumn, ...) ST\_CountAgg

2.0.0 ST\_CountAgg

**ST\_CountAgg**

```
-- ST_CountAgg( ST_SetBandNoDataValue( rast, 249 ) As exclude_nodata,
-- ST_Count( ST_SetBandNoDataValue( rast, 249 ), false ) As include_nodata
SELECT rid, ST_Count( ST_SetBandNoDataValue( rast, 249 ) As exclude_nodata,
        ST_Count( ST_SetBandNoDataValue( rast, 249 ), false ) As include_nodata
FROM dummy_rast WHERE rid=2;
```

| rid | exclude_nodata | include_nodata |
|-----|----------------|----------------|
| 2   | 23             | 25             |

**ST\_CountAgg**

[ST\\_CountAgg](#), [ST\\_SummaryStats](#), [ST\\_SetBandNoDataValue](#)

## 12.9.2 ST\_CountAgg

**ST\_CountAgg** — ST\_CountAgg( raster rast, integer nband, boolean exclude\_nodata\_value, double precision sample\_percent);

**ST\_CountAgg** — ST\_CountAgg( raster rast, integer nband, boolean exclude\_nodata\_value);

**ST\_CountAgg** — ST\_CountAgg( raster rast, boolean exclude\_nodata\_value);

### Synopsis

bigint **ST\_CountAgg**(setof raster rast, integer nband, boolean exclude\_nodata\_value, double precision sample\_percent);

bigint **ST\_CountAgg**(setof raster rast, integer nband, boolean exclude\_nodata\_value);

bigint **ST\_CountAgg**(setof raster rast, boolean exclude\_nodata\_value);

**ST\_CountAgg**

ST\_CountAgg( raster rast, integer nband, boolean exclude\_nodata\_value, double precision sample\_percent);

ST\_CountAgg( raster rast, integer nband, boolean exclude\_nodata\_value);

ST\_CountAgg( raster rast, boolean exclude\_nodata\_value);

exclude\_nodata\_value

nodata

sample\_percent

2.2.0 ST\_CountAgg

2.0.0 ST\_CountAgg

**ST\_Count**

```

WITH foo AS (
  SELECT
    rast.rast
  FROM (
    SELECT ST_SetValue(
      ST_SetValue(
        ST_SetValue(
          ST_AddBand(
            ST_MakeEmptyRaster(10, 10, 10, 10, 2, 2, 0, 0,
              0,0)
            , 1, '64BF', 0, 0
          )
          , 1, 1, 1, -10
        )
        , 1, 5, 4, 0
      )
      , 1, 5, 5, 3.14159
    ) AS rast
  ) AS rast
  FULL JOIN (
    SELECT generate_series(1, 10) AS id
  ) AS id
  ON 1 = 1
)
SELECT
  ST_CountAgg(rast, 1, TRUE)
FROM foo;

 st_countagg
-----
              20
(1 row)

```

**ST\_Count**

[ST\\_Count](#), [ST\\_SummaryStats](#), [ST\\_SetBandNoDataValue](#)

**12.9.3 ST\_Histogram**

**ST\_Histogram** — (bin; width; nband; exclude\_nodata\_value; bins; width; right;)

**ST\_Histogram** (raster rast, integer nband=1, boolean exclude\_nodata\_value=true, integer bins=autocomputed, double precision[] width=NULL, boolean right=false);

**ST\_Histogram** (raster rast, integer nband, integer bins, double precision[] width=NULL, boolean right=false);

**ST\_Histogram** (raster rast, integer nband, boolean exclude\_nodata\_value, integer bins, boolean right);

**ST\_Histogram** (raster rast, integer nband, integer bins, boolean right);

**Synopsis**

setof record **ST\_Histogram**(raster rast, integer nband=1, boolean exclude\_nodata\_value=true, integer bins=autocomputed, double precision[] width=NULL, boolean right=false);

setof record **ST\_Histogram**(raster rast, integer nband, integer bins, double precision[] width=NULL, boolean right=false);

setof record **ST\_Histogram**(raster rast, integer nband, boolean exclude\_nodata\_value, integer bins, boolean right);

setof record **ST\_Histogram**(raster rast, integer nband, integer bins, boolean right);



|   |  |       |  |       |  |   |  |      |
|---|--|-------|--|-------|--|---|--|------|
| 3 |  | 100.4 |  | 138.8 |  | 5 |  | 0.2  |
| 3 |  | 138.8 |  | 177.2 |  | 4 |  | 0.16 |
| 3 |  | 177.2 |  | 215.6 |  | 1 |  | 0.04 |
| 3 |  | 215.6 |  | 254   |  | 4 |  | 0.16 |

**ST\_Histogram**(raster rast, integer nband=1, boolean exclude\_nodata\_value=true, double precision[] quantiles=NULL);

```
SELECT (stats).*
FROM (SELECT rid, ST_Histogram(rast, 2,6) As stats
      FROM dummy_rast
      WHERE rid=2) As foo;
```

| min        | max | count      | percent |   |  |      |
|------------|-----|------------|---------|---|--|------|
| 78         |     | 107.333333 |         | 9 |  | 0.36 |
| 107.333333 |     | 136.666667 |         | 6 |  | 0.24 |
| 136.666667 |     | 166        |         | 0 |  | 0    |
| 166        |     | 195.333333 |         | 4 |  | 0.16 |
| 195.333333 |     | 224.666667 |         | 1 |  | 0.04 |
| 224.666667 |     | 254        |         | 5 |  | 0.2  |

(6 rows)

```
-- ST_Histogram(rast, 2,6,ARRAY[0.5,1,4,100,5]) As stats
-- ST_Histogram(rast, 2,6,ARRAY[0.5,1,4,100,5]) As stats
```

```
SELECT (stats).*
FROM (SELECT rid, ST_Histogram(rast, 2,6,ARRAY[0.5,1,4,100,5]) As stats
      FROM dummy_rast
      WHERE rid=2) As foo;
```

| min   | max | count | percent |    |  |          |
|-------|-----|-------|---------|----|--|----------|
| 78    |     | 78.5  |         | 1  |  | 0.08     |
| 78.5  |     | 79.5  |         | 1  |  | 0.04     |
| 79.5  |     | 83.5  |         | 0  |  | 0        |
| 83.5  |     | 183.5 |         | 17 |  | 0.0068   |
| 183.5 |     | 188.5 |         | 0  |  | 0        |
| 188.5 |     | 254   |         | 6  |  | 0.003664 |

(6 rows)

**ST\_Quantile**

[ST\\_Count](#), [ST\\_SummaryStats](#), [ST\\_SummaryStatsAgg](#)

## 12.9.4 ST\_Quantile

**ST\_Quantile** — (population) (quantile) (percentile)

ST\_Quantile(raster rast, integer nband=1, boolean exclude\_nodata\_value=true, double precision[] quantiles=NULL);

### Synopsis

setof record **ST\_Quantile**(raster rast, integer nband=1, boolean exclude\_nodata\_value=true, double precision[] quantiles=NULL);

setof record **ST\_Quantile**(raster rast, double precision[] quantiles);

```

setof record ST_Quantile(raster rast, integer nband, double precision[] quantiles);
double precision ST_Quantile(raster rast, double precision quantile);
double precision ST_Quantile(raster rast, boolean exclude_nodata_value, double precision quantile=NULL);
double precision ST_Quantile(raster rast, integer nband, double precision quantile);
double precision ST_Quantile(raster rast, integer nband, boolean exclude_nodata_value, double precision quantile);
double precision ST_Quantile(raster rast, integer nband, double precision quantile);

```

### Note

(population) (percentile) 25%, 50%, 75% (percentile)



#### Note

exclude\_nodata\_value NODATA

Changed: 3.1.0 Removed ST\_Quantile(table\_name, column\_name) variant.

2.0.0

### Note

```

UPDATE dummy_rast SET rast = ST_SetBandNoDataValue(rast,249) WHERE rid=2;
--

```

```

SELECT (pvq).*
FROM (SELECT ST_Quantile(rast, ARRAY[0.25,0.75]) As pvq
      FROM dummy_rast WHERE rid=2) As foo
ORDER BY (pvq).quantile;

```

| quantile | value |
|----------|-------|
| 0.25     | 253   |
| 0.75     | 254   |

```

SELECT ST_Quantile(rast, 0.75) As value
FROM dummy_rast WHERE rid=2;

```

```

value
-----
254

```

```

--

```

```

SELECT rid, (ST_Quantile(rast,2)).* As pvc
FROM o_4_boston
WHERE ST_Intersects(rast,

```

```

        ST_GeomFromText('POLYGON((224486 892151,224486 892200,224706 892200,224706
        892151,224486 892151))',26986)
    )
ORDER BY value, quantile,rid
;

```

| rid | quantile | value |
|-----|----------|-------|
| 1   | 0        | 0     |
| 2   | 0        | 0     |
| 14  | 0        | 1     |
| 15  | 0        | 2     |
| 14  | 0.25     | 37    |
| 1   | 0.25     | 42    |
| 15  | 0.25     | 47    |
| 2   | 0.25     | 50    |
| 14  | 0.5      | 56    |
| 1   | 0.5      | 64    |
| 15  | 0.5      | 66    |
| 2   | 0.5      | 77    |
| 14  | 0.75     | 81    |
| 15  | 0.75     | 87    |
| 1   | 0.75     | 94    |
| 2   | 0.75     | 106   |
| 14  | 1        | 199   |
| 1   | 1        | 244   |
| 2   | 1        | 255   |
| 15  | 1        | 255   |

**ST\_Count**

[ST\\_Count](#), [ST\\_SummaryStats](#), [ST\\_SummaryStatsAgg](#), [ST\\_SetBandNoData Value](#)

## 12.9.5 ST\_SummaryStats

**ST\_SummaryStats** — Returns a table with columns: count, sum, mean, stddev, min, max, summarystats. The table has columns: count, sum, mean, stddev, min, max, summarystats. The table has columns: count, sum, mean, stddev, min, max, summarystats. The table has columns: count, sum, mean, stddev, min, max, summarystats.

### Synopsis

```

raster ST_SetScale(raster rast, float8 xy);
raster ST_SetScale(raster rast, float8 x, float8 y);

```

**ST\_SummaryStats**

Returns a table with columns: count, sum, mean, stddev, min, max, summarystats. The table has columns: count, sum, mean, stddev, min, max, summarystats. The table has columns: count, sum, mean, stddev, min, max, summarystats. The table has columns: count, sum, mean, stddev, min, max, summarystats.



**Note**

nodata &#x12;&#x774; &#x544;&#xb2cc; &#xd53d;&#xc140; &#xac12;&#xb9cc; &#xcc98;&#xb9ac;&#xd569;&#xb2c8;&#xb2e4;. &#xaa8;&#xb4e0; &#xd53d;&#xc140;&#xc758; &#xac1c;&#xc218;&#xb97c; &#xad6c;&#xd558;&#xb824;&#xba74; exclude\_nodata\_value &#xb97c; &#xac70;&#xc9d3;&#xc73c;&#xb85c; &#xc124;&#xc815;&#xd558;&#xc2ed;&#xc2dc;&#xc624;.



**Note**

&#xaa8;&#xb4e0; &#xd53d;&#xc140;&#xc744; &#xc0d8;&#xd50c;&#xb9c1;&#xd560; &#xac83;&#xc785;&#xb2c8;&#xb2e4;. &#xb354; &#xbe60;&#xb978; &#xc18d;&#xb3c4;&#xb97c; &#xc6d0;&#xd55c;&#xb2e4;&#xba74;. sample\_percent &#xb97c; 1&#xbcf4;&#xb2e4; &#xc791;&#xc740; &#xac12;&#xc73c;&#xb85c; &#xc124;&#xc815;&#xd558;&#xc2ed;&#xc2dc;&#xc624;.

2.2.0 &#xbc84;&#xc804;&#xbd80;&#xd130; &#xb354; &#xc774;&#xc0c1; ST\_SummaryStats(rastertable, rastercolumn, ...) &#xbcc0;&#xc885; &#xd568;&#xc218;&#xb97c; &#xc9c0;&#xc6d0;&#xd558;&#xc9c0; &#xc54a;&#xc2b5;&#xb2c8;&#xb2e4;. &#xb300;&#xc2e0; **ST\_SummaryStatsAgg** &#xd568;&#xc218;&#xb97c; &#xc774;&#xc6a9;&#xd558;&#xc2ed;&#xc2dc;&#xc624;.

2.0.0 &#xbc84;&#xc804;&#xbd80;&#xd130; &#xc0ac;&#xc6a9;&#xd560; &#xc218; &#xc788;&#xc2b5;&#xb2c8;&#xb2e4;.

**&#xc608;&#xc2dc;: &#xb2e8;&#xc77c; &#xb798;&#xc2a4;&#xd130; &#xd0c0;&#xc77c;**

```
SELECT rid, band, (stats).*
FROM (SELECT rid, band, ST_SummaryStats(rast, band) As stats
      FROM dummy_rast CROSS JOIN generate_series(1,3) As band
      WHERE rid=2) As foo;
```

| rid | band | count | sum  | mean       | stddev    | min | max |
|-----|------|-------|------|------------|-----------|-----|-----|
| 2   | 1    | 23    | 5821 | 253.086957 | 1.248061  | 250 | 254 |
| 2   | 2    | 25    | 3682 | 147.28     | 59.862188 | 78  | 254 |
| 2   | 3    | 25    | 3290 | 131.6      | 61.647384 | 62  | 254 |

**&#xc608;&#xc2dc;: &#xad00;&#xc2ec; &#xac74;&#xbb3c;&#xacfc; &#xad50;&#xcc28;&#xd558;&#xb294; &#xd53d;&#xc140;&#xc694;&#xc57d;**

PostGIS &#xc708;&#xb3c4;&#xc6b0; 64&#xbe44;&#xd2b8; &#xbc84;&#xc804;&#xc5d0;&#xc11c; &#xaa8;&#xb4e0; &#xbcf4;&#xac74;&#xbb3c;&#xb4e4;&#xacfc; &#xd56d;&#xacf5;&#xc0ac;&#xc9c4; &#xd0c0;&#xc77c;&#xb4e4;(&#xac01;&#xac01; &#xac74;&#xbb3c; &#xb808;&#xcdf54;&#xb4dc; 102,000&#xac1c;. 150x150 &#xd53d;&#xc140; &#xd06c;&#xae30;&#xc758; &#xd0c0;&#xc77c; 134,000&#xac1c;)&#xc744; &#xcc98;&#xb9ac;&#xd558;&#xb294; &#xc774; &#xc608;&#xc2dc;&#xac00; 574&#xbc00;&#xb9ac;&#xcd08; &#xac78;&#xb838;&#xc2b5;&#xb2c8;&#xb2e4;.

```
WITH
-- &#xad00;&#xc2ec; &#xd53c;&#xcc98;
  feat AS (SELECT gid As building_id, geom_26986 As geom FROM buildings AS b
           WHERE gid IN(100, 103,150)
           ),
-- builds&#xc758; &#xacbd;&#xacc4;&#xc120;&#xc5d0; &#xb9de;&#xcdb0; ↔
  &#xb798;&#xc2a4;&#xd130; &#xd0c0;&#xc77c;&#xc758; &#xb3c4;&#xb4dc; 2&#xb97c; ↔
  &#xc798;&#xb77c; &#xb0b8; &#xb2e4;&#xc74c;
-- &#xc774; &#xc798;&#xb77c;&#xb0b8; &#xc9c0;&#xc5ed;&#xc758; &#xd1b5;&#xacc4;&#xb97c; ↔
  &#xc5bb;&#xc2b5;&#xb2c8;&#xb2e4;.
  b_stats AS
  (SELECT building_id, (stats).*
   FROM (SELECT building_id, ST_SummaryStats(ST_Clip(rast,2,geom)) As stats
         FROM aerials.boston
```

```

        INNER JOIN feat
        ON ST_Intersects(feats.geom,rast)
    ) As foo
  )
-- &#xb9c8;&#xc9c0;&#xb9c9;&#xc73c;&#xb85c; &#xd1b5;&xacc4;&#xb97c; <-
  &#xc694;&#xc57d;&#xd569;&#xb2c8;&#xb2e4;.
SELECT building_id, SUM(count) As num_pixels
  , MIN(min) As min_pval
  , MAX(max) As max_pval
  , SUM(mean*count)/SUM(count) As avg_pval
  FROM b_stats
WHERE count
> 0
  GROUP BY building_id
  ORDER BY building_id;
building_id | num_pixels | min_pval | max_pval | avg_pval
-----+-----+-----+-----+-----
      100 |      1090 |         1 |        255 | 61.0697247706422
      103 |         655 |         7 |        182 | 70.5038167938931
      150 |         895 |         2 |        252 | 185.642458100559

```

## &#xc608;&#xc2dc;: &#xb798;&#xc2a4;&#xd130; &#xcee4;&#xbc84;&#xb9ac;&#xc9c0;

```

-- &#xac01; &#xbc34;&#xb4dc;&#xc5d0; &#xb300;&#xd55c; &#xd1b5;&xacc4; --
SELECT band, (stats).*
FROM (SELECT band, ST_SummaryStats('o_4_boston','rast', band) As stats
      FROM generate_series(1,3) As band) As foo;

band | count | sum | mean | stddev | min | max
-----+-----+-----+-----+-----+-----+-----
  1 | 8450000 | 725799 | 82.7064349112426 | 45.6800222638537 | 0 | 255
  2 | 8450000 | 700487 | 81.4197705325444 | 44.2161184161765 | 0 | 255
  3 | 8450000 | 575943 | 74.682739408284 | 44.2143885481407 | 0 | 255

-- &#xd14c;&#xc774;&#xbe14;&#xc758; &#xacbd;&#xc6b0;: &#xc0d8;&#xd50c;&#xb9c1;&#xc744; 100% <-
  &#xbbbf8;&#xb9cc;&#xc73c;&#xb85c; &#xc124;&#xc815;&#xd558;&#xba74; <-
  &#xc18d;&#xb3c4;&#xac00; &#xd5a5;&#xc0c1;&#xb429;&#xb2c8;&#xb2e4;.
-- &#xc774; &#xc608;&#xc2dc;&#xc5d0;&#xc11c;&#xb294; 25%&#xb85c; <-
  &#xc124;&#xc815;&#xd574;&#xc11c; &#xd6e8;&#xc52c; &#xbe68;&#xb9ac; &#xb2f5;&#xc744; <-
  &#xbc18;&#xd658;&#xbc1b;&#xc2b5;&#xb2c8;&#xb2e4;.
SELECT band, (stats).*
FROM (SELECT band, ST_SummaryStats('o_4_boston','rast', band,true,0.25) As stats
      FROM generate_series(1,3) As band) As foo;

band | count | sum | mean | stddev | min | max
-----+-----+-----+-----+-----+-----+-----
  1 | 2112500 | 180686 | 82.6890480473373 | 45.6961043857248 | 0 | 255
  2 | 2112500 | 174571 | 81.448503668639 | 44.2252623171821 | 0 | 255
  3 | 2112500 | 144364 | 74.6765884023669 | 44.2014869384578 | 0 | 255

```

## &#xcc38;&xace0;

[summarystats](#), [ST\\_SummaryStatsAgg](#), [ST\\_Count](#), [ST\\_Clip](#)

## 12.9.6 ST\_SummaryStatsAgg

`ST_SummaryStatsAgg` — &#xc885;&#xd569; &#xd568;&#xc218;&#xc785;&#xb2c8;&#xb2e4;. &#xb798;&#xc2a4;&#xd130; &#xc9d1;&#xd569;&#xc758; &#xc785;&#xb825; &#xb798;&#xc2a4;&#xd130; &#xbc34;&#xb4dc;&#xc758; count, sum, mean,

stddev, min, max, `summarystats`, `count`, `sum`, `mean`, `stddev`, `min`, `max`, `nband`, `exclude_nodata_value`, `sample_percent`, `1`.

## Synopsis

`summarystats` `ST_SummaryStatsAgg`(setof raster rast, integer nband, boolean exclude\_nodata\_value, double precision sample\_percent);

`summarystats` `ST_SummaryStatsAgg`(setof raster rast, boolean exclude\_nodata\_value, double precision sample\_percent);

`summarystats` `ST_SummaryStatsAgg`(setof raster rast, integer nband, boolean exclude\_nodata\_value);

## Parameters

`setof raster rast`: a set of rasters to be summarized.  
`integer nband`: the number of bands to be summarized.  
`boolean exclude_nodata_value`: whether to exclude nodata values from the summary statistics.  
`double precision sample_percent`: the percentage of the raster to be summarized.  
`count`, `sum`, `mean`, `stddev`, `min`, `max`, `nband`, `exclude_nodata_value`, `sample_percent`, `1`.



### Note

`nodata` value: the value to use for nodata cells. If not specified, the default value is 0. The `exclude_nodata_value` parameter can be used to exclude nodata values from the summary statistics.



### Note

`sample_percent`: the percentage of the raster to be summarized. If not specified, the default value is 1. The `sample_percent` parameter can be used to specify a different percentage of the raster to be summarized.

2.2.0 `summarystats`, `ST_SummaryStatsAgg`, `count`, `sum`, `mean`, `stddev`, `min`, `max`, `nband`, `exclude_nodata_value`, `sample_percent`, `1`.

## Example

```
WITH foo AS (
  SELECT
    rast.rast
  FROM (
    SELECT ST_SetValue (
      ST_SetValue (
        ST_SetValue (
          ST_AddBand (
            ST_MakeEmptyRaster (10, 10, 10, 10, 2, 2, 0, ←
              0,0)
            , 1, '64BF', 0, 0
          )
        , 1, 1, 1, -10
      )
    , 1, 5, 4, 0
    )
  , 1, 5, 5, 3.14159
) AS rast
```

```

) AS rast
FULL JOIN (
    SELECT generate_series(1, 10) AS id
) AS id
    ON 1 = 1
)
SELECT
    (stats).count,
    round((stats).sum::numeric, 3),
    round((stats).mean::numeric, 3),
    round((stats).stddev::numeric, 3),
    round((stats).min::numeric, 3),
    round((stats).max::numeric, 3)
FROM (
    SELECT
        ST_SummaryStatsAgg(rast, 1, TRUE, 1) AS stats
    FROM foo
) bar;

count | round | round | round | round | round
-----+-----+-----+-----+-----+-----
    20 | -68.584 | -3.429 | 6.571 | -10.000 | 3.142
(1 row)

```

**summarystats**

[summarystats](#), [ST\\_SummaryStats](#), [ST\\_Count](#), [ST\\_Clip](#)

## 12.9.7 ST\_ValueCount

**ST\_ValueCount** — Returns a table with 6 columns: count, sum, mean, stddev, min, max. The count column is the number of pixels with the searchvalue. The sum, mean, stddev, min, and max columns are the sum, mean, standard deviation, minimum, and maximum of the searchvalue. The searchvalue is a double precision value. The nband is the number of bands in the raster. The exclude\_nodata\_value is a boolean value that indicates whether to include or exclude nodata values. The searchvalues is a double precision array of search values. The roundto is a double precision value that indicates the number of decimal places to round to. The OUT value is a double precision value. The OUT count is an integer value.

### Synopsis

```

record ST_ValueCount(setof raster rast, integer nband=1, boolean exclude_nodata_value=true, double precision[] searchvalues=NULL, double precision roundto=0, double precision OUT value, integer OUT count);
record ST_ValueCount(setof raster rast, integer nband, double precision[] searchvalues, double precision roundto=0, double precision OUT value, integer OUT count);
record ST_ValueCount(setof raster rast, double precision[] searchvalues, double precision roundto=0, double precision OUT value, integer OUT count);
bigint ST_ValueCount(setof raster rast, double precision searchvalue, double precision roundto=0);
bigint ST_ValueCount(setof raster rast, integer nband, boolean exclude_nodata_value, double precision searchvalue, double precision roundto=0);
bigint ST_ValueCount(setof raster rast, integer nband, double precision searchvalue, double precision roundto=0);
setof record ST_ValueCount(text rastertable, text rastercolumn, integer nband=1, boolean exclude_nodata_value=true, double precision[] searchvalues=NULL, double precision roundto=0, double precision OUT value, integer OUT count);
setof record ST_ValueCount(text rastertable, text rastercolumn, double precision[] searchvalues, double precision roundto=0,

```

double precision OUT value, integer OUT count);  
 setof record **ST\_ValueCount**(text rastertable, text rastercolumn, integer nband, double precision[] searchvalues, double precision roundto=0, double precision OUT value, integer OUT count);  
 bigint **ST\_ValueCount**(text rastertable, text rastercolumn, integer nband, boolean exclude\_nodata\_value, double precision searchvalue, double precision roundto=0);  
 bigint **ST\_ValueCount**(text rastertable, text rastercolumn, double precision searchvalue, double precision roundto=0);  
 bigint **ST\_ValueCount**(text rastertable, text rastercolumn, integer nband, double precision searchvalue, double precision roundto=0);

### ST\_ValueCount

```
ST_ValueCount(rastertable, rastercolumn, nband, searchvalues, roundto)
ST_ValueCount(rastertable, rastercolumn, nband, exclude_nodata_value, searchvalue, roundto)
ST_ValueCount(rastertable, rastercolumn, searchvalue, roundto)
ST_ValueCount(rastertable, rastercolumn, nband, searchvalue, roundto)
```

ST\_ValueCount(rastertable, rastercolumn, nband, searchvalues, roundto) returns a set of records with the following fields:

| value | count |
|-------|-------|
| 250   | 2     |
| 251   | 1     |
| 252   | 2     |
| 253   | 6     |
| 254   | 12    |



#### Note

exclude\_nodata\_value is a boolean flag that, when set to true, excludes nodata values from the result set. The default value is false.

2.0.0 ST\_ValueCount(rastertable, rastercolumn, nband, searchvalues, roundto)

### ST\_SetBandNoDataValue

```
UPDATE dummy_rast SET rast = ST_SetBandNoDataValue(rast, 249) WHERE rid=2;
-- ST_SetBandNoDataValue(rast, 249)
-- ST_SetBandNoDataValue(rast, 249)
-- ST_SetBandNoDataValue(rast, 249)
```

```
SELECT (pvc).*
FROM (SELECT ST_ValueCount(rast) As pvc
      FROM dummy_rast WHERE rid=2) As foo
ORDER BY (pvc).value;
```

| value | count |
|-------|-------|
| 250   | 2     |
| 251   | 1     |
| 252   | 2     |
| 253   | 6     |
| 254   | 12    |

```
-- ST_SetBandNoDataValue(rast, 249)
-- ST_SetBandNoDataValue(rast, 249)
-- ST_SetBandNoDataValue(rast, 249)
SELECT (pvc).*
FROM (SELECT ST_ValueCount(rast, 1, false) As pvc
```

```
FROM dummy_rast WHERE rid=2) As foo
ORDER BY (pvc).value;
```

| value | count |
|-------|-------|
| 249   | 2     |
| 250   | 2     |
| 251   | 1     |
| 252   | 2     |
| 253   | 6     |
| 254   | 12    |

```
-- &#xc774; &#xc608;&#xc2dc;&#xb294; &#xbc34;&#xb4dc; 2 &#xac00;&#xc6b4;&#xb370; NODATA <←
&#xac12;&#xc774; &#xc544;&#xb2cc; &#xd53d;&#xc140;&#xb4e4;&#xb9cc; <←
&#xc9d1;&#xacc4;&#xd560; &#xac83;&#xc785;&#xb2c8;&#xb2e4;. --
```

```
SELECT (pvc).*
FROM (SELECT ST_ValueCount(rast,2) As pvc
FROM dummy_rast WHERE rid=2) As foo
ORDER BY (pvc).value;
```

| value | count |
|-------|-------|
| 78    | 1     |
| 79    | 1     |
| 88    | 1     |
| 89    | 1     |
| 96    | 1     |
| 97    | 1     |
| 98    | 1     |
| 99    | 2     |
| 112   | 2     |

```
:
```

```
-- &#xc2e4;&#xc81c; &#xc608;&#xc2dc;&#xc785;&#xb2c8;&#xb2e4;. &#xb3c4;&#xd615;&#xacfc; <←
&#xad50;&#xcc28;&#xd558;&#xb294; &#xd56d;&#xacf5;&#xc0ac;&#xc9c4; <←
&#xb798;&#xc2a4;&#xd130; &#xd0c0;&#xc77c;&#xc758; &#xbc34;&#xb4dc; 2&#xc5d0; <←
&#xc788;&#xb294; &#xaa8;&#xb4e0; &#xd53d;&#xc140;&#xc744; &#xc9d1;&#xacc4;&#xd55c; <←
&#xb2e4;&#xc74c;
```

```
-- &#xac1c;&#xc218;&#xac00; 500&#xac1c;&#xb97c; &#xcd08;&#xacfc;&#xd558;&#xb294; <←
&#xd53d;&#xc140; &#xbc34;&#xb4dc; &#xac12;&#xb9cc; <←
&#xbc18;&#xd658;&#xd569;&#xb2c8;&#xb2e4;.
```

```
SELECT (pvc).value, SUM((pvc).count) As total
FROM (SELECT ST_ValueCount(rast,2) As pvc
```

```
FROM o_4_boston
WHERE ST_Intersects(rast,
ST_GeomFromText('POLYGON((224486 892151,224486 892200,224706 892200,224706
892151,224486 892151))',26986)
```

```
)
) As foo
```

```
GROUP BY (pvc).value
HAVING SUM((pvc).count)
```

```
> 500
```

```
ORDER BY (pvc).value;
```

| value | total |
|-------|-------|
| 51    | 502   |
| 54    | 521   |

```
-- &#xac01; &#xb798;&#xc2a4;&#xd130;&#xc5d0;&#xc11c; &#xd2b9;&#xc815; <←
&#xb3c4;&#xd615;&#xacfc; &#xad50;&#xcc28;&#xd558;&#xb294; &#xd0c0;&#xc77c; <←
```



**ST\_MetaData**

[ST\\_MetaData](#), [ST\\_RastFromHexWKB](#), [ST\\_AsBinary/ST\\_AsWKB](#), [ST\\_AsHexWKB](#)

## 12.10.2 ST\_RastFromHexWKB

**ST\_RastFromHexWKB** — Return a raster value from a Hex representation of Well-Known Binary (WKB) raster.

### Synopsis

raster **ST\_RastFromHexWKB**(text wkb);

**ST\_RastFromHexWKB**

Given a Well-Known Binary (WKB) raster in Hex representation, return a raster.

Availability: 2.5.0

**ST\_MetaData**

```
SELECT rid, (foo.md).*
FROM (SELECT rid, ST_MetaData(rast) As md
FROM dummy_rast) As foo;
```

| rid | upperleftx | upperlefty | width | height | scalex | scaley | skewx | skewy | srid | numbands |
|-----|------------|------------|-------|--------|--------|--------|-------|-------|------|----------|
| 1   | 0          | 0          | 10    | 20     | 2      | 3      |       |       | 0    | 3        |
| 2   | 3427927.75 | 5793244    | 5     | 5      | 0.05   | -0.05  |       |       | 0    | 3        |

**ST\_MetaData**

[ST\\_MetaData](#), [ST\\_RastFromWKB](#), [ST\\_AsBinary/ST\\_AsWKB](#), [ST\\_AsHexWKB](#)

## 12.11 ST\_AsBinary/ST\_AsWKB

### 12.11.1 ST\_AsBinary/ST\_AsWKB

**ST\_AsBinary/ST\_AsWKB** — Return the Well-Known Binary (WKB) representation of the raster.

### Synopsis

bytea **ST\_AsBinary**(raster rast, boolean outasin=FALSE);

bytea **ST\_AsWKB**(raster rast, boolean outasin=FALSE);





## Using PostgreSQL Large Object Support to export raster

One way to export raster into another format is using [PostgreSQL large object export functions](#). We'll repeat the prior example but also exporting. Note for this you'll need to have super user access to db since it uses server side lo functions. It will also export to path on server network. If you need export locally, use the psql equivalent lo\_ functions which export to the local file system instead of the server file system.

```
DROP TABLE IF EXISTS tmp_out ;

CREATE TABLE tmp_out AS
SELECT lo_from_bytea(0,
    ST_AsGDALRaster(ST_Union(rast), 'JPEG', ARRAY['QUALITY=50'])
    ) AS loid
FROM dummy_rast
WHERE rast && ST_MakeEnvelope(10, 10, 11, 11);

SELECT lo_export(loid, '/tmp/dummy.jpg')
FROM tmp_out;

SELECT lo_unlink(loid)
FROM tmp_out;
```

## GeoTiff

```
SELECT ST_AsGDALRaster(rast, 'GTiff') As rastjpg
FROM dummy_rast WHERE rid=2;

-- JPEG; GeoTiff; 90%
SELECT ST_AsGDALRaster(rast, 'GTiff',
    ARRAY['COMPRESS=JPEG', 'JPEG_QUALITY=90'],
    4269) As rasttiff
FROM dummy_rast WHERE rid=2;
```

## 

Section [11.3, ST\\_GDALDrivers](#), [ST\\_SRID](#)

## 12.11.4 ST\_AsJPEG

**ST\_AsJPEG** — JPEG (Joint Photographic Exports Group)

### Synopsis

```
bytea ST_AsJPEG(raster rast, text[] options=NULL);
bytea ST_AsJPEG(raster rast, integer nband, integer quality);
bytea ST_AsJPEG(raster rast, integer nband, text[] options=NULL);
bytea ST_AsJPEG(raster rast, integer[] nbands, text[] options=NULL);
bytea ST_AsJPEG(raster rast, integer[] nbands, integer quality);
```



## 12.11.5 ST\_AsPNG

`ST_AsPNG` — PNG (Portable Network Graphics)

3, 4; RGB, RGBA

### Synopsis

```
bytea ST_AsPNG(raster rast, text[] options=NULL);
bytea ST_AsPNG(raster rast, integer nband, integer compression);
bytea ST_AsPNG(raster rast, integer nband, text[] options=NULL);
bytea ST_AsPNG(raster rast, integer[] nbands, integer compression);
bytea ST_AsPNG(raster rast, integer[] nbands, text[] options=NULL);
```

### Options

`SRID` - SRID

`compression` - compression

`options` - options

- `nband` - nband
- `nbands` - nbands
- `compression` - compression
- `options` - options

2.0.0  
GDAL 1.6.0

**&#xc608;&#xc2dc;**

```
SELECT ST_AsPNG(rast) As rastpng
FROM dummy_rast WHERE rid=2;

-- &#xc98;&#xc74c; 3&#xcac1c;&#xc758; &#xb34;&#xb4dc;&#xb97c; ↵
  &#xb0b4;&#xbcf4;&#xb0b4;&#xc11c; &#xb34;&#xb4dc; 3&#xc744; &#xbe68;&#xcac15;,, ↵
  &#xb34;&#xb4dc; 1&#xc744; &#xb179;&#xc0c9;, &#xb34;&#xb4dc; 2&#xb97c; ↵
  &#xd30c;&#xb791;&#xc73c;&#xb85c; &#xb9e4;&#xd551;
SELECT ST_AsPNG(rast, ARRAY[3,1,2]) As rastpng
FROM dummy_rast WHERE rid=2;
```

**&#xc38;&#xc60;**

[ST\\_AsGDALRaster](#), [ST\\_ColorMap](#), [ST\\_GDALDrivers](#), Section 11.3

**12.11.6 ST\_AsTIFF**

**ST\_AsTIFF** — Return the raster selected bands as a single TIFF image (byte array). If no band is specified or any of specified bands does not exist in the raster, then will try to use all bands.

**Synopsis**

```
bytea ST_AsTIFF(raster rast, text[] options='', integer srid=sameassource);
bytea ST_AsTIFF(raster rast, text compression='', integer srid=sameassource);
bytea ST_AsTIFF(raster rast, integer[] nbands, text compression='', integer srid=sameassource);
bytea ST_AsTIFF(raster rast, integer[] nbands, text[] options, integer srid=sameassource);
```

**&#xc124;&#xba85;**

&#xb798;&#xc2a4;&#xd130;&#xc5d0;&#xc11c; &#xc120;&#xd0dd;&#xd55c; &#xb34;&#xb4dc;&#xb4e4;&#xc744; &#xb2e8;&#xc2dc; &#xc774;&#xbbf8;&#xc9c0;&#xb85c; &#xb18;&#xd658;&#xd569;&#xb2c8;&#xb2e4;. &#xb34;&#xb530;&#xb85c; &#xc124;&#xc815;&#xd558;&#xc9c0; &#xc54a;&#xc744; &#xcabd;&#xc6b0;, &#xaa8;&#xb4e0; &#xb34;&#xc4f0;&#xb824; &#xd560; &#xcac83;&#xc785;&#xb2c8;&#xb2e4;. &#xc774; &#xd568;&#xc218;&#xb294; [ST\\_AsGDALRaster](#) &#xb97c; &#xb458;&#xb7ec;&#xc2fc; &#xb798;&#xd37c;&#xc785;&#xb2c8;&#xb2e4;. &#xb35c; &#xd754;&#xd55c; &#xb798;&#xc720;&#xd615;&#xc73c;&#xb85c; &#xb0b4;&#xbcf4;&#xb0b4;&#xc57c; &#xd560; &#xcabd;&#xc6b0; [ST\\_AsGDALRaster](#) &#xd568;&#xc218;&#xb97c; &#xc774;&#xc6a9;&#xd558;&#xc2ed;&#xc2dc;&#xc624;. &#xc5b4;&#xb5a4; &#xcacf5;&#xcac04; &#xc38;&#xc870; SRS &#xd14d;&#xc2a4;&#xd2b8;&#xb3c4; &#xc5c6;&#xc744; &#xcabd;&#xc6b0;, &#xb798;&#xc2a4;&#xd130 &#xcacf5;&#xcac04; &#xc38;&#xc870;&#xb97c; &#xc0ac;&#xc6a9;&#xd569;&#xb2c8;&#xb2e4;. &#xc774; &#xd568;&#xc218;&#xb2e4;&#xc74c;&#xcacf; &#xcac19;&#xc740; &#xb9ce;&#xc740; &#xc635;&#xc158;&#xc744; &#xcac00;&#xc9c4; &#xb9ce;&#xbcc0;&#xc885;&#xc774; &#xc788;&#xc2b5;&#xb2c8;&#xb2e4;:

- nbands - &#xb0b4;&#xbcf4;&#xb0bc; &#xb34;&#xb4dc;&#xb4e4;&#xc758; &#xb30;&#xc5f4;&#xc785;&#xb2c8;&#xb2e4;(P &#xcabd;&#xc6b0; 3&#xc774; &#xcd5c;&#xb300;&#xcac12;&#xc785;&#xb2c8;&#xb2e4;). &#xb34;&#xb4dc;&#xc758; &#xc21c;&#xc11c;&#xb294; RGB&#xc785;&#xb2c8;&#xb2e4;. &#xc608;&#xb97c; &#xb4e4;&#xc5b4; ARRAY[3,2,1]&#xc740; &#xb34;&#xb4dc; 3&#xc744; &#xbe68;&#xcac04;&#xc0c9;. &#xb34;&#xb4dc; 2&#xb97c; &#xcd08;&#xb85d;&#xc0c9;. &#xb34;&#xb4dc; 1&#xc744; &#xd30c;&#xb780;&#xc0c9;&#xc5d0; &#xb9e4;&#xd551;&#xd55c;&#xb2e4;&#xb294; &#xb73b
- compression - &#xc555;&#xcd95; &#xd45c;&#xd604;&#xc2dd;. JPEG90(&#xb610;&#xb294; &#xb2e4;&#xb978; &#xd37c;&#xc2dc; LZW, JPEG, DEFLATE9
- options - GTiff&#xc5d0; &#xb300;&#xd574; &#xc815;&#xc758;&#xb41c; GDAL &#xc0dd;&#xc131; &#xc635;&#xc158;&#xb &#xd14d;&#xc2a4;&#xd2b8; &#xb30;&#xc5f4;&#xc785;&#xb2c8;&#xb2e4;( [ST\\_GDALDrivers](#)&#xc5d0;&#xc11c; GTiff&#xc5c &#xb300;&#xd55c; create\_options&#xb97c; &#xc0b4;&#xd3b4;&#xbcf4;&#xc2ed;&#xc2dc;&#xc624;). &#xc790;&#xc138;&#xd &#xb0b4;&#xc6a9;&#xc740; [GDAL](#) &#xb798;&#xc2a4;&#xd130; &#xd615;&#xc2dd; &#xc635;&#xc158; &#xc744; &#xc38;&#xc

- `srid` - SRID

GDAL 1.6.0

### JPEG, 90%

```
SELECT ST_AsTIFF(rast, 'JPEG90') As rasttiff
FROM dummy_rast WHERE rid=2;
```

[ST\\_GDALDrivers](#), [ST\\_AsGDALRaster](#), [ST\\_SRID](#)

## 12.12 `ST_Clip`

### 12.12.1 `ST_Clip`

`ST_Clip` — Clip a raster to a geometry. The output raster will have the same coordinate system as the input raster. The output raster will have the same resolution as the input raster. The output raster will have the same number of bands as the input raster. The output raster will have the same data type as the input raster. The output raster will have the same nodata value as the input raster. The output raster will have the same extent as the input raster.

#### Synopsis

```
raster ST_Clip(raster rast, integer[] nband, geometry geom, double precision[] nodataval=NULL, boolean crop=TRUE);
raster ST_Clip(raster rast, integer nband, geometry geom, double precision nodataval, boolean crop=TRUE);
raster ST_Clip(raster rast, integer nband, geometry geom, boolean crop);
raster ST_Clip(raster rast, geometry geom, double precision[] nodataval=NULL, boolean crop=TRUE);
raster ST_Clip(raster rast, geometry geom, double precision nodataval, boolean crop=TRUE);
raster ST_Clip(raster rast, geometry geom, boolean crop);
```

#### `ST_Clip`

`ST_Clip` (raster rast, integer[] nband, geometry geom, double precision[] nodataval=NULL, boolean crop=TRUE);

`ST_Clip` (raster rast, integer nband, geometry geom, double precision nodataval, boolean crop=TRUE);

`ST_Clip` (raster rast, integer nband, geometry geom, boolean crop);

`ST_Clip` (raster rast, geometry geom, double precision[] nodataval=NULL, boolean crop=TRUE);

`ST_Clip` (raster rast, geometry geom, double precision nodataval, boolean crop=TRUE);

`ST_Clip` (raster rast, geometry geom, boolean crop);

```
&#xd568;&#xc218;&#xb294; &#xac01; &#xbc34;&#xb4dc;&#xc5d0; &#xd560;&#xb2f9;&#xb420; &#xb2e8;&#xc77c; &#xac12;&#xc785;&#xb825;&#xbc1b;&#xc2b5;&#xb2c8;&#xb2e4;.
```

```
crop &#xc744; &#xc124;&#xc815;&#xd558;&#xc9c0; &#xc54a;&#xc744; &#xacbd;&#xc6b0;., &#xcc38;&#xc73c;&#xb85c;
&#xac00;&#xc815;&#xd569;&#xb2c8;&#xb2e4;. geom &#xbc94;&#xc704;&#xc640; rast &#xbc94;&#xc704;&#xac00;
&#xad50;&#xcc28;&#xd558;&#xb294; &#xbd80;&#xbd84;&#xc744; &#xc798;&#xb77c;&#xb0b8; &#xb798;&#xc2a4;&#xd130;&#xc9c0;
&#xcd9c;&#xb825;&#xd55c;&#xb2e4;&#xb294; &#xc758;&#xbbf8;&#xc785;&#xb2c8;&#xb2e4;. crop &#xc744; &#xac70;&#xc9c0;
&#xc124;&#xc815;&#xd560; &#xacbd;&#xc6b0;., &#xc0c8; &#xb798;&#xc2a4;&#xd130;&#xc758; &#xbc94;&#xc704;&#xb294;
rast &#xc758; &#xbc94;&#xc704;&#xc640; &#xb3d9;&#xc77c;&#xd569;&#xb2c8;&#xb2e4;.
```

```
2.0.0 &#xbc84;&#xc804;&#xbd80;&#xd130; &#xc0ac;&#xc6a9;&#xd560; &#xc218; &#xc788;&#xc2b5;&#xb2c8;&#xb2e4;.
```

```
&#xac1c;&#xc120; &#xc0ac;&#xd56d;. 2.1.0 &#xbc84;&#xc804;&#xc5d0;&#xc11c; C &#xc5b8;&#xc5b4;&#xb85c; &#xb2e4;&#xc4f0;&#xc600;&#xc2b5;&#xb2c8;&#xb2e4;.
```

```
&#xc774; &#xc608;&#xc2dc;&#xb294; MassGIS &#xc0ac;&#xc774;&#xd2b8;&#xc758; MassGIS Aerial Orthos &#xc5d0;&#xc11c;
&#xb2e4;&#xc6b4;&#xb85c;&#xb4dc;&#xd560; &#xc218; &#xc788;&#xb294; &#xb9e4;&#xc0ac;&#xcd94;&#xc138;&#xce20;
&#xd56d;&#xacf5;&#xc0ac;&#xc9c4; &#xb370;&#xc774;&#xd130;&#xb97c; &#xc774;&#xc6a9;&#xd569;&#xb2c8;&#xb2e4;.
&#xb9e4;&#xc0ac;&#xcd94;&#xc138;&#xce20; &#xc8fc; &#xbbf8;&#xd130; &#xb2e8;&#xc704; &#xd3c9;&#xba74;&#xc758;
&#xc88c;&#xd45c;&#xb85c; &#xb3fc; &#xc788;&#xc2b5;&#xb2c8;&#xb2e4;.
```

### **&#xc608;&#xc2dc;: &#xbc34;&#xb4dc; 1&#xac1c; &#xc798;&#xb77c;&#xb0b4;&#xae30;**

```
-- &#xd56d;&#xacf5;&#xc0ac;&#xc9c4; &#xd0c0;&#xc77c;&#xc758; &#xccab; &#xbc88;&#xc9f8; ←
&#xbc34;&#xb4dc;&#xb97c; 20&#xbbf8;&#xd130; &#xbc84;&#xd37c;&#xb85c; ←
&#xc798;&#xb77c;&#xb0b4;&#xae30;
SELECT ST_Clip(rast, 1,
              ST_Buffer(ST_Centroid(ST_Envelope(rast)), 20)
              ) from aerials.boston
WHERE rid = 4;
```

```
-- &#xb798;&#xc2a4;&#xd130;&#xc758; &#xb9c8;&#xc9c0;&#xb9c9; &#xcc28;&#xc6d0;&#xc5d0; ←
&#xb300;&#xd55c; &#xc798;&#xb77c;&#xb0b4;&#xae30;&#xc758; &#xc601;&#xd5a5;&#xc744; ←
&#xbcf4;&#xc5ec;&#xc90d;&#xb2c8;&#xb2e4;.
-- &#xb9c8;&#xc9c0;&#xb9c9; &#xbc94;&#xc704;&#xb97c; &#xc5b4;&#xb5bb;&#xac8c; ←
&#xb3c4;&#xd615;&#xc758; &#xbc94;&#xc704;&#xb85c; ←
&#xc798;&#xb77c;&#xb0b4;&#xb294;&#xc9c0; &#xb208;&#xc5ec;&#xaca8; ←
&#xbcf4;&#xc2ed;&#xc2dc;&#xc624;.
-- crop = true &#xc77c; &#xacbd;&#xc6b0;
SELECT ST_XMax(ST_Envelope(ST_Clip(rast, 1, clipper, true))) As xmax_w_trim,
       ST_XMax(clipper) As xmax_clipper,
       ST_XMax(ST_Envelope(ST_Clip(rast, 1, clipper, false))) As xmax_wo_trim,
       ST_XMax(ST_Envelope(rast)) As xmax_rast_orig
FROM (SELECT rast, ST_Buffer(ST_Centroid(ST_Envelope(rast)), 6) As clipper
      FROM aerials.boston
WHERE rid = 6) As foo;
```

| xmax_w_trim      | xmax_clipper     | xmax_wo_trim     | xmax_rast_orig   |
|------------------|------------------|------------------|------------------|
| 230657.436173996 | 230657.436173996 | 230666.436173996 | 230666.436173996 |



&#xc798;&#xb77c;&#xb0b4;&#xae30;  
 &#xc804;&#xc758;&#xc804;&#xccb4;  
 &#xb798;&#xc2a4;&#xd130; &#xd0c0;&#xc77c;



&#xc798;&#xb77c;&#xb0b8; &#xd6c4;

**&#xc608;&#xc2dc;: crop &#xc5c6;&#xc774; &#xbc34;&#xb4dc; 1&#xac1c;&#xb97c; &#xc798;&#xb77c;&#xb0b8; &#xb2e4;&#xc**  
**&#xb2e4;&#xb978; &#xbc34;&#xb4dc;&#xb4e4;&#xc744; &#xbcc0;&#xacbd; &#xc5c6;&#xc774; &#xb2e4;&#xc2dc; &#xcd94;&**

```

-- &#xc774;&#xc804;&#xacfc; &#xb3d9;&#xc77c;&#xd55c; &#xc608;&#xc2dc;&#xc9c0;&#xb9cc;, ←
  ST_AddBand&#xb97c; &#xc0ac;&#xc6a9;&#xd560; &#xc218; &#xc788;&#xc73c;&#xb824;&#xba74; ←
  crop&#xc744; &#xac70;&#xc9d3;&#xc73c;&#xb85c; &#xc124;&#xc815;&#xd574;&#xc57c; ←
  &#xd569;&#xb2c8;&#xb2e4; .
-- ST_AddBand &#xd568;&#xc218;&#xb97c; &#xc4f0;&#xb824;&#xba74; &#xaa8;&#xb4e0; ←
  &#xbc34;&#xb4dc;&#xc758; &#xb108;&#xbe44; &#xbc0f; &#xb192;&#xc774;&#xac00; ←
  &#xb3d9;&#xc77c;&#xd574;&#xc57c; &#xd558;&#xae30; ←
  &#xb54c;&#xbb38;&#xc785;&#xb2c8;&#xb2e4; .
SELECT ST_AddBand(ST_Clip(rast, 1,
                        ST_Buffer(ST_Centroid(ST_Envelope(rast)),20),false
                        ), ARRAY[ST_Band(rast,2),ST_Band(rast,3)] ) from aerials.boston
WHERE rid = 6;
    
```



&#xc798;&#xb77c;&#xb0b4;&#xae30;  
 &#xc804;&#xc758; &#xc804;&#xccb4;  
 &#xb798;&#xc2a4;&#xd130; &#xd0c0;&#xc77c;



&#xc798;&#xb77c;&#xb0b8; &#xd6c4; -  
 &#xbe44;&#xd604;&#xc2e4;&#xc801;

**&#xc608;&#xc2dc;: &#xaa8;&#xb4e0; &#xc34;&#xb4dc; &#xc798;&#xb77c;&#xb0b4;&#xae30;**

```

-- &#xd56d;&#xacf5;&#xc0ac;&#xc9c4; &#xd0c0;&#xc77c;&#xc758; &#xaa8;&#xb4e0;  ←
  &#xc34;&#xb4dc;&#xb97c; 20&#xbbf8;&#xd130; &#xc84;&#xd37c;&#xb85c;  ←
  &#xc798;&#xb77c;&#xb0b4;&#xae30;
-- &#xcc28;&#xc774;&#xc810;&#xc774;&#xb77c;&#xba74; &#xc798;&#xb77c;&#xb0bc;  ←
  &#xd2b9;&#xc815; &#xc34;&#xb4dc;&#xb97c; &#xc9c0;&#xc815;&#xd558;&#xc9c0;  ←
  &#xc54a;&#xb294;&#xb2e4;&#xb294; &#xc810;&#bfd0;&#xc785;&#xb2c8;&#xb2e4; .
-- &#xb530;&#xb77c;&#xc11c; &#xaa8;&#xb4e0; &#xc34;&#xb4dc;&#xb97c;  ←
  &#xc798;&#xb77c;&#xb0c5;&#xb2c8;&#xb2e4; .
SELECT ST_Clip(rast,
               ST_Buffer(ST_Centroid(ST_Envelope(rast)), 20),
               false
            ) from aerials.boston
WHERE rid = 4;
    
```



- `grayscale` - 8BUI (grayscale - 8BUI of gray)
- `pseudocolor` - 8BUI(4 channels)
- `fire` - 8BUI(4 channels)
- `bluered` - 8BUI(4 channels)

```

colormap (5 0 0 0 255; 4 100:50 55 255; 1 150,100 150 255; 0% 255 255 255 255; nv 0 0 0 0)
gdaldem (method INTERPOLATE; EXACT; NEAREST;

```

```

5 0 0 0 255
4 100:50 55 255
1 150,100 150 255
0% 255 255 255 255
nv 0 0 0 0

```

```

colormap (GDAL; gdaldem);

```

- `INTERPOLATE` -
- `EXACT` -
- `NEAREST` -



**Note**

ColorBrewer

**Warning**

NODATA &#x12;&#x744; &#xac00;&#xc9c0;&#xc9c0; &#xc54a;&#xc744; &#xac83;&#xc785;&#xb2c8;&#xb2e4;. NODATA &#xac12;&#xc774; &#xd544;&#xc694;&#xd558;&#xb2e4;&#xba74; **ST\_SetBandNoDataValue** &#xb97c; &#xc774;&#xc6a9;&#xd574;&#xc11c; NODATA &#xac12;&#xc744; &#xc124;&#xc815;&#xd558;&#xc2ed;&#xc2dc;&#xc624;.

2.1.0 &#xbc84;&#xc804;&#xbd80;&#xd130; &#xc0ac;&#xc6a9;&#xd560; &#xc218; &#xc788;&#xc2b5;&#xb2c8;&#xb2e4;.

**&#xc608;&#xc2dc;**

&#xb2e4;&#xc74c;&#xc740; &#xc608;&#xc2dc;&#xc6a9; &#xac00;&#xc9dc; &#xd14c;&#xc774;&#xbe14;&#xc785;&#xb2c8;&#xb2e4;.

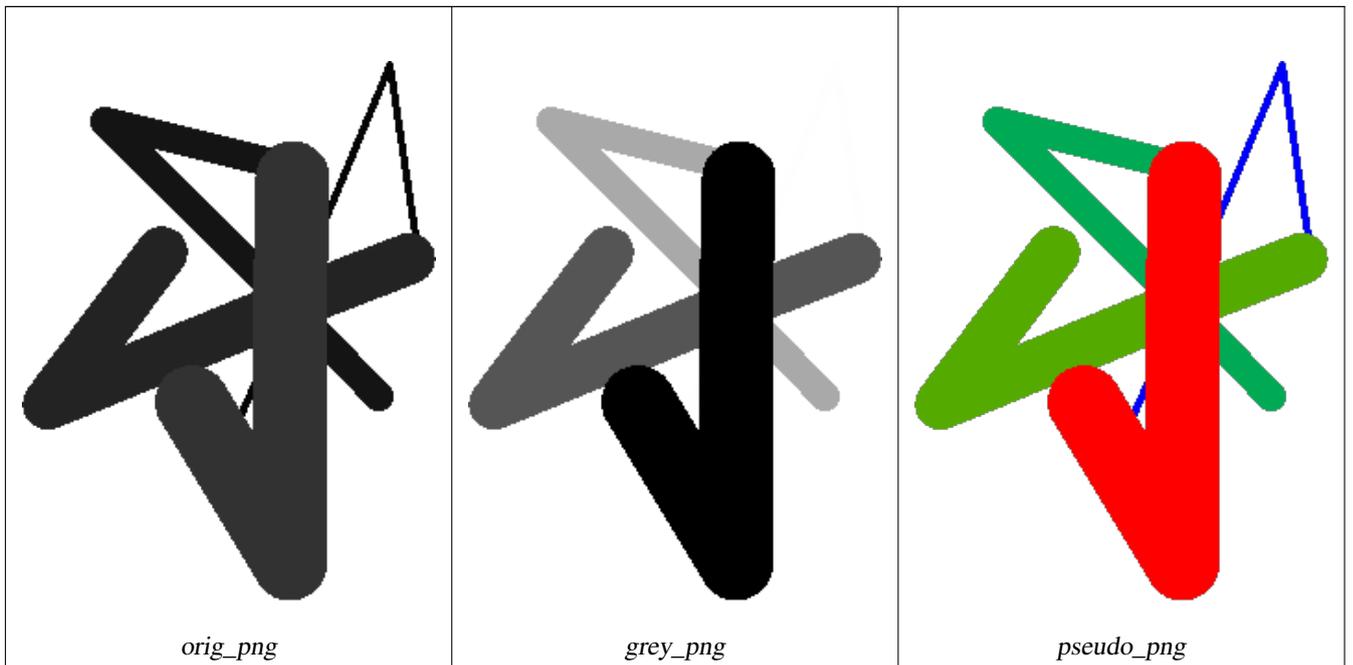
```
-- &#xd14c;&#xc2a4;&#xd2b8; &#xb798;&#xc2a4;&#xd130; &#xd14c;&#xc774;&#xbe14; ↔
&#xc791;&#xc131; --
DROP TABLE IF EXISTS funky_shapes;
CREATE TABLE funky_shapes(rast raster);

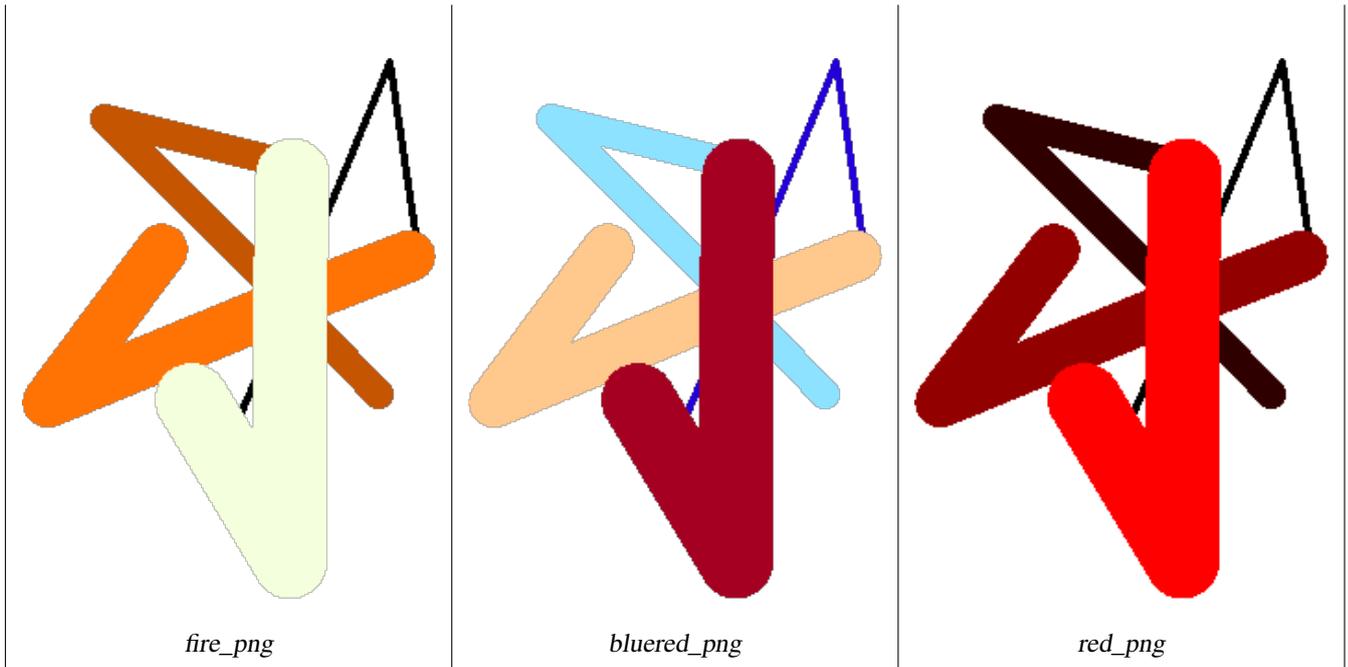
INSERT INTO funky_shapes (rast)
WITH ref AS (
    SELECT ST_MakeEmptyRaster( 200, 200, 0, 200, 1, -1, 0, 0) AS rast
)
SELECT
    ST_Union(rast)
FROM (
    SELECT
        ST_AsRaster(
            ST_Rotate(
                ST_Buffer(
                    ST_GeomFromText('LINESTRING(0 2,50 50,150 150,125 ↔
                    50)'),
                    i*2
                ),
                pi() * i * 0.125, ST_Point(50,50)
            ),
            ref.rast, '8BUI'::text, i * 5
        ) AS rast
    FROM ref
    CROSS JOIN generate_series(1, 10, 3) AS i
) AS shapes;
```

```
SELECT
    ST_NumBands(rast) As n_orig,
    ST_NumBands(ST_ColorMap(rast,1, 'greyscale')) As ngrey,
    ST_NumBands(ST_ColorMap(rast,1, 'pseudocolor')) As npseudo,
    ST_NumBands(ST_ColorMap(rast,1, 'fire')) As nfire,
    ST_NumBands(ST_ColorMap(rast,1, 'bluered')) As nbluered,
    ST_NumBands(ST_ColorMap(rast,1, '
100% 255 0 0
80% 160 0 0
50% 130 0 0
30% 30 0 0
20% 60 0 0
0% 0 0 0
nv 255 255 255
')) As nred
FROM funky_shapes;
```

```
n_orig | ngrey | npseudo | nfire | nbluered | nred
-----+-----+-----+-----+-----+-----
      1 |    1 |    4 |    4 |    4 |    3
```

```
ST_AsPNG(
  ST_ColorMap(rast,1,'greyscale')) As grey_png,
  ST_AsPNG(ST_ColorMap(rast,1,'pseudocolor')) As pseudo_png,
  ST_AsPNG(ST_ColorMap(rast,1,'nfire')) As fire_png,
  ST_AsPNG(ST_ColorMap(rast,1,'bluered')) As bluered_png,
  ST_AsPNG(ST_ColorMap(rast,1,'
100% 255 0 0
80% 160 0 0
50% 130 0 0
30% 30 0 0
20% 60 0 0
0% 0 0 0
nv 255 255 255
')) As red_png
FROM funky_shapes;
```





**ST\_Grayscale**

[ST\\_AsPNG](#), [ST\\_AsRaster](#), [ST\\_Band](#), [ST\\_BandIndex](#), [ST\\_BandName](#), [ST\\_BandNumber](#), [ST\\_BandNumberFromName](#), [ST\\_BandNumberToName](#), [ST\\_BandNames](#), [ST\\_BandNamesFromRaster](#), [ST\\_BandNamesToRaster](#), [ST\\_BandNumberFromName](#), [ST\\_BandNumberToName](#), [ST\\_BandNames](#), [ST\\_BandNamesFromRaster](#), [ST\\_BandNamesToRaster](#), [ST\\_Grayscale](#), [ST\\_NumBands](#), [ST\\_Reclass](#), [ST\\_SetBandNoDataValue](#), [ST\\_Union](#)

### 12.12.3 ST\_Grayscale

**ST\_Grayscale** — Creates a new one-8BUI band raster from the source raster and specified bands representing Red, Green and Blue

#### Synopsis

- (1) raster **ST\_Grayscale**(raster rast, integer redband=1, integer greenband=2, integer blueband=3, text extntype=INTERSECTION);
- (2) raster **ST\_Grayscale**(rastbandarg[] rastbandargset, text extntype=INTERSECTION);

**ST\_Grayscale**

Create a raster with one 8BUI band given three input bands (from one or more rasters). Any input band whose pixel type is not 8BUI will be reclassified using [ST\\_Reclass](#).



#### Note

This function is not like [ST\\_ColorMap](#) with the `grayscale` keyword as [ST\\_ColorMap](#) operates on only one band while this function expects three bands for RGB. This function applies the following equation for converting RGB to Grayscale:  $0.2989 * RED + 0.5870 * GREEN + 0.1140 * BLUE$

Availability: 2.5.0

**Example 1**

```
SET postgis.gdal_enabled_drivers = 'ENABLE_ALL';
SET postgis.enable_outdb_rasters = True;

WITH apple AS (
  SELECT ST_AddBand(
    ST_MakeEmptyRaster(350, 246, 0, 0, 1, -1, 0, 0, 0),
    '/tmp/apple.png'::text,
    NULL::int[]
  ) AS rast
)
SELECT
  ST_AsPNG(rast) AS original_png,
  ST_AsPNG(ST_Grayscale(rast)) AS grayscale_png
FROM apple;
```

*original\_png**grayscale\_png***Example 2**

```
SET postgis.gdal_enabled_drivers = 'ENABLE_ALL';
SET postgis.enable_outdb_rasters = True;

WITH apple AS (
  SELECT ST_AddBand(
    ST_MakeEmptyRaster(350, 246, 0, 0, 1, -1, 0, 0, 0),
    '/tmp/apple.png'::text,
    NULL::int[]
  ) AS rast
)
SELECT
  ST_AsPNG(rast) AS original_png,
  ST_AsPNG(ST_Grayscale(
    ARRAY[
      ROW(rast, 1)::rastbandarg, -- red
      ROW(rast, 2)::rastbandarg, -- green
      ROW(rast, 3)::rastbandarg, -- blue
    ]::rastbandarg[]
  )) AS grayscale_png
FROM apple;
```

**ST\_AsPNG, ST\_Reclass, ST\_ColorMap**

**ST\_AsPNG, ST\_Reclass, ST\_ColorMap**

## 12.12.4 ST\_Intersection

ST\_Intersection — Returns the intersection of two geometries. The result is a geometry of the same type as the inputs. If either input is NULL, the result is NULL. If the inputs are of different SRIDs, the result will have the SRID of the first input. If the inputs are of the same SRID, the result will have the SRID of the inputs. If the inputs are of different SRIDs and the result is NULL, the SRID of the result will be the SRID of the first input.

### Synopsis

```
setof geomval ST_Intersection(geometry geom, raster rast, integer band_num=1);
setof geomval ST_Intersection(raster rast, geometry geom);
setof geomval ST_Intersection(raster rast, integer band, geometry geom);
raster ST_Intersection(raster rast1, raster rast2, double precision[] nodataval);
raster ST_Intersection(raster rast1, raster rast2, text returnband, double precision[] nodataval);
raster ST_Intersection(raster rast1, integer band1, raster rast2, integer band2, double precision[] nodataval);
raster ST_Intersection(raster rast1, integer band1, raster rast2, integer band2, text returnband, double precision[] nodataval);
```

**ST\_Intersection**

ST\_Intersection(geometry geom1, geometry geom2) Returns the intersection of two geometries. The result is a geometry of the same type as the inputs. If either input is NULL, the result is NULL. If the inputs are of different SRIDs, the result will have the SRID of the first input. If the inputs are of the same SRID, the result will have the SRID of the inputs. If the inputs are of different SRIDs and the result is NULL, the SRID of the result will be the SRID of the first input.

```
geomval ST_Intersection(geometry geom1, geometry geom2, integer band_num=1);
geomval ST_Intersection(raster rast1, raster rast2, integer band_num=1);
geomval ST_Intersection(raster rast1, integer band1, raster rast2, integer band2, double precision[] nodataval);
geomval ST_Intersection(raster rast1, integer band1, raster rast2, integer band2, text returnband, double precision[] nodataval);
ST_Intersection(geometry geom1, geometry geom2) PostGIS 2.1.0; ST_Intersection(geometry geom1, geometry geom2, integer band_num=1) PostGIS 2.1.0; ST_Intersection(raster rast1, raster rast2, double precision[] nodataval) PostGIS 2.1.0; ST_Intersection(raster rast1, raster rast2, text returnband, double precision[] nodataval) PostGIS 2.1.0; ST_Intersection(raster rast1, integer band1, raster rast2, integer band2, double precision[] nodataval) PostGIS 2.1.0; ST_Intersection(raster rast1, integer band1, raster rast2, integer band2, text returnband, double precision[] nodataval) PostGIS 2.1.0;
```

```
ST_Intersection(geometry geom1, geometry geom2) PostGIS 2.1.0; ST_Intersection(geometry geom1, geometry geom2, integer band_num=1) PostGIS 2.1.0; ST_Intersection(raster rast1, raster rast2, double precision[] nodataval) PostGIS 2.1.0; ST_Intersection(raster rast1, raster rast2, text returnband, double precision[] nodataval) PostGIS 2.1.0; ST_Intersection(raster rast1, integer band1, raster rast2, integer band2, double precision[] nodataval) PostGIS 2.1.0; ST_Intersection(raster rast1, integer band1, raster rast2, integer band2, text returnband, double precision[] nodataval) PostGIS 2.1.0;
```

```
ST_Intersection(geometry geom1, geometry geom2) PostGIS 2.1.0; ST_Intersection(geometry geom1, geometry geom2, integer band_num=1) PostGIS 2.1.0; ST_Intersection(raster rast1, raster rast2, double precision[] nodataval) PostGIS 2.1.0; ST_Intersection(raster rast1, raster rast2, text returnband, double precision[] nodataval) PostGIS 2.1.0; ST_Intersection(raster rast1, integer band1, raster rast2, integer band2, double precision[] nodataval) PostGIS 2.1.0; ST_Intersection(raster rast1, integer band1, raster rast2, integer band2, text returnband, double precision[] nodataval) PostGIS 2.1.0;
```

```
ST_Intersection(geometry geom1, geometry geom2) PostGIS 2.1.0; ST_Intersection(geometry geom1, geometry geom2, integer band_num=1) PostGIS 2.1.0; ST_Intersection(raster rast1, raster rast2, double precision[] nodataval) PostGIS 2.1.0; ST_Intersection(raster rast1, raster rast2, text returnband, double precision[] nodataval) PostGIS 2.1.0; ST_Intersection(raster rast1, integer band1, raster rast2, integer band2, double precision[] nodataval) PostGIS 2.1.0; ST_Intersection(raster rast1, integer band1, raster rast2, integer band2, text returnband, double precision[] nodataval) PostGIS 2.1.0;
```

&#xcd9c;&#xb825;&#xbb3c;&#xc758; &#xbaa8;&#xb4e0; &#xbc34;&#xb4dc;&#xc5d0;&#xc11c; NODATA &#xac12;&#xc744; &#xac00;&#xc9c0;&#xac8c; &#xb429;&#xb2c8;&#xb2e4;. &#xc989;. NODATA &#xac12;&#xc744; &#xac00;&#xc9c4; &#xd53d;&#xad50;&#xcc28;&#xd558;&#xb294; &#xd53d;&#xc140;&#xc744; &#xbaa8;&#xb450; NODATA &#xac12;&#xc744; &#xac00;&#xc9c4; &#xd53d;&#xc140;&#xb85c; &#xcd9c;&#xb825;&#xd569;&#xb2c8;&#xb2e4;.

ST\_Intersection &#xd568;&#xc218;&#xac00; &#xcd9c;&#xb825;&#xd558;&#xb294; &#xb798;&#xc2a4;&#xd130;&#xb294; &#xad50;&#xcc28;&#xd558;&#xc9c0; &#xc54a;&#xb294; &#xbd80;&#xbd84;&#xc5d0; &#xd560;&#xb2f9;&#xb41c; NODATA &#xac12;&#xc744; &#xac00;&#xc9c0;&#xace0; &#xc788;&#xc5b4;&#xc57c; &#xd569;&#xb2c8;&#xb2e4;. &#xc0ac;&#xc6a9;&#xc694;&#xccad;&#xd558;&#xb290;&#xb0d0;&#xc5d0; &#xb530;&#xb77c; 1&#xac1c; &#xb610;&#xb294; 2&#xac1c;&#xc758; NODATA &#xac12;&#xc744; &#xac00;&#xc9c4; nodataval[] &#xbc30;&#xc5f4;&#xc744; &#xc785;&#xb825;&#xd574;&#xc140;&#xc5b4;&#xb5a4; &#xcd9c;&#xb825; &#xbc34;&#xb4dc;&#xc5d0; &#xb300;&#xd574;&#xc11c;&#xb3c4; NODATA &#xac12;&#xc744; &#xc815;&#xc758;&#xd558;&#xac70;&#xb098; &#xb300;&#xccb4;&#xd560; &#xc218; &#xc788;&#xc2b5;&#xb2c8;&#xb2e4;. &#xbc30;&#xc5f4;&#xc758; &#xccab; &#bc88;&#xc9f8; &#xac12;&#xc740; &#xccab; &#bc88;&#xc9f8; &#bc34;&#xb4dc;&#xc5d0; NODATA &#xac12;&#xc744; &#xb300;&#xccb4;&#xd558;&#xace0;. &#xb450; &#bc88;&#xc9f8; &#xac12;&#xc740; &#xb450; &#bc88;&#xc9f8; &#bc34;&#xb4dc;&#xc758; NODATA &#xac12;&#xc744; &#xb300;&#xccb4;&#xd569;&#xb2c8;&#xb2e4;. &#xb9cc;&#xc57d;&#xc785;&#xb825; &#bc34;&#xb4dc; &#xac00;&#xc6b4;&#xb370; &#xd558;&#xb098;&#xc5d0; &#xc815;&#xc744; NODATA &#xac12;&#xc774; &#xc5c6;&#xace0; &#xc785;&#xb825;&#xb41c; NODATA &#xac12; &#bc30;&#xc5f4;&#xb3c4; &#xc5c6;&#xc744; &#xacbd;&#xc6b0;. ST\_MinPossibleValue &#xd568;&#xc218;&#xb97c; &#xd1b5;&#xd574; NODATA &#xac12;&#xc744; &#xc120;&#xd0dd;&#xd569;&#xb2c8;&#xb2e4;. NODATA &#xac12;&#xc758; &#bc30;&#xc5f4;&#xc744; &#xc785;&#xb825;&#bc1b;&#xb294; &#baa8;&#xb4e0; &#bcc0;&#xc885; &#xd568;&#xc218;&#xb294; &#xc694;&#xccad;&#xc0ac;&#xc6a9;&#xc01; &#bc34;&#xb4dc;&#xc5d0; &#xd560;&#xb2f9;&#xb420; &#xb2e8;&#xc77c; &#xac12;&#xb3c4; &#xc785;&#xb825;&#xbaa8;&#xb4e0; &#bcc0;&#xc885; &#xd568;&#xc218;&#xc5d0;&#xc11c;. &#bc34;&#xb4dc; &#bc88;&#xd638;&#xb97c; &#b530;&#xb85c; &#xc9c0;&#xc815;&#xd558;&#xc9c0; &#xc54a;&#xc744; &#xacbd;&#xc6b0; &#bc34;&#xb4dc; 1&#xb85c; &#xac00;&#xc815;&#xd569;&#xb2c8;&#xb2e4;. &#xb798;&#xc2a4;&#xd130;&#xc640; &#xb3c4;&#xd615; &#xc0ac;&#xc774;&#xad50;&#xcc28; &#xbd80;&#xbd84;&#xc744; &#xb798;&#xc2a4;&#xd130;&#xb85c; &#bc18;&#xd658;&#xbc1b;&#xc544;&#xd560; &#xacbd;&#xc6b0;. **ST\_Clip** &#xc744; &#xcc38;&#xc870;&#xd558;&#xc2ed;&#xc2dc;&#xc624;.

**Note**  
 &#xcd9c;&#xb825; &#xbc94;&#xc704; &#xb610;&#xb294; NODATA &#xac12;&#xc744; &#xb9de;&#xb2e5;&#xb728;&#xb838;&#xc744; &#xb54c; &#xb34;&#xc5c7;&#xc744; &#bc18;&#xd658;&#xd560;&#xc9c0;&#xb97c; &#xb354; &#xc138;&#xbc00;&#xd788; &#xc870;&#xc815;&#xd558;&#xb824;&#xba74;. **ST\_MapAlgebraExpr** &#xd568;&#xc218;&#xc758; &#xb450; &#xb798;&#xc2a4;&#xd130;&#xb97c; &#xc785;&#xb825;&#bc1b;&#xb294; &#bc84;&#xc804;&#xc744; &#xc774;&#xc6a9;&#xd558;&#xc2ed;&#xc2dc;&#xc624;.

**Note**  
 &#xb798;&#xc2a4;&#xd130; &#xc2a4;&#xd398;&#xc774;&#xc2a4;&#xc5d0;&#xc11c; &#xb798;&#xc2a4;&#xd130; &#bc34;&#xb4dc;&#xc640; &#xb3c4;&#xd615;&#xc758; &#xad50;&#xcc28; &#xbd80;&#xbd84;&#xc744; &#xacc4;&#xc0b0;&#xd558;&#xb824;&#xba74;. **ST\_Clip** &#xc744; &#xc774;&#xc6a9;&#xd558;&#xc2ed;&#xc2dc;&#xc624;. ST\_Clip &#xd568;&#xc218;&#xb294; &#bcf5;&#xc218;&#xc758; &#bc34;&#xb4dc;&#xb97c; &#xac00;&#xc9c4; &#xb798;&#xc2a4;&#xd130;&#xb97c; &#xc785;&#xb825;&#bc1b;&#xc73c;&#xba70;. &#xb798;&#xc2a4;&#xd130;&#xd654;&#xb41c; &#xb3c4;&#xd615;&#xacfc; &#xc77c;&#xc5e8;&#xd558;&#xb294; &#bc34;&#xb4dc;&#xb97c; &#bc18;&#xd658;&#xd558;&#xc9c0; &#xc54a;&#xc2b5;&#xb2c8;&#xb2e4;.

**Note**  
 ST\_Intersects &#xbc0f; &#xb798;&#xc2a4;&#xd130;&#xc5d0; &#xb300;&#xd55c; &#xc778;&#xb371;&#xc2a4; &#xadf8;&#xb9ac;&#xace0;/&#xb610;&#xb294; &#xb3c4;&#xd615; &#xc5f4;&#xacfc; &#xacb0;&#xd569;&#xd574;&#xc11c; ST\_Intersection &#xd568;&#xc218;&#xb97c; &#xc0ac;&#xc6a9;&#xd574;&#xc57c; &#xd569;&#xb2c8;&#xb2e4;.

&#xac1c;&#xc120; &#xc0ac;&#xd56d;. 2.0.0&#xbd80;&#xd130; &#xb798;&#xc2a4;&#xd130; &#xc2a4;&#xd398;&#xc774;&#xc2a4;&#xad50;&#xcc28; &#xbd80;&#xbd84;&#xc744; &#xad6c;&#xd560; &#xc218; &#xc788;&#xc2b5;&#xb2c8;&#xb2e4;. 2.0.0

geomval, ST\_Intersects, ST\_MapAlgebraExpr, ST\_Clip, ST\_AsText

geomval, ST\_Intersects, ST\_MapAlgebraExpr, ST\_Clip, ST\_AsText

```
SELECT
  foo.rid,
  foo.gid,
  ST_AsText((foo.geomval).geom) As geomwkt,
  (foo.geomval).val
FROM (
  SELECT
    A.rid,
    g.gid,
    ST_Intersection(A.rast, g.geom) As geomval
  FROM dummy_rast AS A
  CROSS JOIN (
    VALUES
      (1, ST_Point(3427928, 5793243.85) ),
      (2, ST_GeomFromText('LINESTRING(3427927.85 5793243.75,3427927.8 5793243.75,3427927.8 5793243.8)')),
      (3, ST_GeomFromText('LINESTRING(1 2, 3 4)'))
    ) As g(gid,geom)
  WHERE A.rid = 2
) As foo;
```

| rid | gid | geomwkt   | val |
|-----|-----|---|-----|
| 2   | 1   | POINT(3427928 5793243.85)                                       | 249 |
| 2   | 1   | POINT(3427928 5793243.85)                                       | 253 |
| 2   | 2   | POINT(3427927.85 5793243.75)                                    | 254 |
| 2   | 2   | POINT(3427927.8 5793243.8)                                      | 251 |
| 2   | 2   | POINT(3427927.8 5793243.8)                                      | 253 |
| 2   | 2   | LINESTRING(3427927.8 5793243.75,3427927.8 5793243.8)            | 252 |
| 2   | 2   | MULTILINESTRING((3427927.8 5793243.8,3427927.8 5793243.75),...) | 250 |
| 2   | 3   | GEOMETRYCOLLECTION EMPTY  |     |

geomval, ST\_Intersects, ST\_MapAlgebraExpr, ST\_Clip, ST\_AsText

geomval, ST\_Intersects, ST\_MapAlgebraExpr, ST\_Clip, ST\_AsText

## 12.12.5 geomval, ST\_Intersects, ST\_MapAlgebraExpr, ST\_Clip, ST\_AsText

geomval, ST\_Intersects, ST\_MapAlgebraExpr, ST\_Clip, ST\_AsText



```

&#xb798;&#xc2a4;&#xd130;&#xc758; &#xd53d;&#xc140; &#xc704;&#xce58;&#xb4e4;&#xc758; &#xc9d1;&#xd569;&#xc7
(&#xc778;&#xb371;&#xc2a4;&#xac00; 0-&#xae30;&#xbc18;&#xc778;) &#xc678;&#xacfd; &#xcc28;&#xc6d0;&#xc740;
&#xb798;&#xc2a4;&#xd130; &#xbc88;&#xd638;&#xc785;&#xb2c8;&#xb2e4;. &#xc678;&#xacfd; &#xcc28;&#xc6d0;
&#xc778;&#xb371;&#xc2a4; 0&#xc5d0; &#xc788;&#xb294; &#xc704;&#xce58;&#xb294; &#xcd9c;&#xb825; &#xb798;&#xc
&#xd53d;&#xc140; &#xc704;&#xce58;&#xc785;&#xb2c8;&#xb2e4;. &#xac01; &#xc678;&#xacfd; &#xcc28;&#xc6d0;&#xc5
&#xb300;&#xd574;. &#xb0b4;&#xacfd; &#xcc28;&#xc6d0;&#xc5d0; X &#xbc0f; Y &#xb97c; &#xc704;&#xd55c; &#xb450;
&#xc694;&#xc18c;&#xac00; &#xc788;&#xc2b5;&#xb2c8;&#xb2e4;. &#xc138; &#xbc88;&#xc9f8; &#xc778;&#xc218;
userargs &#xb294; &#xc5b4;&#xb5a4; &#xc0ac;&#xc6a9;&#xc790; &#xc9c0;&#xc815; &#xc778;&#xc218;&#xb77c;&#xc
&#xb118;&#xaca8;&#xc8fc;&#xae30; &#xc704;&#xd55c; &#xac83;&#xc785;&#xb2c8;&#xb2e4;.

```

```

regprocedure &#xc778;&#xc218;&#xb97c; SQL &#xd568;&#xc218;&#xc5d0; &#xc785;&#xb825;&#xd558;&#xb824;&#xba7
&#xc785;&#xb825;&#xd558;&#xae30; &#xc704;&#xd55c; &#xc804;&#xccb4; &#xd568;&#xc218; &#xc11c;&#xba85;&#xc7
&#xd544;&#xc694;&#xd558;&#xace0;. &#xadf8; &#xb2e4;&#xc74c; regprocedure &#xc720;&#xd615;&#xc73c;&#xb85c;
&#xd615;&#xbcc0;&#xd658;&#xd574;&#xc57c; &#xd569;&#xb2c8;&#xb2e4;. &#xc55e;&#xc758; PL/pgSQL &#xd568;&#xc
&#xc778;&#xc218;&#xb85c;&#xc11c; &#xb118;&#xaca8;&#xc8fc;&#xb824;&#xba74;. &#xd574;&#xb2f9; &#xc778;&#xc21
&#xc704;&#xd55c; SQL&#xc740; &#xb2e4;&#xc74c;&#xacfc; &#xac19;&#xc2b5;&#xb2c8;&#xb2e4;.

```

```
'sample_callbackfunc(double precision[], integer[], text[])':::regprocedure
```

```

&#xd574;&#xb2f9; &#xc778;&#xc218;&#xac00; &#xd568;&#xc218;&#xc758; &#xba85;&#xce6d;. &#xd568;&#xc218;
&#xc778;&#xc218;&#xb4e4;&#xc758; &#xc720;&#xd615;. &#xba85;&#xce6d; &#xbc0f; &#xc778;&#xc218; &#xc720;&#xd
&#xac10;&#xc2fc; &#xb530;&#xc634;&#xd45c;. &#xadf8;&#xb9ac;&#xace0; regprocedure &#xb85c;&#xc758; &#xd615;&#xc
&#xd3ec;&#xd568;&#xd558;&#xace0; &#xc788;&#xb2e4;&#xb294; &#xc810;&#xc5d0; &#xc8fc;&#xc758;&#xd558;&#xc2

```

**mask** An n-dimensional array (matrix) of numbers used to filter what cells get passed to map algebra call-back function. 0 means a neighbor cell value should be treated as no-data and 1 means value should be treated as data. If weight is set to true, then the values, are used as multipliers to multiple the pixel value of that value in the neighborhood position.

**weighted** mask &#xc12;&#xc5d0; &#xac00;&#xc911;&#xce58;&#xb97c; &#xc801;&#xc6a9;&#xd574;&#xc57c; &#xd560;&#xc9c0;&#xac12;&#xc73c;&#xb85c; &#xacf1;&#xd574;&#xc57c; &#xd560;&#xc9c0;) &#xb9d0;&#xc9c0;(mask&#xb97c; &#xc785;&#xc5d0;&#xc808; &#xbc84;&#xc804;&#xc5d0;&#xb9cc; &#xc801;&#xc6a9;&#xd560;&#xc9c0;) &#xb97c; &#xd45c;&#xc2&#xbd88; &#xc12;(&#xcc38;/&#xac70;&#xc9d3;)&#xc785;&#xb2c8;&#xb2e4;.

**pixeltype** pixeltype &#xc744; &#xc124;&#xc815;&#xd560; &#xacbd;&#xc6b0;. &#xc0c8; &#xb798;&#xc2a4;&#xd130;&#xc758;&#xc34;&#xb4dc; &#xd558;&#xb098;&#xac00; &#xd574;&#xb2f9; &#xd53d;&#xc140; &#xc720;&#xd615;&#xc774; &#xb420; &#xac83;&#xc785;&#xb2c8;&#xb2e4;. pixeltype &#xc774; NULL&#xc774;&#xac70;&#xb098; &#xc0dd;&#xacbd;&#xc6b0;. &#xc0c8; &#xb798;&#xc2a4;&#xd130; &#xc34;&#xb4dc;&#xac00; &#xccab; &#xbc88;&#xc9f8; &#xb798;&#xc2a4;&#xd130;&#xc758; &#xc9c0;&#xc815;&#xb41c; &#xc34;&#xb4dc;&#xc640;(&#xc94;&#xc704; &#xc720;&#xd615;&#xc758; &#xacbd;&#xc6b0; INTERSECTION, UNION, FIRST, CUSTOM), &#xb610;&#xb294; &#xc801;&#xc808;&#xd55c; &#xb798;&#xc2a4;&#xd130;&#xc758; &#xc9c0;&#xc815;&#xb41c; &#xc34;&#xb4dc;&#xc640;&#xc720;&#xd615;&#xc758; &#xacbd;&#xc6b0; SECOND, LAST) &#xb3d9;&#xc77c;&#xd55c; &#xd53d;&#xc140; &#xc720;&#xd615;&#xc744; &#xac00;&#xc9c0;&#xac8c; &#xb420; &#xac83;&#xc785;&#xb2c8;&#xb2e4;. &#xc5b4;&#xb5 &#xc720;&#xd615;&#xc778;&#xc9c0; &#xd655;&#xc2e0;&#xd558;&#xc9c0; &#xbabb; &#xd55c;&#xb2e4;&#xba74;. &#xc5b8;&#xc81c;&#xb098; pixeltype &#xc744; &#xc124;&#xc815;&#xd558;&#xc2ed;&#xc2dc;&#xc624; &#xcd9c;&#xb825; &#xb798;&#xc2a4;&#xd130;&#xc758; &#xd53d;&#xc140; &#xc720;&#xd615;&#xc740; **ST\_BandPixelType** &#xbaa9;&#xb85d;&#xc5d0; &#xc874;&#xc7ac;&#xd558;&#xb294; &#xc720;&#xd615; &#xac00;&#xc6b4;&#xb370; &#xd558;&#xb098;&#xc774;&#xac70;&#xb098;. &#xc0dd;&#xb7b5;&#xb418;&#xac70;&#xb098;. NULL&#xb85c; &#xc124;&#xd569;&#xb2c8;&#xb2e4;.

**extenttype** &#xc0ac;&#xc6a9;&#xd560; &#xc218; &#xc788;&#xb294; &#xac12;&#xc740; INTERSECTION(&#xae30;&#xbcfc8;&#xb85c; UNION, FIRST(&#xb798;&#xc2a4;&#xd130; 1&#xac1c;&#xb97c; &#xc785;&#xb825;&#xc1b;&#xb294; &#xbcc0;&#xc885;&#xae30;&#xbcfc8;&#xac12;), SECOND, LAST, CUSTOM&#xc785;&#xb2c8;&#xb2e4;.

**customextent** extenttype &#xc774; CUSTOM&#xc77c; &#xacbd;&#xc6b0;. &#xb798;&#xc2a4;&#xd130;&#xac00; customextent &#xb97c; &#xc785;&#xb825;&#xc1b;&#xc544;&#xc57c;&#xb9cc; &#xd569;&#xb2c8;&#xb2e4;. &#xbcc0;&#xc885; 1&#xc758; 4&#xbc88;&#xc9f8; &#xc608;&#xc2dc;&#xb97c; &#xcc38;&#xc870;&#xd558;&#xc2ed;&#xc2dc;&#xc624;.

**distancex** The distance in pixels from the reference cell in x direction. So width of resulting matrix would be 2\*distancex + 1.If not specified only the reference cell is considered (neighborhood of 0).

**distancey**  $2 * \text{distancey} + 1$

**userargs** `callbackfunc`, `variadic text`, `callbackfunc`, `userargs`

#### Note



(`VARIADIC`, `PostgreSQL Query Language (SQL) Functions`, "SQL Functions with Variable Numbers of Arguments")

#### Note



`callbackfunc`, `text[]`

`1`, `2`, `3`, `4`

`2`, `3`, `4`

`4`

`2.2.0`

`2.1.0`

`1`

`1`

```
WITH foo AS (
  SELECT 1 AS rid, ST_AddBand(ST_MakeEmptyRaster(2, 2, 0, 0, 1, -1, 0, 0, 0), 1, '16 ←
    BUI', 1, 0) AS rast
)
SELECT
  ST_MapAlgebra(
    ARRAY[ROW(rast, 1)]::rastbandarg[],
    'sample_callbackfunc(double precision[], int[], text[])'::regprocedure
  ) AS rast
FROM foo
```

WITH foo AS (

```

    SELECT 1 AS rid, ST_AddBand(ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(2, 2, 0, 0, 1, ←
        -1, 0, 0, 0), 1, '16BUI', 1, 0), 2, '8BUI', 10, 0), 3, '32BUI', 100, 0) AS rast
)
SELECT
    ST_MapAlgebra(
        ARRAY[ROW(rast, 3), ROW(rast, 1), ROW(rast, 3), ROW(rast, 2)]::rastbandarg ←
        [],
        'sample_callbackfunc(double precision[], int[], text[])::regprocedure
    ) AS rast
FROM foo
```

WITH foo AS (

```

    SELECT 1 AS rid, ST_AddBand(ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(2, 2, 0, 0, 1, ←
        -1, 0, 0, 0), 1, '16BUI', 1, 0), 2, '8BUI', 10, 0), 3, '32BUI', 100, 0) AS rast ←
    UNION ALL
    SELECT 2 AS rid, ST_AddBand(ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(2, 2, 0, 1, 1, ←
        -1, 0, 0, 0), 1, '16BUI', 2, 0), 2, '8BUI', 20, 0), 3, '32BUI', 300, 0) AS rast
)
SELECT
    ST_MapAlgebra(
        ARRAY[ROW(t1.rast, 3), ROW(t2.rast, 1), ROW(t2.rast, 3), ROW(t1.rast, 2)]:: ←
        rastbandarg[],
        'sample_callbackfunc(double precision[], int[], text[])::regprocedure
    ) AS rast
FROM foo t1
CROSS JOIN foo t2
WHERE t1.rid = 1
    AND t2.rid = 2
```

WITH foo AS (

```

    SELECT 0 AS rid, ST_AddBand(ST_MakeEmptyRaster(2, 2, 0, 0, 1, -1, 0, 0, 0), 1, '16 ←
        BUI', 1, 0) AS rast UNION ALL
    SELECT 1, ST_AddBand(ST_MakeEmptyRaster(2, 2, 2, 0, 1, -1, 0, 0, 0), 1, '16BUI', 2, ←
        0) AS rast UNION ALL
    SELECT 2, ST_AddBand(ST_MakeEmptyRaster(2, 2, 4, 0, 1, -1, 0, 0, 0), 1, '16BUI', 3, ←
        0) AS rast UNION ALL

    SELECT 3, ST_AddBand(ST_MakeEmptyRaster(2, 2, 0, -2, 1, -1, 0, 0, 0), 1, '16BUI', ←
        10, 0) AS rast UNION ALL
    SELECT 4, ST_AddBand(ST_MakeEmptyRaster(2, 2, 2, -2, 1, -1, 0, 0, 0), 1, '16BUI', ←
        20, 0) AS rast UNION ALL
    SELECT 5, ST_AddBand(ST_MakeEmptyRaster(2, 2, 4, -2, 1, -1, 0, 0, 0), 1, '16BUI', ←
        30, 0) AS rast UNION ALL

    SELECT 6, ST_AddBand(ST_MakeEmptyRaster(2, 2, 0, -4, 1, -1, 0, 0, 0), 1, '16BUI', ←
        100, 0) AS rast UNION ALL
    SELECT 7, ST_AddBand(ST_MakeEmptyRaster(2, 2, 2, -4, 1, -1, 0, 0, 0), 1, '16BUI', ←
        200, 0) AS rast UNION ALL
    SELECT 8, ST_AddBand(ST_MakeEmptyRaster(2, 2, 4, -4, 1, -1, 0, 0, 0), 1, '16BUI', ←
        300, 0) AS rast
)
SELECT
    t1.rid,
```

```

        ST_MapAlgebra (
            ARRAY[ROW(ST_Union(t2.rast), 1)]::rastbandarg[],
            'sample_callbackfunc(double precision[], int[], text[])'::regprocedure,
            '32BUI',
            'CUSTOM', t1.rast,
            1, 1
        ) AS rast
FROM foo t1
CROSS JOIN foo t2
WHERE t1.rid = 4
      AND t2.rid BETWEEN 0 AND 8
      AND ST_Intersects(t1.rast, t2.rast)
GROUP BY t1.rid, t1.rast

```

&#xc774;&#xc6c3;&#xc744; &#xac00;&#xc9c4; &#xcee4;&#xbc84;&#xb9ac;&#xc9c0;&#xc758; &#xd0c0;&#xc77c;&#xb4e4;&#xc608;&#xc2dc;&#xc640; &#xc720;&#xc0ac;&#xd558;&#xc9c0;&#xb9cc; PostgreSQL 9.0&#xc5d0;&#xc11c; &#xc791;&#xb3d0;&#xcffc;&#xb9ac;&#xc785;&#xb2c8;&#xb2e4;.

```

WITH src AS (
    SELECT 0 AS rid, ST_AddBand(ST_MakeEmptyRaster(2, 2, 0, 0, 1, -1, 0, 0, 0), 1, '16 ←
      BUI', 1, 0) AS rast UNION ALL
    SELECT 1, ST_AddBand(ST_MakeEmptyRaster(2, 2, 2, 0, 1, -1, 0, 0, 0), 1, '16BUI', 2, ←
      0) AS rast UNION ALL
    SELECT 2, ST_AddBand(ST_MakeEmptyRaster(2, 2, 4, 0, 1, -1, 0, 0, 0), 1, '16BUI', 3, ←
      0) AS rast UNION ALL

    SELECT 3, ST_AddBand(ST_MakeEmptyRaster(2, 2, 0, -2, 1, -1, 0, 0, 0), 1, '16BUI', ←
      10, 0) AS rast UNION ALL
    SELECT 4, ST_AddBand(ST_MakeEmptyRaster(2, 2, 2, -2, 1, -1, 0, 0, 0), 1, '16BUI', ←
      20, 0) AS rast UNION ALL
    SELECT 5, ST_AddBand(ST_MakeEmptyRaster(2, 2, 4, -2, 1, -1, 0, 0, 0), 1, '16BUI', ←
      30, 0) AS rast UNION ALL

    SELECT 6, ST_AddBand(ST_MakeEmptyRaster(2, 2, 0, -4, 1, -1, 0, 0, 0), 1, '16BUI', ←
      100, 0) AS rast UNION ALL
    SELECT 7, ST_AddBand(ST_MakeEmptyRaster(2, 2, 2, -4, 1, -1, 0, 0, 0), 1, '16BUI', ←
      200, 0) AS rast UNION ALL
    SELECT 8, ST_AddBand(ST_MakeEmptyRaster(2, 2, 4, -4, 1, -1, 0, 0, 0), 1, '16BUI', ←
      300, 0) AS rast
)
WITH foo AS (
    SELECT
        t1.rid,
        ST_Union(t2.rast) AS rast
    FROM src t1
    JOIN src t2
        ON ST_Intersects(t1.rast, t2.rast)
        AND t2.rid BETWEEN 0 AND 8
    WHERE t1.rid = 4
    GROUP BY t1.rid
), bar AS (
    SELECT
        t1.rid,
        ST_MapAlgebra (
            ARRAY[ROW(t2.rast, 1)]::rastbandarg[],
            'raster_nmapalgebra_test(double precision[], int[], text[])':: ←
              regprocedure,
            '32BUI',
            'CUSTOM', t1.rast,
            1, 1
        ) AS rast
    FROM src t1

```

```

        JOIN foo t2
            ON t1.rid = t2.rid
    )
SELECT
    rid,
    (ST_Metadata(rast)),
    (ST_BandMetadata(rast, 1)),
    ST_Value(rast, 1, 1, 1)
FROM bar;

```

### 608: 2 3

798: 1; 34; 87;

```

WITH foo AS (
    SELECT 1 AS rid, ST_AddBand(ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(2, 2, 0, 0, 1, ←
        -1, 0, 0, 0), 1, '16BUI', 1, 0), 2, '8BUI', 10, 0), 3, '32BUI', 100, 0) AS rast
)
SELECT
    ST_MapAlgebra(
        rast, ARRAY[3, 1, 3, 2]::integer[],
        'sample_callbackfunc(double precision[], int[], text[])::regprocedure
    ) AS rast
FROM foo

```

798: 1; 34; 1;

```

WITH foo AS (
    SELECT 1 AS rid, ST_AddBand(ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(2, 2, 0, 0, 1, ←
        -1, 0, 0, 0), 1, '16BUI', 1, 0), 2, '8BUI', 10, 0), 3, '32BUI', 100, 0) AS rast
)
SELECT
    ST_MapAlgebra(
        rast, 2,
        'sample_callbackfunc(double precision[], int[], text[])::regprocedure
    ) AS rast
FROM foo

```

### 608: 4

798: 2; 34; 2;

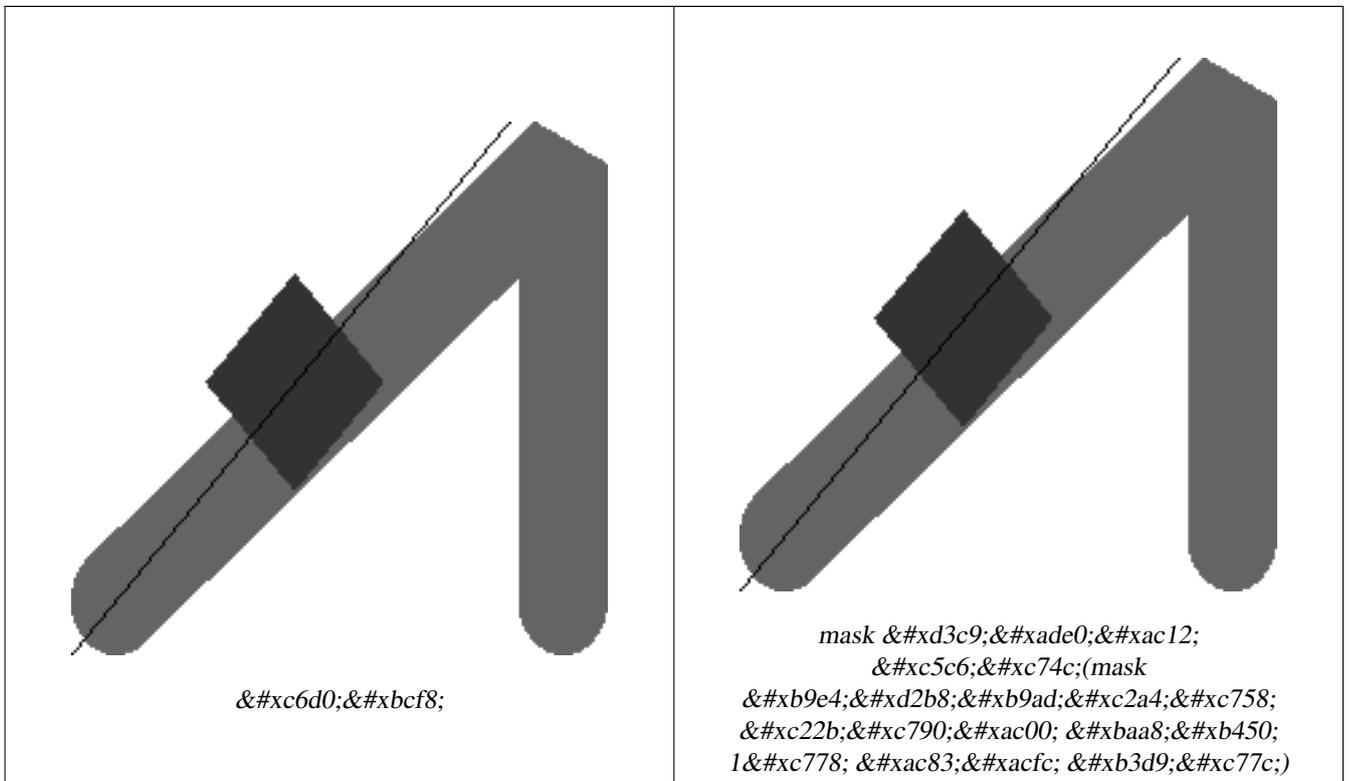
```

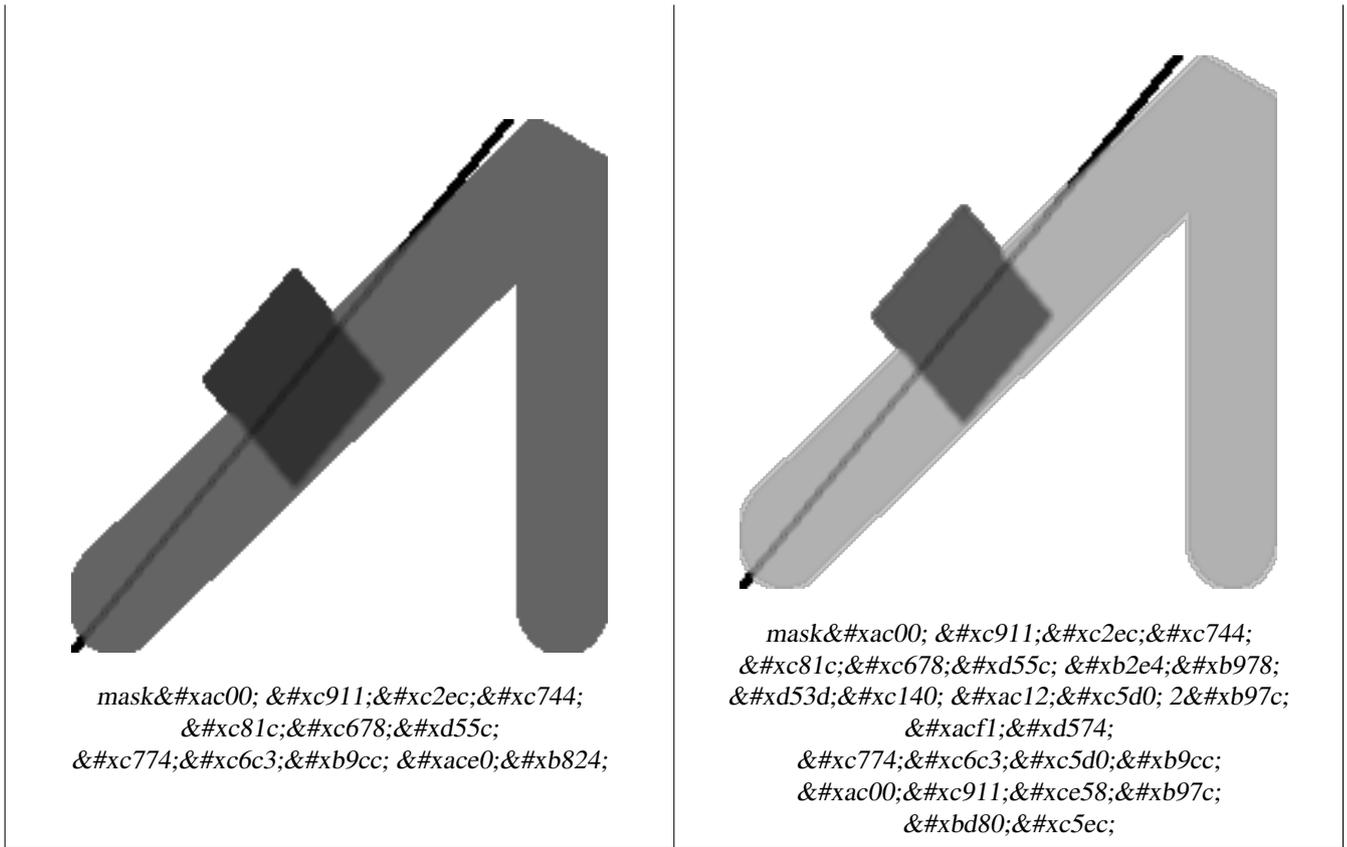
WITH foo AS (
    SELECT 1 AS rid, ST_AddBand(ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(2, 2, 0, 0, 1, ←
        -1, 0, 0, 0), 1, '16BUI', 1, 0), 2, '8BUI', 10, 0), 3, '32BUI', 100, 0) AS rast ←
    UNION ALL
    SELECT 2 AS rid, ST_AddBand(ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(2, 2, 0, 1, 1, ←
        -1, 0, 0, 0), 1, '16BUI', 2, 0), 2, '8BUI', 20, 0), 3, '32BUI', 300, 0) AS rast
)
SELECT
    ST_MapAlgebra(
        t1.rast, 2,
        t2.rast, 1,
        'sample_callbackfunc(double precision[], int[], text[])::regprocedure
    ) AS rast
FROM foo t1
CROSS JOIN foo t2
WHERE t1.rid = 1
    AND t2.rid = 2

```

**mask**

```
WITH foo AS (SELECT
  ST_SetBandNoDataValue (
  ST_SetValue (ST_SetValue (ST_AsRaster (
    ST_Buffer (
      ST_GeomFromText ('LINESTRING(50 50,100 90,100 50)'), 5,'join=bevel') ←
      ,
      200,200,ARRAY['8BUI'], ARRAY[100], ARRAY[0]), ST_Buffer('POINT(70 ←
      70) '::geometry,10,'quad_segs=1') ,50),
    'LINESTRING(20 20, 100 100, 150 98) '::geometry,1),0) AS rast )
SELECT 'original' AS title, rast
FROM foo
UNION ALL
SELECT 'no mask mean value' AS title, ST_MapAlgebra(rast,1,'ST_mean4ma(double precision[], ←
  int[], text[]) '::regprocedure) AS rast
FROM foo
UNION ALL
SELECT 'mask only consider neighbors, exclude center' AS title, ST_MapAlgebra(rast,1,' ←
  ST_mean4ma(double precision[], int[], text[]) '::regprocedure,
  '{1,1,1}, {1,0,1}, {1,1,1} '::double precision[], false) As rast
FROM foo
UNION ALL
SELECT 'mask weighted only consider neighbors, exclude center multi otehr pixel values by ←
  2' AS title, ST_MapAlgebra(rast,1,'ST_mean4ma(double precision[], int[], text[]) ':: ←
  regprocedure,
  '{2,2,2}, {2,0,2}, {2,2,2} '::double precision[], true) As rast
FROM foo;
```





**&#xccc38;&#xace0;**

[rastbandarg](#), [ST\\_Union](#), [ST\\_MapAlgebraExpr](#)

### 12.12.6 ST\_MapAlgebraExpr

ST\_MapAlgebraExpr — &#xd45c;&#xd604;&#xc2dd; &#xbc84;&#xc804; - &#xc785;&#xb825; &#xb798;&#xc2a4;&#xd130; 1&#xac1c; &#xb610;&#xb294; 2&#xac1c;, &#xbc34;&#xb4dc; &#xc778;&#xb371;&#xc2a4;, &#xadf8;&#xb9ac;&#xace0; &#xc0ac;&#xc6a9;&#xc790; &#xc9c0;&#xc815; SQL &#xd45c;&#xd604;&#xc2dd; 1&#xac1c; &#xc774;&#xc0c1;&#xc744; &#xc785;&#xb825;&#xbc1b;&#xc544; &#xbc34;&#xb4dc; 1&#xac1c;&#xb97c; &#xac00;&#xc9c4; &#xb798;&#xc2a4;&#xd130;&#xbc18;&#xd658;&#xd569;&#xb2c8;&#xb2e4;.

#### Synopsis

raster **ST\_MapAlgebra**(raster rast, integer nband, text pixeltype, text expression, double precision nodataval=NULL);  
 raster **ST\_MapAlgebra**(raster rast, text pixeltype, text expression, double precision nodataval=NULL);  
 raster **ST\_MapAlgebra**(raster rast1, integer nband1, raster rast2, integer nband2, text expression, text pixeltype=NULL, text extnttype=INTERSECTION, text nodata1expr=NULL, text nodata2expr=NULL, double precision nodatanodataval=NULL);  
 raster **ST\_MapAlgebra**(raster rast1, raster rast2, text expression, text pixeltype=NULL, text extnttype=INTERSECTION, text nodata1expr=NULL, text nodata2expr=NULL, double precision nodatanodataval=NULL);

**&#xc124;&#xba85;**

&#xd45c;&#xd604;&#xc2dd; &#xbc84;&#xc804; - &#xc785;&#xb825; &#xb798;&#xc2a4;&#xd130; 1&#xac1c; &#xb610;&#xb294; 2&#xac1c;, &#xbc34;&#xb4dc; &#xc778;&#xb371;&#xc2a4;, &#xadf8;&#xb9ac;&#xace0; &#xc0ac;&#xc6a9;&#xc790; &#xc9c0;&#xc815; SQL &#xd45c;&#xd604;&#xc2dd; 1&#xac1c; &#xc774;&#xc0c1;&#xc744; &#xc785;&#xb825;&#xbc1b;&#xc544; &#xbc34;&#xb4dc; 1&#xac1c;&#xb97c; &#xac00;&#xc9c4; &#xb798;&#xc2a4;&#xd130;&#xb97c; &#xbc18;&#xd658;&#xd569;&#xb2c8;&#xb2e4;.

2.1.0

### **1, 2**

`expression` `rast`; `band` `1`; `pixeltype` `rast`; `extenttype` `intersection`

`pixeltype` `rast`; `extenttype` `intersection`

`expression` `rast`; `band` `1`; `pixeltype` `rast`; `extenttype` `intersection`

1. `[rast]` - `rast`
2. `[rast.val]` - `rast`
3. `[rast.x]` - `rast`
4. `[rast.y]` - `rast`

### **3, 4**

`expression` `rast1`, `rast2`; `band` `1`, `band2`; `pixeltype` `rast`; `extenttype` `intersection`

`expression` `rast`; `band` `1`; `pixeltype` `rast`; `extenttype` `intersection`

`pixeltype` `rast`; `extenttype` `intersection`

`extenttype` `intersection`

1. `INTERSECTION` - `rast`

2. UNION -
3. FIRST -
4. SECOND -

```
nodata1expr rast1 NODATA rast2 NODATA
rast1 &#xc758; &#xd53d; &#xc140; &#xb4e4; &#xc774; &#xc12; &#xc774; &#xc0; &#xc0c1; &#xc751; &#xd558; &#xb294;
rast2 &#xc758; &#xd53d; &#xc140; &#xb4e4; &#xc774; &#xc12; &#xc774;
&#xc0; &#xc9c0; &#xc0; &#xc788; &#xc744; &#xb54c; &#xc5b4; &#xb5a4; &#xc83; &#xc744; &#xb18; &#xd658; &#xd558; &#xb294;
&#xc0c1; &#xc218; &#xb9cc; &#xb610; &#xb294; rast2 &#xc640; &#xb9cc;
&#xad00; &#xb828; &#xb41c; &#xb300; &#xc218; &#xd45c; &#xd604; &#xc2dd; &#xc785; &#xb2c8; &#xb2e4;.
```

```
nodata2expr rast2 NODATA rast1 NODATA
rast2 &#xc758; &#xd53d; &#xc140; &#xb4e4; &#xc774; &#xc12; &#xc774; &#xc0; &#xc0c1; &#xc751; &#xd558; &#xb294;
rast1 &#xc758; &#xd53d; &#xc140; &#xb4e4; &#xc774; &#xc12; &#xc774;
&#xc0; &#xc9c0; &#xc0; &#xc788; &#xc744; &#xb54c; &#xc5b4; &#xb5a4; &#xc83; &#xc744; &#xb18; &#xd658; &#xd558; &#xb294;
&#xc0c1; &#xc218; &#xb9cc; &#xb610; &#xb294; rast1 &#xc640; &#xb9cc;
&#xad00; &#xb828; &#xb41c; &#xb300; &#xc218; &#xd45c; &#xd604; &#xc2dd; &#xc785; &#xb2c8; &#xb2e4;.
```

```
nodatanodataval &#xc0c1; &#xc751; &#xd558; &#xb294; rast1 &#xb0f;
rast2 &#xc758; &#xd53d; &#xc140; &#xb4e4; &#xc774; &#xbaa8; &#xb450; NODATA &#xc12; &#xc77c; &#xcabd; &#xc6b;
&#xb18; &#xd658; &#xd558; &#xb294; &#xc22b; &#xc790; &#xc0c1; &#xc218; &#xc785; &#xb2c8; &#xb2e4;.
```

- expression, nodata1expr &#xb0f; nodata2expr &#xc5d0; &#xd0a4; &#xc6cc; &#xb4dc; &#xb97c; &#xc4f8; &#xc218; &#xc788; &#xc2b5; &#xb2c8; &#xb2e4;.
1. [rast1] - rast1 &#xc5d0; &#xc788; &#xb294; &#xad00; &#xc2ec; &#xd53d; &#xc140; &#xc758; &#xd53d; &#xc140; &#xc12;
  2. [rast1.val] - rast1 &#xc5d0; &#xc788; &#xb294; &#xad00; &#xc2ec; &#xd53d; &#xc140; &#xc758; &#xd53d; &#xc140; &#xc12;
  3. [rast1.x] - rast1 &#xc5d0; &#xc788; &#xb294; &#xad00; &#xc2ec; &#xd53d; &#xc140; &#xc758; 1- &#xae30; &#xb18; &#xd53d; &#xc140; &#xc5f4;
  4. [rast1.y] - rast1 &#xc5d0; &#xc788; &#xb294; &#xad00; &#xc2ec; &#xd53d; &#xc140; &#xc758; 1- &#xae30; &#xb18; &#xd53d; &#xc140; &#xd589;
  5. [rast2] - rast2 &#xc5d0; &#xc788; &#xb294; &#xad00; &#xc2ec; &#xd53d; &#xc140; &#xc758; &#xd53d; &#xc140; &#xc12;
  6. [rast2.val] - rast2 &#xc5d0; &#xc788; &#xb294; &#xad00; &#xc2ec; &#xd53d; &#xc140; &#xc758; &#xd53d; &#xc140; &#xc12;
  7. [rast2.x] - rast2 &#xc5d0; &#xc788; &#xb294; &#xad00; &#xc2ec; &#xd53d; &#xc140; &#xc758; 1- &#xae30; &#xb18; &#xd53d; &#xc140; &#xc5f4;
  8. [rast2.y] - rast2 &#xc5d0; &#xc788; &#xb294; &#xad00; &#xc2ec; &#xd53d; &#xc140; &#xc758; 1- &#xae30; &#xb18; &#xd53d; &#xc140; &#xd589;

## &#xc608; &#xc2dc;: &#xbcc0; &#xc885; 1 &#xb0f; 2

```
WITH foo AS (
  SELECT ST_SetValue(ST_AddBand(ST_MakeEmptyRaster(3, 3, 0, 0, 1, -1, 0, 0, 0), 1, '8 ←
    BUI', 1, 0), 1, 2, 5) AS rast
)
SELECT
  (ST_DumpValues(rast, 1)) [2] [1]
FROM foo;

st_dumpvalues
-----
5
(1 row)
```

**3.4**

```

WITH foo AS (
  SELECT 1 AS rid, ST_AddBand(ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(2, 2, 0, 0, 1, ←
    -1, 0, 0, 0), 1, '16BUI', 1, 0), 2, '8BUI', 10, 0), 3, '32BUI', 100, 0) AS rast ←
  UNION ALL
  SELECT 2 AS rid, ST_AddBand(ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(2, 2, 0, 1, 1, ←
    -1, 0, 0, 0), 1, '16BUI', 2, 0), 2, '8BUI', 20, 0), 3, '32BUI', 300, 0) AS rast
)
SELECT
  ST_MapAlgebra(
    t1.rast, 2,
    t2.rast, 1,
    'sample_callbackfunc(double precision[], int[], text[])'::regprocedure
  ) AS rast
FROM foo t1
CROSS JOIN foo t2
WHERE t1.rid = 1
  AND t2.rid = 2

```

**3.5**

`rastbandarg, ST_Union, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100, 101, 102, 103, 104, 105, 106, 107, 108, 109, 110, 111, 112, 113, 114, 115, 116, 117, 118, 119, 120, 121, 122, 123, 124, 125, 126, 127, 128, 129, 130, 131, 132, 133, 134, 135, 136, 137, 138, 139, 140, 141, 142, 143, 144, 145, 146, 147, 148, 149, 150, 151, 152, 153, 154, 155, 156, 157, 158, 159, 160, 161, 162, 163, 164, 165, 166, 167, 168, 169, 170, 171, 172, 173, 174, 175, 176, 177, 178, 179, 180, 181, 182, 183, 184, 185, 186, 187, 188, 189, 190, 191, 192, 193, 194, 195, 196, 197, 198, 199, 200, 201, 202, 203, 204, 205, 206, 207, 208, 209, 210, 211, 212, 213, 214, 215, 216, 217, 218, 219, 220, 221, 222, 223, 224, 225, 226, 227, 228, 229, 230, 231, 232, 233, 234, 235, 236, 237, 238, 239, 240, 241, 242, 243, 244, 245, 246, 247, 248, 249, 250, 251, 252, 253, 254, 255, 256, 257, 258, 259, 260, 261, 262, 263, 264, 265, 266, 267, 268, 269, 270, 271, 272, 273, 274, 275, 276, 277, 278, 279, 280, 281, 282, 283, 284, 285, 286, 287, 288, 289, 290, 291, 292, 293, 294, 295, 296, 297, 298, 299, 300, 301, 302, 303, 304, 305, 306, 307, 308, 309, 310, 311, 312, 313, 314, 315, 316, 317, 318, 319, 320, 321, 322, 323, 324, 325, 326, 327, 328, 329, 330, 331, 332, 333, 334, 335, 336, 337, 338, 339, 340, 341, 342, 343, 344, 345, 346, 347, 348, 349, 350, 351, 352, 353, 354, 355, 356, 357, 358, 359, 360, 361, 362, 363, 364, 365, 366, 367, 368, 369, 370, 371, 372, 373, 374, 375, 376, 377, 378, 379, 380, 381, 382, 383, 384, 385, 386, 387, 388, 389, 390, 391, 392, 393, 394, 395, 396, 397, 398, 399, 400, 401, 402, 403, 404, 405, 406, 407, 408, 409, 410, 411, 412, 413, 414, 415, 416, 417, 418, 419, 420, 421, 422, 423, 424, 425, 426, 427, 428, 429, 430, 431, 432, 433, 434, 435, 436, 437, 438, 439, 440, 441, 442, 443, 444, 445, 446, 447, 448, 449, 450, 451, 452, 453, 454, 455, 456, 457, 458, 459, 460, 461, 462, 463, 464, 465, 466, 467, 468, 469, 470, 471, 472, 473, 474, 475, 476, 477, 478, 479, 480, 481, 482, 483, 484, 485, 486, 487, 488, 489, 490, 491, 492, 493, 494, 495, 496, 497, 498, 499, 500, 501, 502, 503, 504, 505, 506, 507, 508, 509, 510, 511, 512, 513, 514, 515, 516, 517, 518, 519, 520, 521, 522, 523, 524, 525, 526, 527, 528, 529, 530, 531, 532, 533, 534, 535, 536, 537, 538, 539, 540, 541, 542, 543, 544, 545, 546, 547, 548, 549, 550, 551, 552, 553, 554, 555, 556, 557, 558, 559, 560, 561, 562, 563, 564, 565, 566, 567, 568, 569, 570, 571, 572, 573, 574, 575, 576, 577, 578, 579, 580, 581, 582, 583, 584, 585, 586, 587, 588, 589, 590, 591, 592, 593, 594, 595, 596, 597, 598, 599, 600, 601, 602, 603, 604, 605, 606, 607, 608, 609, 610, 611, 612, 613, 614, 615, 616, 617, 618, 619, 620, 621, 622, 623, 624, 625, 626, 627, 628, 629, 630, 631, 632, 633, 634, 635, 636, 637, 638, 639, 640, 641, 642, 643, 644, 645, 646, 647, 648, 649, 650, 651, 652, 653, 654, 655, 656, 657, 658, 659, 660, 661, 662, 663, 664, 665, 666, 667, 668, 669, 670, 671, 672, 673, 674, 675, 676, 677, 678, 679, 680, 681, 682, 683, 684, 685, 686, 687, 688, 689, 690, 691, 692, 693, 694, 695, 696, 697, 698, 699, 700, 701, 702, 703, 704, 705, 706, 707, 708, 709, 710, 711, 712, 713, 714, 715, 716, 717, 718, 719, 720, 721, 722, 723, 724, 725, 726, 727, 728, 729, 730, 731, 732, 733, 734, 735, 736, 737, 738, 739, 740, 741, 742, 743, 744, 745, 746, 747, 748, 749, 750, 751, 752, 753, 754, 755, 756, 757, 758, 759, 760, 761, 762, 763, 764, 765, 766, 767, 768, 769, 770, 771, 772, 773, 774, 775, 776, 777, 778, 779, 780, 781, 782, 783, 784, 785, 786, 787, 788, 789, 790, 791, 792, 793, 794, 795, 796, 797, 798, 799, 800, 801, 802, 803, 804, 805, 806, 807, 808, 809, 810, 811, 812, 813, 814, 815, 816, 817, 818, 819, 820, 821, 822, 823, 824, 825, 826, 827, 828, 829, 830, 831, 832, 833, 834, 835, 836, 837, 838, 839, 840, 841, 842, 843, 844, 845, 846, 847, 848, 849, 850, 851, 852, 853, 854, 855, 856, 857, 858, 859, 860, 861, 862, 863, 864, 865, 866, 867, 868, 869, 870, 871, 872, 873, 874, 875, 876, 877, 878, 879, 880, 881, 882, 883, 884, 885, 886, 887, 888, 889, 890, 891, 892, 893, 894, 895, 896, 897, 898, 899, 900, 901, 902, 903, 904, 905, 906, 907, 908, 909, 910, 911, 912, 913, 914, 915, 916, 917, 918, 919, 920, 921, 922, 923, 924, 925, 926, 927, 928, 929, 930, 931, 932, 933, 934, 935, 936, 937, 938, 939, 940, 941, 942, 943, 944, 945, 946, 947, 948, 949, 950, 951, 952, 953, 954, 955, 956, 957, 958, 959, 960, 961, 962, 963, 964, 965, 966, 967, 968, 969, 970, 971, 972, 973, 974, 975, 976, 977, 978, 979, 980, 981, 982, 983, 984, 985, 986, 987, 988, 989, 990, 991, 992, 993, 994, 995, 996, 997, 998, 999, 1000`

**12.12.7 ST\_MapAlgebraExpr**

`ST_MapAlgebraExpr` — `rast` (raster), `band` (integer), `expression` (text), `nodataval` (double precision, default NULL);

**Synopsis**

raster `ST_MapAlgebraExpr`(raster `rast`, integer `band`, text `pixeltype`, text `expression`, double precision `nodataval`=NULL);

**3.6****Warning**

`ST_MapAlgebraExpr` 2.1.0 `band` (integer), `expression` (text), `nodataval` (double precision, default NULL);

`ST_MapAlgebraExpr` (raster `rast`, integer `band`, text `pixeltype`, text `expression`, double precision `nodataval`=NULL);

```
&#xc9c0;&#xb9ac;&#xcc38;&#xc870;, &#xb108;&#xbe44; &#xbc0f; &#xb192;&#xc774;&#xc774;&#xc9c0;&#xb9cc;, &#xbc34;&#xb1&#xac1c;&#xb9cc; &#xac00;&#xc9c8; &#xac83;&#xc785;&#xb2c8;&#xb2e4;.
```

```
pixeltype &#xc744; &#xc124;&#xc815;&#xd560; &#xacbd;&#xc6b0;, &#xc0c8; &#xb798;&#xc2a4;&#xd130;&#xc758;
&#xbc34;&#xb4dc;&#xac00; &#xd574;&#xb2f9; &#xd53d;&#xc140; &#xc720;&#xd615;&#xc774; &#xb420; &#xac83;&#xc785;&#xb2c8;&#xb2e4;
pixeltype &#xc774; NULL&#xc77c; &#xacbd;&#xc6b0;, &#xc0c8; &#xb798;&#xc2a4;&#xd130;&#xc758; &#xbc34;&#xb4dc;&#xc785;&#xb825;
rast &#xc758; &#xbc34;&#xb4dc;&#xc640; &#xb3d9;&#xc77c;&#xd55c; &#xd53d;&#xc140; &#xc720;&#xb420; &#xac83;&#xc785;&#xb2c8;&#xb2e4;.
```

```
&#xd45c;&#xd604;&#xc2dd;&#xc5d0;&#xc11c; &#xc6d0;&#xbcf8; &#bc34;&#xb4dc;&#xc758; &#xd53d;&#xc140; &#xac12;&#xc774;
&#xcc38;&#xc870;&#xd558;&#xb294; &#xb370; [rast], 1-&#xae30;&#xbc18; &#xd53d;&#xc140; &#xc5f4; &#xc778;&#xb371;
&#xcc38;&#xc870;&#xd558;&#xb294; &#xb370; [rast.x], 1-&#xae30;&#xbc18; &#xd53d;&#xc140; &#xd589; &#xc778;&#xb371;
&#xcc38;&#xc870;&#xd558;&#xb294; &#xb370; [rast.y] &#xc6a9;&#xc5b4;&#xb97c; &#xc0ac;&#xc6a9;&#xd560; &#xc218;
&#xc788;&#xc2b5;&#xb2c8;&#xb2e4;.
```

```
2.0.0 &#xbc84;&#xc804;&#xbd80;&#xd130; &#xc0ac;&#xc6a9;&#xd560; &#xc218; &#xc788;&#xc2b5;&#xb2c8;&#xb2e4;.
```

### &#xc608;&#xc2dc;

```
&#xc6d0;&#xbcf8; &#xb798;&#xc2a4;&#xd130; 2&#xac1c;&#xb97c; &#xc785;&#xb825;&#xbc1b;&#xb294; &#xaa8;&#xb4c8;&#xd568;&#xc218;&#xc778;
&#xc6d0;&#xbcf8;&#xc73c;&#xb85c;&#xbd80;&#xd130; &#bc34;&#xb4dc; 1&#xac1c;&#xb97c; &#xac00;&#xc9c4; &#xc0c8; &#xb798;&#xc2a4;&#xd130;&#xb97c;
&#xc0dd;&#xc131;&#xd569;&#xb2c8;&#xb2e4;.
```

```
ALTER TABLE dummy_rast ADD COLUMN map_rast raster;
CREATE FUNCTION mod_fct(pixel float, pos integer[], variadic args text[])
RETURNS float
AS $$
BEGIN
    RETURN pixel::integer % 2;
END;
$$
LANGUAGE 'plpgsql' IMMUTABLE;

UPDATE dummy_rast SET map_rast = ST_MapAlgebraFct(rast,NULL,'mod_fct(float,integer[],text ←
[])':regprocedure) WHERE rid = 2;

SELECT ST_Value(rast,1,i,j) As origval, ST_Value(map_rast, 1, i, j) As mapval
FROM dummy_rast CROSS JOIN generate_series(1, 3) AS i CROSS JOIN generate_series(1,3) AS j
WHERE rid = 2;
```

| origval | mapval |
|---------|--------|
| 253     | 1      |
| 254     | 0      |
| 253     | 1      |
| 253     | 1      |
| 254     | 0      |
| 254     | 0      |
| 250     | 0      |
| 254     | 0      |
| 254     | 0      |

```
&#xc7ac;&#xbd84;&#xb958;&#xb97c; &#xc70;&#xce58;&#xace0; NODATA &#xac12;&#xc744; 0&#xc73c;&#xb85c; &#xc124;&#xc774;
&#xc6d0;&#xbcf8;&#xc73c;&#xb85c;&#xbd80;&#xd130; &#xd53d;&#xc140; &#xc720;&#xd615;&#xc774; 2BUI&#xc778;,
&#bc34;&#xb4dc; 1&#xac1c;&#xb97c; &#xac00;&#xc9c4; &#xc0c8; &#xb798;&#xc2a4;&#xd130;&#xb97c; &#xc0dd;&#xc131;&#xd569;&#xb2c8;&#xb2e4;.
```

```
ALTER TABLE dummy_rast ADD COLUMN map_rast2 raster;
CREATE FUNCTION classify_fct(pixel float, pos integer[], variadic args text[])
RETURNS float
AS
$$
```

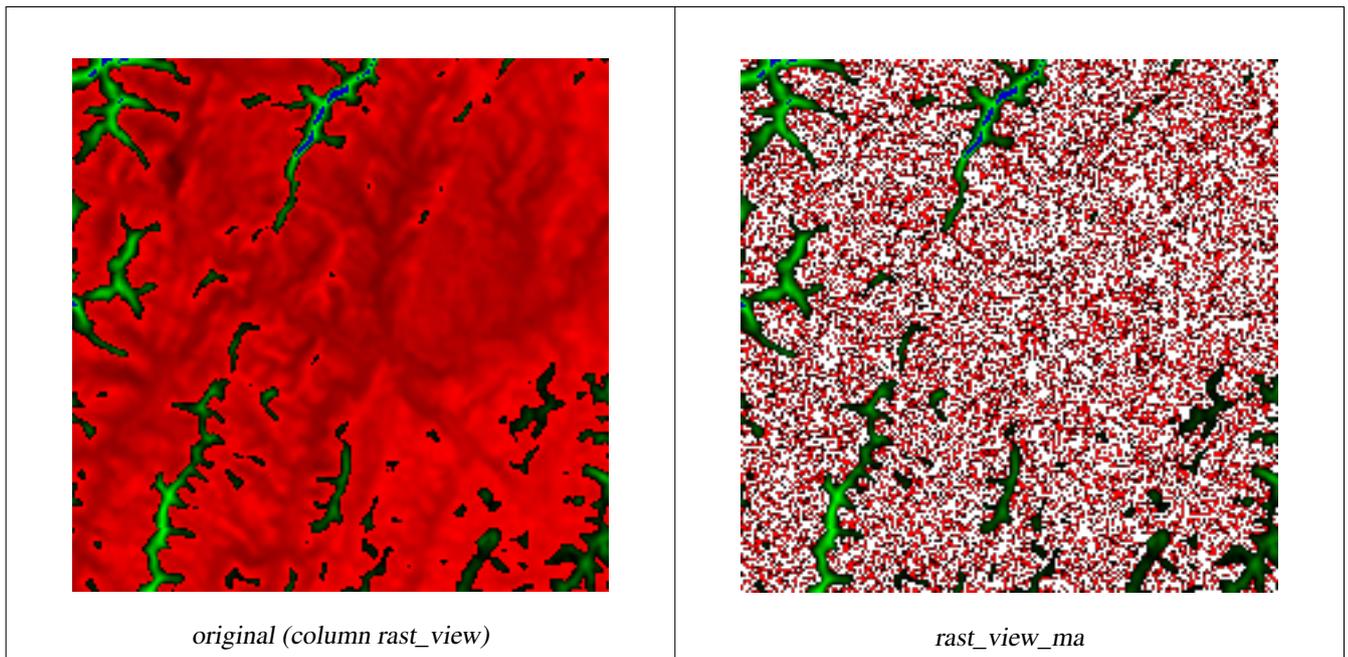
```
DECLARE
  nodata float := 0;
BEGIN
  IF NOT args[1] IS NULL THEN
    nodata := args[1];
  END IF;
  IF pixel < 251 THEN
    RETURN 1;
  ELSIF pixel = 252 THEN
    RETURN 2;
  ELSIF pixel > 252 THEN
    RETURN 3;
  ELSE
    RETURN nodata;
  END IF;
END;
$$
LANGUAGE 'plpgsql';
UPDATE dummy_rast SET map_rast2 = ST_MapAlgebraFct(rast,'2BUI','classify_fct(float,integer ←
  [],text[])'::regprocedure, '0') WHERE rid = 2;

SELECT DISTINCT ST_Value(rast,1,i,j) As origval, ST_Value(map_rast2, 1, i, j) As mapval
FROM dummy_rast CROSS JOIN generate_series(1, 5) AS i CROSS JOIN generate_series(1,5) AS j
WHERE rid = 2;

  origval | mapval
-----+-----
      249 |      1
      250 |      1
      251 |
      252 |      2
      253 |      3
      254 |      3

SELECT ST_BandPixelType(map_rast2) As b1pixtyp
FROM dummy_rast WHERE rid = 2;

  b1pixtyp
-----
  2BUI
```



The image shows the result of applying the `rast_plus_tan` function to a raster. The original raster (left) has a red background with green branching features. The resulting raster (right) shows a noisy red and white pattern with the green branching features preserved.

```

CREATE FUNCTION rast_plus_tan(pixel float, pos integer[], variadic args text[])
RETURNS float
AS
$$
BEGIN
    RETURN tan(pixel) * pixel;
END;
$$
LANGUAGE 'plpgsql';

SELECT ST_AddBand(
    ST_AddBand(
        ST_AddBand(
            ST_MakeEmptyRaster(rast_view),
            ST_MapAlgebraFct(rast_view,1,NULL,'rast_plus_tan(float,integer[], ←
                text[])'::regprocedure)
        ),
        ST_Band(rast_view,2)
    ),
    ST_Band(rast_view, 3) As rast_view_ma
)
FROM wind
WHERE rid=167;
  
```

**ST\_MapAlgebraFct**

[ST\\_MapAlgebraExpr](#), [ST\\_MapAlgebraFct](#), [ST\\_BandPixelType](#), [ST\\_GeoReference](#), [ST\\_Value](#)

## 12.12.8 ST\_MapAlgebraExpr

ST\_MapAlgebraExpr — raster ST\_MapAlgebraExpr(raster rast1, raster rast2, text expression, text pixeltype=same\_as\_rast1\_band, text extenttype=INTERSECTION, text nodata1expr=NULL, text nodata2expr=NULL, double precision nodatanodataval=NULL); raster ST\_MapAlgebraExpr(raster rast1, integer band1, raster rast2, integer band2, text expression, text pixeltype=same\_as\_rast1\_band, text extenttype=INTERSECTION, text nodata1expr=NULL, text nodata2expr=NULL, double precision nodatanodataval=NULL);

### Synopsis

raster **ST\_MapAlgebraExpr**(raster rast1, raster rast2, text expression, text pixeltype=same\_as\_rast1\_band, text extenttype=INTERSECTION, text nodata1expr=NULL, text nodata2expr=NULL, double precision nodatanodataval=NULL);  
 raster **ST\_MapAlgebraExpr**(raster rast1, integer band1, raster rast2, integer band2, text expression, text pixeltype=same\_as\_rast1\_band, text extenttype=INTERSECTION, text nodata1expr=NULL, text nodata2expr=NULL, double precision nodatanodataval=NULL);

**Warning**



#### Warning

**ST\_MapAlgebraExpr** 2.1.0 raster ST\_MapAlgebraExpr(raster rast1, raster rast2, text expression, text pixeltype=same\_as\_rast1\_band, text extenttype=INTERSECTION, text nodata1expr=NULL, text nodata2expr=NULL, double precision nodatanodataval=NULL); raster ST\_MapAlgebraExpr(raster rast1, integer band1, raster rast2, integer band2, text expression, text pixeltype=same\_as\_rast1\_band, text extenttype=INTERSECTION, text nodata1expr=NULL, text nodata2expr=NULL, double precision nodatanodataval=NULL);

raster ST\_MapAlgebraExpr(raster rast1, raster rast2, text expression, text pixeltype=same\_as\_rast1\_band, text extenttype=INTERSECTION, text nodata1expr=NULL, text nodata2expr=NULL, double precision nodatanodataval=NULL); raster ST\_MapAlgebraExpr(raster rast1, integer band1, raster rast2, integer band2, text expression, text pixeltype=same\_as\_rast1\_band, text extenttype=INTERSECTION, text nodata1expr=NULL, text nodata2expr=NULL, double precision nodatanodataval=NULL);

**expression** **ST\_MapAlgebraExpr**(raster rast1, raster rast2, text expression, text pixeltype=same\_as\_rast1\_band, text extenttype=INTERSECTION, text nodata1expr=NULL, text nodata2expr=NULL, double precision nodatanodataval=NULL); raster **ST\_MapAlgebraExpr**(raster rast1, integer band1, raster rast2, integer band2, text expression, text pixeltype=same\_as\_rast1\_band, text extenttype=INTERSECTION, text nodata1expr=NULL, text nodata2expr=NULL, double precision nodatanodataval=NULL);

**pixeltype** **ST\_MapAlgebraExpr**(raster rast1, raster rast2, text expression, text pixeltype=same\_as\_rast1\_band, text extenttype=INTERSECTION, text nodata1expr=NULL, text nodata2expr=NULL, double precision nodatanodataval=NULL); raster **ST\_MapAlgebraExpr**(raster rast1, integer band1, raster rast2, integer band2, text expression, text pixeltype=same\_as\_rast1\_band, text extenttype=INTERSECTION, text nodata1expr=NULL, text nodata2expr=NULL, double precision nodatanodataval=NULL);

**extenttype** INTERSECTION, UNION, FIRST, SECOND;

1. INTERSECTION - Returns the area of the intersection of the two polygons.
2. UNION - Returns the area of the union of the two polygons.
3. FIRST - Returns the area of the first polygon that intersects the second polygon.
4. SECOND - Returns the area of the second polygon that intersects the first polygon.

**nodata1expr** rast1 NODATA; **nodata2expr** rast2 NODATA;

**nodatanodataval** rast1 NODATA, rast2 NODATA;

**pixeltype** rast1, rast2; **pixeltype** NULL;

**2.0.0** **&#xc608;&#xc2dc;: &#xc34;&#xb4dc; 2&#xac1c;&#xc758; &#xad50;&#xcc28; &#xc0f; &#xd1b5;&#xd569;**

**&#xc608;&#xc2dc;: &#xc34;&#xb4dc; 2&#xac1c;&#xc758; &#xad50;&#xcc28; &#xc0f; &#xd1b5;&#xd569;**

```
-- &#xb798;&#xc2a4;&#xd130; &#xc9d1;&#xd569; &#xc0dd;&#xc131; --
DROP TABLE IF EXISTS fun_shapes;
CREATE TABLE fun_shapes(rid serial PRIMARY KEY, fun_name text, rast raster);

-- &#xb9e4;&#xc0ac;&#xcd94;&#xc138;&#xce20; &#xc8fc; &#xbbf8;&#xd130; &#xb2e8;&#xc704; &#xd3c9;&#xba74;&#xc758; &#bcf4;&#xc2a4;&#xd134; &#xc8fc;&#xc704;&#xc5d0; &#ba87; &#xac1c;&#xc758; &#xd615;&#xc0c1;&#xc744; &#xc0bd;&#xc785; --
INSERT INTO fun_shapes(fun_name, rast)
VALUES ('ref', ST_AsRaster(ST_MakeEnvelope(235229, 899970, 237229, 901930,26986),200,200,'8 BUI',0,0));

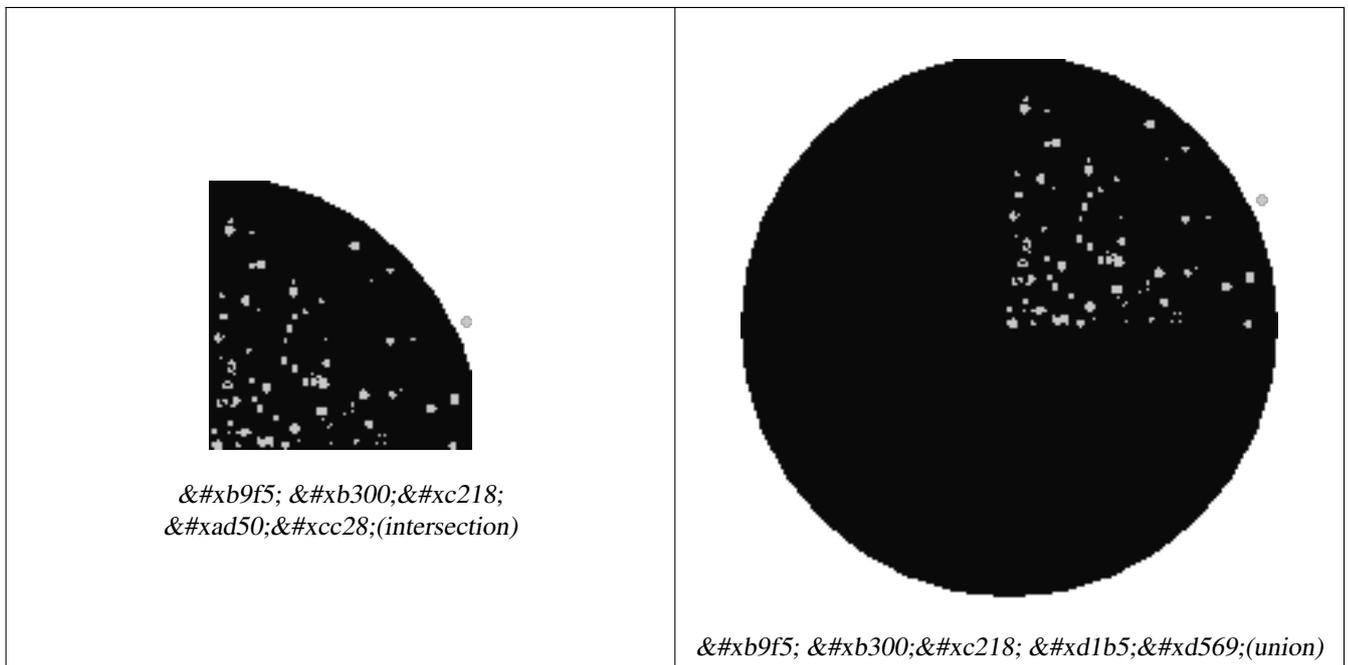
INSERT INTO fun_shapes(fun_name,rast)
```

```

WITH ref(rast) AS (SELECT rast FROM fun_shapes WHERE fun_name = 'ref' )
SELECT 'area' AS fun_name, ST_AsRaster(ST_Buffer(ST_SetSRID(ST_Point(236229, 900930),26986) ←
, 1000),
                                ref.rast,'8BUI', 10, 0) As rast
FROM ref
UNION ALL
SELECT 'rand bubbles',
      ST_AsRaster(
        (SELECT ST_Collect(geom)
         FROM (SELECT ST_Buffer(ST_SetSRID(ST_Point(236229 + i*random()*100, 900930 + j* ←
random()*100),26986), random()*20) As geom
              FROM generate_series(1,10) As i, generate_series(1,10) As j
              ) As foo ), ref.rast,'8BUI', 200, 0)
FROM ref;

-- &#xb9e4;&#xd551; --
SELECT ST_MapAlgebraExpr(
      area.rast, bub.rast, '[rast2.val]', '8BUI', 'INTERSECTION', '[rast2.val]', ←
'[rast1.val]') As interrast,
      ST_MapAlgebraExpr(
      area.rast, bub.rast, '[rast2.val]', '8BUI', 'UNION', '[rast2.val]', ←
'[rast1.val]') As unionrast
FROM
  (SELECT rast FROM fun_shapes WHERE
   fun_name = 'area') As area
CROSS JOIN (SELECT rast
FROM fun_shapes WHERE
 fun_name = 'rand bubbles') As bub

```



**&#xc608;&#xc2dc;: &#xce94;&#xbc84;&#xc2a4; &#xc0c1;&#xc5d0; &#xb798;&#xc2a4;&#xd130;&#xb4e4;&#xc744; &#xac1c;&#xb34;&#xb4dc;&#xb85c;&#xc11c; &#xc624;&#xbc84;&#xb808;&#xc774;**

```

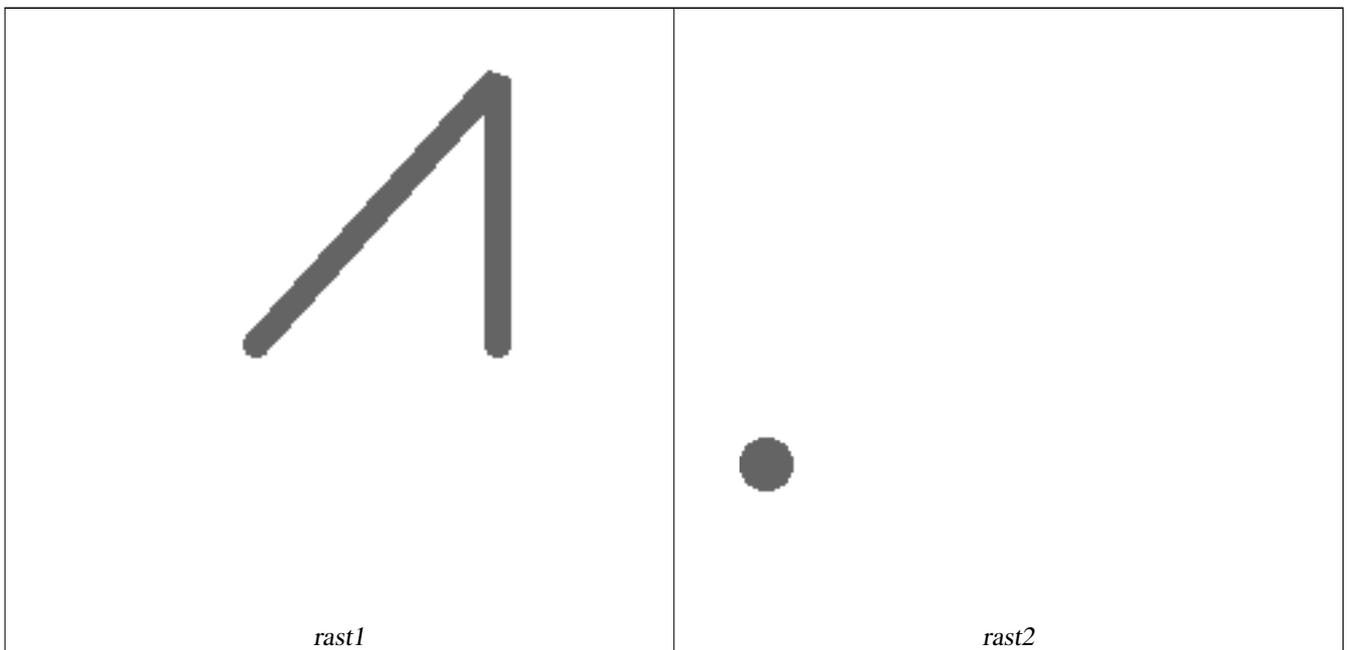
-- &#xbaa8;&#xb4e0; &#xb2e8;&#xc77c; &#xbc34;&#xb4dc; &#xb798;&#xc2a4;&#xd130;&#xb97c; ←
&#xd68c;&#xc0c9;&#xc870;&#xb85c; &#xb9cc;&#xb4e4;&#xae30; &#xc704;&#xd574; ST_AsPNG ←
&#xb97c; &#xd1b5;&#xd574; &#xc774;&#xbbf8;&#xc9c0;&#xb97c; ←
&#xb80c;&#xb354;&#xb9c1;&#xd569;&#xb2c8;&#xb2e4;. --

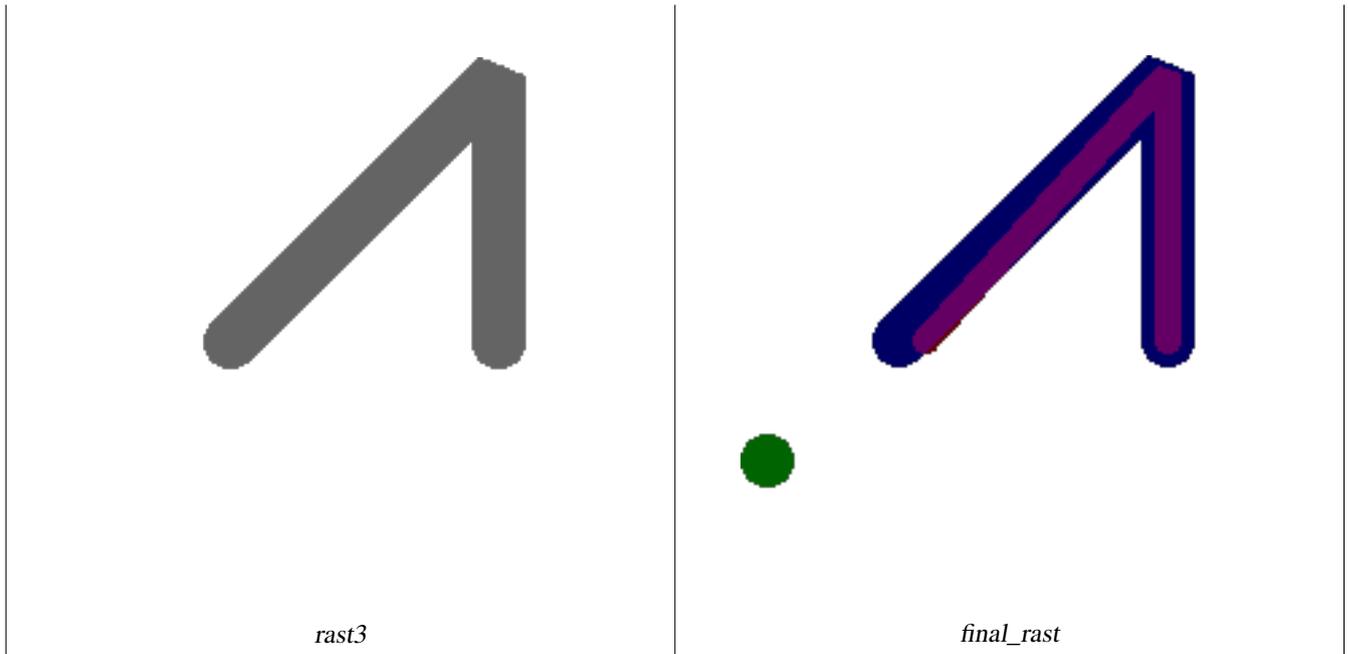
```

```

WITH mygeoms
  AS ( SELECT 2 As bnum, ST_Buffer(ST_Point(1,5),10) As geom
      UNION ALL
      SELECT 3 AS bnum,
            ST_Buffer(ST_GeomFromText('LINESTRING(50 50,150 150,150 50)'), 10,'join=↔
            bevel') As geom
      UNION ALL
      SELECT 1 As bnum,
            ST_Buffer(ST_GeomFromText('LINESTRING(60 50,150 150,150 50)'), 5,'join=↔
            bevel') As geom
    ),
  -- 1:1 canvas
  canvas
  AS (SELECT ST_AddBand(ST_MakeEmptyRaster(200,
    200,
    ST_XMin(e)::integer, ST_YMax(e)::integer, 1, -1, 0, 0) , '8BUI'::text,0) As rast
    FROM (SELECT ST_Extent(geom) As e,
          Max(ST_SRID(geom)) As srid
         from mygeoms
        ) As foo
    ),
  rbands AS (SELECT ARRAY(SELECT ST_MapAlgebraExpr(canvas.rast, ST_AsRaster(m.geom, canvas
    .rast, '8BUI', 100),
    '[rast2.val]', '8BUI', 'FIRST', '[rast2.val]', '[rast1.val]') As rast
    FROM mygeoms AS m CROSS JOIN canvas
    ORDER BY m.bnum) As rasts
    )
  SELECT rasts[1] As rast1 , rasts[2] As rast2, rasts[3] As rast3, ST_AddBand(
    ST_AddBand(rasts[1],rasts[2]), rasts[3]) As final_rast
  FROM rbands;

```





**&#xc608;&#xc2dc;: &#xd56d;&#xacf5;&#xc0ac;&#xc9c4; &#xc0c1;&#xc5d0; &#xc120;&#xd0dd;&#xb41c; &#xd68d;&#xc9c0;&#&#xb450;&#xae8; 2&#xbbf8;&#xd130;&#xc758; &#xacbd;&#xacc4;&#xc120;&#xc744; &#xc624;&#xbc84;&#xb808;&#xc774;**

```
-- &#xc98;&#xc74c; 2&#xac1c;&#xc758; &#xbc34;&#xb4dc;&#xb97c; <-
&#xc798;&#xb77c;&#xb0b4;&#xace0; &#xc138; &#xbc88;&#xc9f8; &#xbc34;&#xb4dc;&#xb97c; <-
&#xb3c4;&#xd615;&#xacfc; &#xacb9;&#xce5c; , &#xbc34;&#xb4dc; 3&#xac1c;&#xb97c; <-
&#xac00;&#xc9c4; &#xc0c8; &#xb798;&#xc2a4;&#xd130;&#xb97c; <-
&#xc0dd;&#xc131;&#xd569;&#xb2c8;&#xb2e4; .
-- PostGIS &#xc708;&#xb3c4;&#xc6b0; 64&#xbe44;&#xd2b8; <-
&#xc124;&#xce58;&#xbcf8;&#xc5d0;&#xc11c; &#xc774; &#xcffc;&#xb9ac;&#xac00; 3.6&#xcd08; <-
&#xac78;&#xb838;&#xc2b5;&#xb2c8;&#xb2e4; .
WITH pr AS
-- &#xc5f0;&#xc0b0; &#xc21c;&#xc11c;&#xc5d0; <-
&#xc8fc;&#xc758;&#xd558;&#xc2ed;&#xc2dc;&#xc624; : &#xaa8;&#xb4e0; <-
&#xb798;&#xc2a4;&#xd130;&#xb97c; &#xd574;&#xb2f9; &#xc9c0;&#xc5ed;&#xc758; <-
&#xcc28;&#xc6d0;&#xc5d0; &#xb9de;&#xac8c; &#xc798;&#xb77c;&#xb0c5;&#xb2c8;&#xb2e4; .
(SELECT ST_Clip(rast,ST_Expand(geom,50) ) As rast, g.geom
FROM aerials.o_2_boston AS r INNER JOIN
-- &#xad00;&#xc2ec; &#xd68d;&#xc9c0;&#xb97c; &#xd1b5;&#xd569;&#xd574;&#xc11c; <-
&#xb098;&#xc911;&#xc5d0; &#xad50;&#xcc28;&#xc2dc;&#xd0ac; &#xc218; &#xc788;&#xb294; <-
&#xb2e8;&#xc77c; &#xb3c4;&#xd615;&#xc73c;&#xb85c; &#xb9cc;&#xb4ed;&#xb2c8;&#xb2e4; .
(SELECT ST_Union(ST_Transform(the_geom,26986)) AS geom
FROM landparcels WHERE pid IN('0303890000', '0303900000')) As g
ON ST_Intersects(rast::geometry, ST_Expand(g.geom,50)
),
-- &#xadf8;&#xb9ac;&#xace0; &#xb798;&#xc2a4;&#xd130; &#xc870;&#xac01;&#xb4e4;&#xc744; <-
&#xd1b5;&#xd569;&#xd569;&#xb2c8;&#xb2e4; .
-- &#xb798;&#xc2a4;&#xd130;&#xb97c; ST_Union&#xc73c;&#xb85c; <-
&#xd1b5;&#xd569;&#xd558;&#xb294; &#xc791;&#xc5c5;&#xc740; &#xc870;&#xae08; <-
&#xb290;&#xb9b0; &#xd3b8;&#xc774;&#xc9c0;&#xb9cc; , &#xb798;&#xc2a4;&#xd130;&#xb97c; <-
&#xc798;&#xac8c; &#xcabc;&#xac24;&#xc218;&#xb85d; &#xc18d;&#xb3c4;&#xb294; <-
&#xbe68;&#xb77c;&#xc9d1;&#xb2c8;&#xb2e4; .
-- &#xb530;&#xb77c;&#xc11c; &#xba3c;&#xc800; &#xc798;&#xb77c;&#xb0b8; <-
&#xb2e4;&#xc74c;&#xc5d0; &#xd1b5;&#xd569;&#xd558;&#xb294; &#xd3b8;&#xc774; <-
&#xc88b;&#xc2b5;&#xb2c8;&#xb2e4; .
prunion AS
(SELECT ST_AddBand(NULL, ARRAY[ST_Union(rast,1),ST_Union(rast,2),ST_Union(rast,3)] ) As <-
clipped,geom
```

```

FROM pr
GROUP BY geom)
-- 0x798;0x2a4;d130; 0xc870;0xac01;b4e4;0xc744;  ←
  0xd1b5;0xd569;0xd558;0xace0; 0xd68d;0xc9c0; 0xacbd;0xacc4;0xc120;0xacfc;  ←
  0xacb9;0xce5c;
-- 0xacb0;0xacfc; 0xb798;0xc2a4;d130;b97c;  ←
  0xbc18;0xd658;0xd569;b2c8;b2e4;.
-- 0xcc98;0xc74c; 20xac1c;0xc758; 0xbc34;b4dc;b97c; 0xcd94;0xc00;0xd55c;  ←
  0xb2e4;0xc74c;, 0xc138; 0xbc88;0xc9f8; 0xbc34;b4dc;0xc640;  ←
  0xb3c4;0xd615;0xc5d0; 0xb9f5; 0xb300;0xc218;b97c;  ←
  0xc801;0xc6a9;0xd569;b2c8;b2e4;.
SELECT ST_AddBand(ST_Band(clipped,ARRAY[1,2])
, ST_MapAlgebraExpr(ST_Band(clipped,3), ST_AsRaster(ST_Buffer(ST_Boundary(geom),2), ←
  clipped, '8BUI',250),
  '[rast2.val]', '8BUI', 'FIRST', '[rast2.val]', '[rast1.val]') ) As rast
FROM prunion;

```



0xd30c;0xb780;0xc0c9; 0xb77c;0xc778;0xc774; 0xc120;0xd0dd;0xb41c; 0xd68d;0xc9c0;0xc758;  
0xacbd;0xacc4;0xc120;0xc785;0xb2c8;0xb2e4;.

0xcc38;0xace0;

[ST\\_MapAlgebraExpr](#), [ST\\_AddBand](#), [ST\\_AsPNG](#), [ST\\_AsRaster](#), [ST\\_MapAlgebraFct](#), [ST\\_BandPixelType](#), [ST\\_GeoReference](#),  
[ST\\_Value](#), [ST\\_Union](#), [ST\\_Union](#)

## 12.12.9 ST\_MapAlgebraFct

[ST\\_MapAlgebraFct](#) — 0xb798;0xc2a4;d130; 0xbc34;b4dc; 10xac1c; 0xbc84;0xc804;: 0xc785;0xb825;  
0xb798;0xc2a4;d130;0xc5d0; 0xb300;0xd574; 0xc720;0xd6a8;0xd55c; PostgreSQL 0xb300;0xc218; 0xc5f0;  
0xc801;0xc6a9;0xd574;0xc11c; 0xd615;0xc131;0xb418;0xace0;, 0xc124;0xc815;0xd55c; 0xd53d;0xc140;  
0xc720;0xd615;0xc744; 0xc00;0xc9c4;, 0xbc34;b4dc; 10xac1c;0xb97c; 0xc00;0xc9c4; 0xc0c8; 0xb798;0



```

    &#xc0c1;&#xad00;&#xc5c6;&#xc774;) &#xac1c;&#xbcc4;&#xc801;&#xc778; &#xb798;&#xc2a4;&#xd130; &#xc140;&#xc758;
    &#xac12;&#xc785;&#xb2c8;&#xb2e4;. &#xb450; &#xbcb8;&#xc9f8; &#xc778;&#xc218;&#xb294; &#xd604;&#xc7ac; &#xc98;&#xc758;
    &#xc140;&#xc758; &#xc704;&#xc58;&#xb97c; '{x,y}' &#xc11c;&#xc2dd;&#xc73c;&#xb85c; &#xd45c;&#xd604;&#xd55c;
    &#xac83;&#xc785;&#xb2c8;&#xb2e4;. &#xc138; &#xbcb8;&#xc9f8; &#xc778;&#xc218;&#xb294; ST_MapAlgebraFct &#xc758;
    &#xbaa8;&#xb4e0; &#xb098;&#xba38;&#xc9c0; &#xd30c;&#xb77c;&#xbbf8;&#xd130;&#xb4e4;&#xc744; userfunction
    &#xc744; &#xd1b5;&#xd574; &#xc124;&#xc815;&#xd574;&#xc57c; &#xd55c;&#xb2e4;&#xb294; &#xac83;&#xc744; &#xc758;&#xc758;
regprocedure &#xc778;&#xc218;&#xb97c; SQL &#xd568;&#xc218;&#xc5d0; &#xc785;&#xb825;&#xd558;&#xb824;&#xba74;
&#xc785;&#xb825;&#xd558;&#xae30; &#xc704;&#xd55c; &#xc804;&#xc9c4; &#xd568;&#xc218; &#xc11c;&#xba85;&#xc774;
&#xd544;&#xc694;&#xd558;&#xc9c0; &#xadf8; &#xb2e4;&#xc74c; regprocedure &#xc720;&#xd615;&#xc73c;&#xb85c;
&#xd615;&#xbcc0;&#xd658;&#xd574;&#xc57c; &#xd569;&#xb2c8;&#xb2e4;. &#xc55e;&#xc758; PL/pgSQL &#xd568;&#xc218;&#xc778;
&#xc218;&#xb85c;&#xc11c; &#xb118;&#xcaca8;&#xc8fc;&#xb824;&#xba74;. &#xd574;&#xb2f9; &#xc778;&#xc218;&#xc704;
&#xd55c; SQL&#xc740; &#xb2e4;&#xc74c;&#xc9c4; &#xc19;&#xc2b5;&#xb2c8;&#xb2e4;:

```

```
'simple_function(float, integer[], text[])'::regprocedure
```

```

&#xd574;&#xb2f9; &#xc778;&#xc218;&#xcac00; &#xd568;&#xc218;&#xc758; &#xba85;&#xc9c4;. &#xd568;&#xc218; &#xc778;&#xc720;
&#xd615;. &#xba85;&#xc9c4; &#xbcbf; &#xc778;&#xc218; &#xc720;&#xd615;&#xc744; &#xc10;&#xc2fc; &#xb530;&#xc758;
&#xadf8;&#xb9ac;&#xc9c0; regprocedure &#xb85c;&#xc758; &#xd615;&#xbcc0;&#xd658;&#xc744; &#xd3ec;&#xd568;&#xd558;
&#xc788;&#xb2e4;&#xb294; &#xc810;&#xc5d0; &#xc8fc;&#xc758;&#xd558;&#xc2ed;&#xc2dc;&#xc624;.

```

```

userfunction &#xc5d0; &#xb4e4;&#xc5b4;&#xcac00;&#xb294; &#xc138; &#xbcb8;&#xc9f8; &#xc778;&#xc218;&#xb294;
variadic text &#xb30;&#xc5f4;&#xc785;&#xb2c8;&#xb2e4;. &#xc5b4;&#xb5a4; ST_MapAlgebraFct &#xd568;&#xc218;
&#xd638;&#xcd9c;&#xc5d0;&#xb3c4; &#xc785;&#xb825;&#xb418;&#xb294; &#xae38;&#xc9c0; &#xae34; &#xd14d;&#xc2a4;&#xc778;
&#xc218;&#xb4e4;&#xc774; &#xbaa8;&#xb450; &#xc9c0;&#xc815;&#xb41c; userfunction &#xc5d0; &#xb118;&#xc778;
args &#xc778;&#xc218;&#xc5d0; &#xb2f4;&#xcaca8;&#xc9d1;&#xb2c8;&#xb2e4;.

```

#### Note



([ST\\_MapAlgebraFct](#) &#xc778;&#xc218;&#xb97c; &#xc785;&#xb825;&#xbcbf;&#xb294;) VARIADIC [ST\\_MapAlgebraFct](#) &#xd0a4;&#xc6cc;&#xb4dc;&#xc5d0; &#xb300;&#xd55c; &#xb354; &#xc790;&#xc138;&#xd55c; &#xc815;&#xbcbf;&#xb97c; &#xc54c;&#xc9c0; &#xc2f6;&#xb2e4;&#xba74;. PostgreSQL [Query Language \(SQL\) Functions](#) &#xc758; "SQL Functions with Variable Numbers of Arguments" &#xb2e8;&#xc6d0;&#xc744; &#xc38;&#xc870;&#xd558;&#xc2ed;&#xc2dc;&#xc624;.

#### Note



[ST\\_MapAlgebraFct](#) &#xcac04; &#xc9c4;&#xb9ac;&#xb97c; &#xc704;&#xd574; &#xc0ac;&#xc6a9;&#xc790; &#xd568;&#xc218;&#xc5d0; &#xc5b4;&#xb5a4; &#xc778;&#xc218;&#xb97c; &#xb118;&#xcaca8;&#xc8fc;&#xae30;&#xb85c; &#xd558;&#xc9c0; &#xb9d0;&#xc9c0;&#xc5d0; &#xc0c1;&#xad00;&#xc5c6;&#xc774;. userfunction &#xc5d0; &#xb4e4;&#xc5b4;&#xcac00;&#xb294; text[] &#xc778;&#xc218;&#xb294; &#xd544;&#xc694;&#xd569;&#xb2c8;&#xb2e4;.

2.0.0 [ST\\_MapAlgebraFct](#) &#xc804;&#xbd80;&#xd130; &#xc0ac;&#xc6a9;&#xd560; &#xc218; &#xc788;&#xc2b5;&#xb2c8;&#xb2e4;.

#### [ST\\_MapAlgebraFct](#)

[ST\\_MapAlgebraFct](#) &#xbcf8; &#xb798;&#xc2a4;&#xd130; 2&#xcac1c;&#xb97c; &#xc785;&#xb825;&#xbcbf;&#xb294; &#xbaa8;&#xb4c8;&#xc758; &#xd568;&#xc218;&#xc778; &#xc6d0;&#xbcf8;&#xc73c;&#xb85c;&#xbd80;&#xd130; &#xb34;&#xb4dc; 1&#xcac1c;&#xb97c; &#xcac00;&#xc9c4; &#xc0c8; &#xb798;&#xc2a4;&#xd130;&#xb97c; &#xc0dd;&#xc131;&#xd569;&#xb2c8;&#xb2e4;.

```

ALTER TABLE dummy_rast ADD COLUMN map_rast raster;
CREATE FUNCTION mod_fct(pixel float, pos integer[], variadic args text[])
RETURNS float
AS $$
BEGIN
    RETURN pixel::integer % 2;

```

```

END;
$$
LANGUAGE 'plpgsql' IMMUTABLE;

UPDATE dummy_rast SET map_rast = ST_MapAlgebraFct(rast,NULL,'mod_fct(float,integer[],text ←
  [])'::regprocedure) WHERE rid = 2;

SELECT ST_Value(rast,1,i,j) As origval, ST_Value(map_rast, 1, i, j) As mapval
FROM dummy_rast CROSS JOIN generate_series(1, 3) AS i CROSS JOIN generate_series(1,3) AS j
WHERE rid = 2;

```

| origval | mapval |
|---------|--------|
| 253     | 1      |
| 254     | 0      |
| 253     | 1      |
| 253     | 1      |
| 254     | 0      |
| 254     | 0      |
| 250     | 0      |
| 254     | 0      |
| 254     | 0      |

```

&#xc7ac;&#xbd84;&#xb958;&#xb97c; &#xac70;&#xce58;&#xace0; NODATA &#xac12;&#xc744; &#xc0ac;&#xc6a9;&#xc790;
&#xd568;&#xc218;(0)&#xc73c;&#xb85c; &#xb118;&#xaca8;&#xc9c4; &#xd30c;&#xb77c;&#xbbf8;&#xd130;&#xb85c; &#xc124;&#xc6d0;
&#xbcf8;&#xc73c;&#xb85c;&#xbd80;&#xd130; &#xd53d;&#xc140; &#xc720;&#xd615;&#xc774; 2BUI&#xc778;
&#xbc34;&#xb4dc; 1&#xac1c;&#xb97c; &#xac00;&#xc9c4; &#xc0c8; &#xb798;&#xc2a4;&#xd130;&#xb97c; &#xc0dd;&#xc131;&#xc7ac;

```

```

ALTER TABLE dummy_rast ADD COLUMN map_rast2 raster;
CREATE FUNCTION classify_fct(pixel float, pos integer[], variadic args text[])
RETURNS float
AS
$$
DECLARE
  nodata float := 0;
BEGIN
  IF NOT args[1] IS NULL THEN
    nodata := args[1];
  END IF;
  IF pixel < 251 THEN
    RETURN 1;
  ELSIF pixel = 252 THEN
    RETURN 2;
  ELSIF pixel > 252 THEN
    RETURN 3;
  ELSE
    RETURN nodata;
  END IF;
END;
$$
LANGUAGE 'plpgsql';
UPDATE dummy_rast SET map_rast2 = ST_MapAlgebraFct(rast,'2BUI','classify_fct(float,integer ←
  [],text[])'::regprocedure, '0') WHERE rid = 2;

SELECT DISTINCT ST_Value(rast,1,i,j) As origval, ST_Value(map_rast2, 1, i, j) As mapval
FROM dummy_rast CROSS JOIN generate_series(1, 5) AS i CROSS JOIN generate_series(1,5) AS j
WHERE rid = 2;

```

| origval | mapval |
|---------|--------|
| 249     | 1      |
| 250     | 1      |





```
PL/pgSQL
```

```
CREATE OR REPLACE FUNCTION simple_function_for_two_rasters(pixel1 FLOAT, pixel2 FLOAT, pos ←
    INTEGER[], VARIADIC args TEXT[])
RETURNS FLOAT
AS $$ BEGIN
RETURN 0.0;
END; $$
LANGUAGE 'plpgsql' IMMUTABLE;
```

```
tworastuserfunc
rast1
rast2
ST_MapAlgebraFct
```

```
regprocedure
SQL
```

```
'simple_function(double precision, double precision, integer[], text[])'::regprocedure
```

```
regprocedure
```

The fourth argument to the `tworastuserfunc` is a variadic text array. All trailing text arguments to any `ST_MapAlgebraFct` call are passed through to the specified `tworastuserfunc`, and are contained in the `userargs` argument.

#### Note



(`VARIADIC` `Query Language (SQL) Functions`, "SQL Functions with Variable Numbers of Arguments"

#### Note



`tworastuserfunc` `text[]`

2.0.0

**&#xc608;&#xc2dc;: &#xce94;&#xbc84;&#xc2a4;&#xc0c1;&#xc5d0; &#xb798;&#xc2a4;&#xd130;&#xb4e4;&#xc744; &#xac1c;&#&#xbc34;&#xb4dc;&#xb85c;&#xc11c; &#xc624;&#xbc84;&#xb808;&#xc774;**

```
-- &#xc0ac;&#xc6a9;&#xc790; &#xc9c0;&#xc815; &#xd568;&#xc218;&#xb97c; &#xc815;&#xc758; --
CREATE OR REPLACE FUNCTION raster_mapalgebra_union(
    rast1 double precision,
    rast2 double precision,
    pos integer[],
    VARIADIC userargs text[]
)
RETURNS double precision
AS $$
DECLARE
BEGIN
    CASE
        WHEN rast1 IS NOT NULL AND rast2 IS NOT NULL THEN
            RETURN ((rast1 + rast2)/2.);
        WHEN rast1 IS NULL AND rast2 IS NULL THEN
            RETURN NULL;
        WHEN rast1 IS NULL THEN
            RETURN rast2;
        ELSE
            RETURN rast1;
    END CASE;

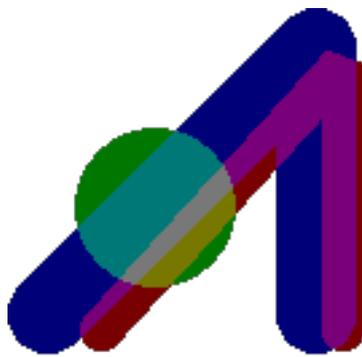
    RETURN NULL;
END;
$$ LANGUAGE 'plpgsql' IMMUTABLE COST 1000;

-- &#xb798;&#xc2a4;&#xd130; &#xd14c;&#xc2a4;&#xd2b8; &#xd14c;&#xc774;&#xbe14; &#xc900;&#xbe44;
DROP TABLE IF EXISTS map_shapes;
CREATE TABLE map_shapes(rid serial PRIMARY KEY, rast raster, bnum integer, descrip text);
INSERT INTO map_shapes(rast,bnum, descrip)
WITH mygeoms
    AS ( SELECT 2 As bnum, ST_Buffer(ST_Point(90,90),30) As geom, 'circle' As descrip
        UNION ALL
        SELECT 3 AS bnum,
            ST_Buffer(ST_GeomFromText('LINESTRING(50 50,150 150,150 50)'), 15) As geom, ←
            'big road' As descrip
        UNION ALL
        SELECT 1 As bnum,
            ST_Translate(ST_Buffer(ST_GeomFromText('LINESTRING(60 50,150 150,150 50)'), ←
            8,'join=bevel'), 10,-6) As geom, 'small road' As descrip
    ),
-- &#xce94;&#xbc84;&#xc2a4;&#xc0c0; &#xb3c4;&#xd615;&#xc5d0; 1:1 ←
    &#xd53d;&#xc140;&#xc774; &#xb418;&#xb3c4;&#xb85d; &#xc815;&#xc758;
canvas
    AS ( SELECT ST_AddBand(ST_MakeEmptyRaster(250,
        250,
        ST_XMin(e)::integer, ST_YMax(e)::integer, 1, -1, 0, 0 ) , '8BUI'::text,0) As rast
        FROM (SELECT ST_Extent(geom) As e,
            Max(ST_SRID(geom)) As srid
            from mygeoms
            ) As foo
    )
-- &#xce94;&#xbc84;&#xc2a4;&#xc5d0; &#xc815;&#xb82c;&#xb41c; ←
    &#xb798;&#xc2a4;&#xd130;&#xb97c; &#xbc18;&#xd658;
SELECT ST_AsRaster(m.geom, canvas.rast, '8BUI', 240) As rast, bnum, descrip
    FROM mygeoms AS m CROSS JOIN canvas
UNION ALL
SELECT canvas.rast, 4, 'canvas'
FROM canvas;
```

```

--
INSERT INTO map_shapes(rast,bnum,descrip)
SELECT ST_AddBand(ST_AddBand(rasts[1], rasts[2]),rasts[3]), 4, 'map bands overlay fct union
(canvas)'
FROM (SELECT ARRAY(SELECT ST_MapAlgebraFct(m1.rast, m2.rast,
'raster_mapalgebra_union(double precision, double precision,
integer[], text[])'::regprocedure, '8BUI', 'FIRST')
FROM map_shapes As m1 CROSS JOIN map_shapes As m2
WHERE m1.descrip = 'canvas' AND m2.descrip <> 'canvas' ORDER BY m2.bnum) As rasts)
As foo;

```



(R: small road, G: circle, B: big road)

CREATE OR REPLACE FUNCTION raster\_mapalgebra\_userargs (

```

    rast1 double precision,
    rast2 double precision,
    pos integer[],
    VARIADIC userargs text[]
)
RETURNS double precision
AS $$
DECLARE
BEGIN
    CASE
        WHEN rast1 IS NOT NULL AND rast2 IS NOT NULL THEN
            RETURN least(userargs[1]::integer, (rast1 + rast2)/2.);
        WHEN rast1 IS NULL AND rast2 IS NULL THEN

```

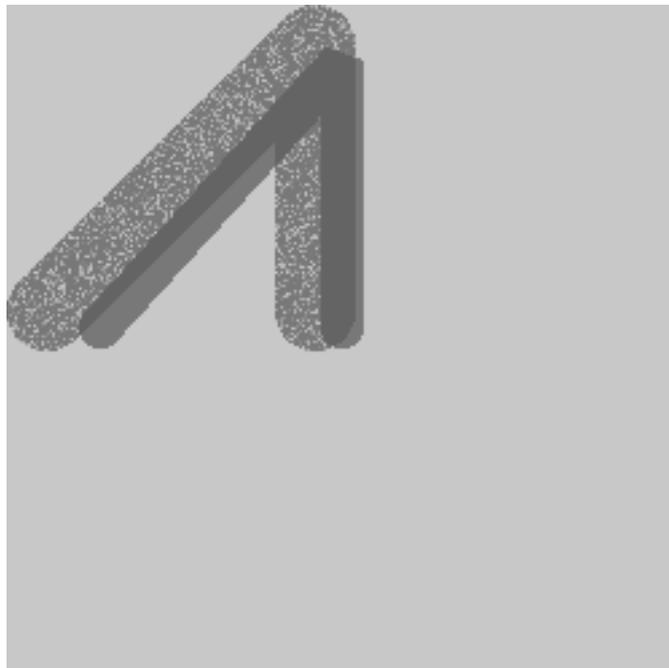
```

        RETURN userargs[2]::integer;
    WHEN rast1 IS NULL THEN
        RETURN greatest(rast2, random()*userargs[3]::integer)::integer;
    ELSE
        RETURN greatest(rast1, random()*userargs[4]::integer)::integer;
    END CASE;

    RETURN NULL;
END;
$$ LANGUAGE 'plpgsql' VOLATILE COST 1000;

SELECT ST_MapAlgebraFct(m1.rast, 1, m1.rast, 3,
    'raster_mapalgebra_userargs(double precision, double precision, integer[], text[])'::regprocedure,
    '8BUI', 'INTERSECT', '100', '200', '200', '0')
    FROM map_shapes As m1
    WHERE m1.descrip = 'map bands overlay fct union (canvas)';

```



[ST\\_MapAlgebraFct](#), [ST\\_MapAlgebraFctNgb](#), [ST\\_MapAlgebraExpr](#), [ST\\_BandPixelType](#), [ST\\_GeoReference](#), [ST\\_SetValue](#)

**ST\_MapAlgebraFctNgb**

[ST\\_MapAlgebraExpr](#), [ST\\_BandPixelType](#), [ST\\_GeoReference](#), [ST\\_SetValue](#)

### 12.12.11 ST\_MapAlgebraFctNgb

**ST\_MapAlgebraFctNgb** — [ST\\_MapAlgebraFctNgb](#) (Map Algebra Nearest Neighbor) neighborhood





```
FROM katrinas_rescaled
limit 1;
```



**&#xcc38;&#xace0;**

[ST\\_MapAlgebraFct](#), [ST\\_MapAlgebraExpr](#), [ST\\_Rescale](#)

### 12.12.12 ST\_Reclass

ST\_Reclass — &#xc6d0;&#xbcf8;&#xc73c;&#xb85c;&#xbd80;&#xd130; &#xc7ac;&#xbd84;&#xb958;&#xb41c; &#xbc34;&#xb4dc;&#xc720;&#xd615;&#xc73c;&#xb85c; &#xc774;&#xb8e8;&#xc5b4;&#xc9c4; &#xc0c8; &#xb798;&#xc2a4;&#xd130;&#xb97c; &#xc0dd;&#xc131;&#xd569;&#xb2c8;&#xb2e4;. nband &#xb294; &#xbcc0;&#xacbd;&#xd560; &#bc34;&#xb4dc;&#xb97c; &#xac00;&#xb9ac;&#xd0b5;&#xb2c8;&#xb2e4;. nband &#xb97c; &#xb530;&#xb85c; &#xc124;&#xc815;&#xd558;&#xc9c0; &#xc54a;&#xc744; &#xacbd;&#xc6b0; &#bc34;&#xb4dc; 1&#xb85c; &#xac00;&#xc815;&#xd569;&#xb2c8;&#xb2e4;. &#xb2e4;&#xbaa8;&#xb4e0; &#bc34;&#xb4dc;&#xb4e4;&#xc740; &#xbcc0;&#xacbd; &#xc5c6;&#xc774; &#bc18;&#xd658;&#xb429;&#xc2e4;&#xc81c; &#xc0ac;&#xb840;: &#xbcf4;&#xae30; &#xc88b;&#xc740; &#xd615;&#xc2dd;&#xc73c;&#xb85c; &#xb354; &#xac04;&#xb2e8;&#xd558;&#xac8c; &#xb80c;&#xb354;&#xb9c1;&#xd558;&#xae30; &#xc704;&#xd574; 16BUI &#bc34;&#xb8BUI &#xb4f1;&#xb4f1;&#xc73c;&#xb85c; &#xbcc0;&#xd658;&#xd558;&#xc2ed;&#xc2dc;&#xc624;

#### Synopsis

```
raster ST_Reclass(raster rast, integer nband, text reclassexpr, text pixeltype, double precision nodataval=NULL);
raster ST_Reclass(raster rast, reclassarg[] VARIADIC reclassargset);
raster ST_Reclass(raster rast, text reclassexpr, text pixeltype);
```

**&#xc124;&#xba85;**

&#xc785;&#xb825; &#xb798;&#xc2a4;&#xd130;(rast)&#xc5d0; &#xb300;&#xd574; reclassexpr &#xc774; &#xc815;&#xc758;&#xc720;&#xd6a8;&#xd55c; PostgreSQL &#xb300;&#xc218; &#xc5f0;&#xc0b0;&#xc744; &#xc801;&#xc6a9;&#xd574;&#xc11c;

```

band &#xb97c; &#xc124;&#xc815;&#xd558;&#xc9c0; &#xc54a;&#xc744; &#xacbd;&#xc6b0;, &#xbc34;&#xb4dc; 1&#xb85c;
&#xac00;&#xc815;&#xd569;&#xb2c8;&#xb2e4;. &#xc774; &#xc0c8; &#xb798;&#xc2a4;&#xd130;&#xb294; &#xc6d0;&#xbcf8;
&#xb798;&#xc2a4;&#xd130;&#xc640; &#xb3d9;&#xc77c;&#xd55c; &#xc9c0;&#xb9ac;&#xc38;&#xc870;, &#xb108;&#xbe44;
&#xbc0f;&#xb192;&#xc774;&#xb97c; &#xac00;&#xc9c8; &#xac83;&#xc785;&#xb2c8;&#xb2e4;. &#xc9c0;&#xc815;&#xd558;&#xc
&#xc54a;&#xc740; &#xbc34;&#xb4dc;&#xb4e4;&#xc740; &#xbcc0;&#xacbd; &#xc5c6;&#xc774; &#xbc18;&#xd658;&#xb420;
&#xac83;&#xc785;&#xb2c8;&#xb2e4;. &#xc720;&#xd6a8;&#xd55c; &#xc7ac;&#xbd84;&#xb958; &#xd45c;&#xd604;&#xc2dd;&#
&#xb300;&#xd55c; &#xc124;&#xba85;&#xc774; &#xd544;&#xc694;&#xd55c; &#xacbd;&#xc6b0; reclassarg &#xb97c; &#xc38;&#xc
&#xc0c8; &#xb798;&#xc2a4;&#xd130;&#xc758; &#xbc34;&#xb4dc;&#xb4e4;&#xc740; pixeltype &#xd53d;&#xc140;
&#xc720;&#xd615;&#xc77c; &#xac83;&#xc785;&#xb2c8;&#xb2e4;. reclassargset &#xb97c; &#xb118;&#xaca8;&#xbcb1b;&#
&#xacbd;&#xc6b0;, &#xac01; reclassarg(&#xc7ac;&#xbd84;&#xb958; &#xc778;&#xc218;)&#xac00; &#xc0dd;&#xc131;&#xb41c;
&#xac01; &#bc34;&#xb4dc;&#xc758; &#xc2b5;&#xc131;&#xc744; &#xc815;&#xc758;&#xd569;&#xb2c8;&#xb2e4;.
2.0.0 &#xbc84;&#xc804;&#xbd80;&#xd130; &#xc0ac;&#xc6a9;&#xd560; &#xc218; &#xc788;&#xc2b5;&#xb2c8;&#xb2e4;.

```

### &#xae30;&#xbcf8; &#xc608;&#xc2dc;

```

&#xc6d0;&#xbcf8;&#xc758; &#xbc34;&#xb4dc; 2&#xb97c; 8BUI&#xc5d0;&#xc11c; 4BUI&#xb85c; &#xbcc0;&#xd658;&#xd558;
101&#xc5d0;&#xc11c; 254&#xae4c;&#xc9c0;&#xc758; &#xbaa8;&#xb4e0; &#xac12;&#xc744; NODATA &#xac12;&#xc73c;&#xb
&#xc124;&#xc815;&#xd55c; &#xc0c8; &#xb798;&#xc2a4;&#xd130;&#xb97c; &#xc0dd;&#xc131;&#xd569;&#xb2c8;&#xb2e4;.

```

```

ALTER TABLE dummy_rast ADD COLUMN reclass_rast raster;
UPDATE dummy_rast SET reclass_rast = ST_Reclass(rast,2,'0-87:1-10, 88-100:11-15, ←
101-254:0-0', '4BUI',0) WHERE rid = 2;

SELECT i as col, j as row, ST_Value(rast,2,i,j) As origval,
       ST_Value(reclass_rast, 2, i, j) As reclassval,
       ST_Value(reclass_rast, 2, i, j, false) As reclassval_include_nodata
FROM dummy_rast CROSS JOIN generate_series(1, 3) AS i CROSS JOIN generate_series(1,3) AS j
WHERE rid = 2;

```

| col | row | origval | reclassval | reclassval_include_nodata |
|-----|-----|---------|------------|---------------------------|
| 1   | 1   | 78      | 9          | 9                         |
| 2   | 1   | 98      | 14         | 14                        |
| 3   | 1   | 122     |            | 0                         |
| 1   | 2   | 96      | 14         | 14                        |
| 2   | 2   | 118     |            | 0                         |
| 3   | 2   | 180     |            | 0                         |
| 1   | 3   | 99      | 15         | 15                        |
| 2   | 3   | 112     |            | 0                         |
| 3   | 3   | 169     |            | 0                         |

### &#xc7ac;&#xbd84;&#xb958; &#xc778;&#xc218;&#xb97c; &#xc5ec;&#xb7ec; &#xac1c; &#xc774;&#xc6a9;&#xd558;&#xb294; &#xace0;&#xae09; &#xc608;&#xc2dc;

```

&#xc6d0;&#xbcf8; &#xb798;&#xc2a4;&#xd130;&#xc758; &#xbc34;&#xb4dc; 1, 2, 3&#xc744; &#xac01;&#xac01; 1BB,
4BUI, 4BUI&#xb85c; &#xbcc0;&#xd658;&#xd558;&#xace0; &#xc7ac;&#xbd84;&#xb958;&#xd55c; &#xc0c8; &#xb798;&#xc2a4;&#
&#xc0dd;&#xc131;&#xd569;&#xb2c8;&#xb2e4;. &#xc774; &#xc608;&#xc2dc;&#xac00; (&#xc774;&#xb860;&#xc801;&#xc73c;&#
&#xc0ac;&#xc6a9;&#xc790;&#xac00; &#xac00;&#xc9c4; &#xbc34;&#xb4dc;&#xc758; &#xac1c;&#xc218;&#xb9cc;&#xd07c;)
&#xbb34;&#xd55c;&#xd55c; &#xac1c;&#xc218;&#xc758; &#xc7ac;&#xbd84;&#xb958; &#xc778;&#xc218;&#xb97c; &#xc785;&#
&#xc218; &#xc788;&#xb294;, &#xb2e4;&#xc591;&#xd55c; &#xac1c;&#xc218;&#xc758; &#xc778;&#xc218;&#xb97c; &#xc785;&#
reclassarg &#xc778;&#xc218;&#xb97c; &#xc774;&#xc6a9;&#xd55c;&#xb2e4;&#xb294; &#xc810;&#xc5d0; &#xc8fc;&#xc758;

```

```

UPDATE dummy_rast SET reclass_rast =
  ST_Reclass(rast,
    ROW(2,'0-87]:1-10, (87-100]:11-15, (101-254]:0-0', '4BUI',NULL)::reclassarg,
    ROW(1,'0-253]:1, 254:0', '1BB', NULL)::reclassarg,

```

```
ROW(3, '0-70]:1, (70-86:2, [86-150]:3, [150-255:4', '4BUI', NULL)::reclassarg
) WHERE rid = 2;
```

```
SELECT i as col, j as row, ST_Value(rast,1,i,j) As ov1, ST_Value(reclass_rast, 1, i, j) As ←
rv1,
ST_Value(rast,2,i,j) As ov2, ST_Value(reclass_rast, 2, i, j) As rv2,
ST_Value(rast,3,i,j) As ov3, ST_Value(reclass_rast, 3, i, j) As rv3
FROM dummy_rast CROSS JOIN generate_series(1, 3) AS i CROSS JOIN generate_series(1,3) AS j
WHERE rid = 2;
```

| col | row | ov1 | rv1 | ov2 | rv2 | ov3 | rv3 |
|-----|-----|-----|-----|-----|-----|-----|-----|
| 1   | 1   | 253 | 1   | 78  | 9   | 70  | 1   |
| 2   | 1   | 254 | 0   | 98  | 14  | 86  | 3   |
| 3   | 1   | 253 | 1   | 122 | 0   | 100 | 3   |
| 1   | 2   | 253 | 1   | 96  | 14  | 80  | 2   |
| 2   | 2   | 254 | 0   | 118 | 0   | 108 | 3   |
| 3   | 2   | 254 | 0   | 180 | 0   | 162 | 4   |
| 1   | 3   | 250 | 1   | 99  | 15  | 90  | 3   |
| 2   | 3   | 254 | 0   | 112 | 0   | 108 | 3   |
| 3   | 3   | 254 | 0   | 169 | 0   | 175 | 4   |

```
32BF &#x34; &#x4d; &#x9c; &#x00; &#x9c; &#x798; &#x2a; &#xd130; &#x97c; &#x34; &#xae3
&#x88b; &#x740; &#x34; &#x4d; &#x5ec; &#x7ec; &#xac1c; &#xc5d0; &#xb9e4; &#xd551; &#xd558; &#xb294; &#xace0; &#
&#xc608; &#xc2dc;
```

```
32BF &#x34; &#x4d; &#xd558; &#xb098; &#xb9cc; &#xac00; &#xc9c4; &#xb798; &#xc2a4; &#xd130; &#xb85c; &#xbd80; &#xd1
&#xc0c8; &#xb85c; &#xc6b4; &#x34; &#x4d; 3&#xac1c; ((8BUI,8BUI,8BUI) &#xb97c; &#xac00; &#xc9c4; &#x34; &#xae30;
&#xc88b; &#x740; &#xc0c8; &#xb798; &#xc2a4; &#xd130; &#xb97c; &#xc0dd; &#xc131; &#xd569; &#xb2c8; &#xb2e4;.
```

```
ALTER TABLE wind ADD COLUMN rast_view raster;
UPDATE wind
set rast_view = ST_AddBand( NULL,
ARRAY[
ST_Reclass(rast, 1, '0.1-10]:1-10, 9-10]:11, (11-33:0'::text, '8BUI'::text, 0),
ST_Reclass(rast, 1, '11-33):0-255, [0-32:0, (34-1000:0'::text, '8BUI'::text, 0),
ST_Reclass(rast, 1, '0-32]:0, (32-100:100-255'::text, '8BUI'::text, 0)
]
);
```

```
&#xcc38; &#xace0;
```

[ST\\_AddBand](#), [ST\\_Band](#), [ST\\_BandPixelType](#), [ST\\_MakeEmptyRaster](#), [reclassarg](#), [ST\\_Value](#)

### 12.12.13 ST\_Union

**ST\_Union** — &#xb798; &#xc2a4; &#xd130; &#xd0c0; &#xc77c; &#xc9d1; &#xd569; &#xc744; 1&#xac1c; &#xc774; &#xc0c1; &#xc75
&#x34; &#x4d; &#xb85c; &#xc774; &#xb8e8; &#xc5b4; &#xc9c4; &#xb2e8; &#xc77c; &#xb798; &#xc2a4; &#xd130; &#xb85c;
&#xd1b5; &#xd569; &#xd569; &#xb2c8; &#xb2e4;.

#### Synopsis

```
bytea ST_AsJPEG(raster rast, text[] options=NULL);
bytea ST_AsJPEG(raster rast, integer nband, integer quality);
bytea ST_AsJPEG(raster rast, integer nband, text[] options=NULL);
bytea ST_AsJPEG(raster rast, integer[] nbands, text[] options=NULL);
bytea ST_AsJPEG(raster rast, integer[] nbands, integer quality);
```



```
WHERE ST_Intersects(rast, ST_GeomFromText('LINESTRING(230486 887771, 230500 88772)',26986) ←
);
```

```
&#xc608;&#xc2dc;: &#xd0c0;&#xc77c;&#xc774; &#xb3c4;&#xd615;&#xacfc; &#xad50;&#xcc28;&#xd558;&#xb294; &#xbd80;&#&#xd1b5;&#xd569;&#xd55c; &#xb2e4;&#xc911; &#xbc34;&#xb4dc; &#xb798;&#xc2a4;&#xd130;&#xb97c; &#bc18;&#xd658;
```

```
&#xb2e4;&#xc74c;&#xc740; &#bc34;&#xb4dc;&#xb4e4;&#xc758; &#xd558;&#xc704; &#xc9d1;&#xd569;&#xb9cc;&#xc744;
&#xc6d0;&#xd558;&#xac70;&#xb098;,&#xb610;&#xb294; &#bc34;&#xb4dc;&#xb4e4;&#xc758; &#xc21c;&#xc11c;&#xb97c;
&#xbcc0;&#xacbd;&#xd558;&#xace0;&#xc790; &#xd558;&#xb294; &#xacbd;&#xc6b0; &#xb354; &#xae34; &#bb38;&#bc95;&#&#xc0ac;&#xc6a9;&#xd558;&#xb294; &#xc608;&#xc2dc;&#xc785;&#xb2c8;&#xb2e4;.
```

```
-- &#xb77c;&#xc778;&#xacfc; &#xad50;&#xcc28;&#xd558;&#xb294; &#baa8;&#xb4e0; ←
&#xd0c0;&#xc77c;&#xb4e4;&#xc744; &#baa8;&#xc740; &#xb2e4;&#xc911; &#bc34;&#xb4dc; ←
&#xb798;&#xc2a4;&#xd130;&#xb97c; &#xc0dd;&#xc131;&#xd569;&#xb2c8;&#xb2e4; .
SELECT ST_Union(rast,ARRAY[ROW(2, 'LAST'), ROW(1, 'LAST'), ROW(3, 'LAST')]:unionarg[])
FROM aerials.boston
WHERE ST_Intersects(rast, ST_GeomFromText('LINESTRING(230486 887771, 230500 88772)',26986) ←
);
```

&#xcc38;&#xace0;

[unionarg](#), [ST\\_Envelope](#), [ST\\_ConvexHull](#), [ST\\_Clip](#), [ST\\_Union](#)

## 12.13 &#xb0b4;&#xc7a5; &#xb9f5; &#xb300;&#xc218; &#xcf5c;&#xbc31; &#xd568;&#xc218;

### 12.13.1 ST\_Distinct4ma

ST\_Distinct4ma — &#xc774;&#xc6c3;&#xc5d0;&#xc11c; &#xc720;&#xc77c;&#xd55c; &#xd53d;&#xc140; &#xac12;&#xb4e4;&#&#xac1c;&#xc218;&#xb97c; &#xacc4;&#xc0b0;&#xd558;&#xb294; &#xb798;&#xc2a4;&#xd130; &#xacf5;&#xac04; &#xcc98;&#&#xd568;&#xc218;&#xc785;&#xb2c8;&#xb2e4;.

#### Synopsis

float8 ST\_Distinct4ma(float8[][] matrix, text nodatamode, text[] VARIADIC args);

double precision ST\_Distinct4ma(double precision[][][] value, integer[][] pos, text[] VARIADIC userargs);

&#xc124;&#xba85;

&#xd53d;&#xc140;&#xc758; &#xc774;&#xc6c3;&#xc5d0; &#xc788;&#xb294; &#xc720;&#xc77c;&#xd55c; &#xd53d;&#xc140;
&#xac12;&#xb4e4;&#xc758; &#xac1c;&#xc218;&#xb97c; &#xacc4;&#xc0b0;&#xd569;&#xb2c8;&#xb2e4;.



#### Note

&#xbcc0;&#xc885; 1&#xc740; ST\_MapAlgebraFctNgb &#xc758; &#xcf5c;&#xbc31;
&#xd30c;&#xb77c;&#xbbf8;&#xd130;&#xb85c; &#xc4f0;&#xc774;&#xb294; &#xd2b9;&#xd654;&#xb41c;
&#xcf5c;&#xbc31; &#xd568;&#xc218;&#xc785;&#xb2c8;&#xb2e4;.



#### Note

&#xbcc0;&#xc885; 2&#xb294; &#xb0b4;&#xc7a5; &#xb9f5; &#xb300;&#xc218; &#xcf5c;&#xbc31; &#xd568;&#xc218;
&#xc758; &#xcf5c;&#xbc31; &#xd30c;&#xb77c;&#xbbf8;&#xd130;&#xb85c; &#xc4f0;&#xc774;&#xb294;
&#xd2b9;&#xd654;&#xb41c; &#xcf5c;&#xbc31; &#xd568;&#xc218;&#xc785;&#xb2c8;&#xb2e4;.



$\hat{z}(x_0) = \frac{\sum_{j=1}^m z(x_j) d_{ij}^{-k}}{\sum_{j=1}^m d_{ij}^{-k}}$

$k =$  (power factor),  $0 < k < 1$

$$\hat{z}(x_0) = \frac{\sum_{j=1}^m z(x_j) d_{ij}^{-k}}{\sum_{j=1}^m d_{ij}^{-k}}$$

$k =$  (power factor),  $0 < k < 1$



**Note**

The function `ST_MinDist4ma` is implemented as a SQL function. It takes a point and a set of points as input and returns the distance from the point to the nearest point in the set. The function is implemented as a SQL function.

2.1.0 `ST_MinDist4ma` (point, geometry) returns float8

`ST_MinDist4ma`

```
-- ST_MinDist4ma(point, geometry) returns float8
```

`ST_MinDist4ma`

`ST_MinDist4ma` (point, geometry) returns float8

### 12.13.3 ST\_Max4ma

`ST_Max4ma` — (float8[][] matrix, text nodatamode, text[] VARIADIC args);

**Synopsis**

float8 `ST_Max4ma`(float8[][] matrix, text nodatamode, text[] VARIADIC args);  
 double precision `ST_Max4ma`(double precision[][] value, integer[][] pos, text[] VARIADIC userargs);

`ST_Max4ma`

`ST_Max4ma` (float8[][] matrix, text nodatamode, text[] VARIADIC args);

`ST_Max4ma` (double precision[][] value, integer[][] pos, text[] VARIADIC userargs);



**Note**

The function `ST_Max4ma` is implemented as a SQL function. It takes a matrix and a set of points as input and returns the maximum value from the matrix. The function is implemented as a SQL function.



**Note**

`ST_MapAlgebraFctNgb` (2D) is a function that takes a raster and a text value and returns a raster of the same type and resolution. The text value is used to specify the function to be applied to each cell. The function is applied to the cell value and the text value. The function is applied to the cell value and the text value. The function is applied to the cell value and the text value.

**Note**

`ST_MapAlgebraFctNgb` (2D) is a function that takes a raster and a text value and returns a raster of the same type and resolution. The text value is used to specify the function to be applied to each cell. The function is applied to the cell value and the text value. The function is applied to the cell value and the text value.

**Warning**

2.1.0 `ST_MapAlgebraFctNgb` (2D) is a function that takes a raster and a text value and returns a raster of the same type and resolution. The text value is used to specify the function to be applied to each cell. The function is applied to the cell value and the text value. The function is applied to the cell value and the text value.

2.0.0 `ST_MapAlgebraFctNgb` (2D) is a function that takes a raster and a text value and returns a raster of the same type and resolution. The text value is used to specify the function to be applied to each cell. The function is applied to the cell value and the text value. The function is applied to the cell value and the text value.

`ST_MapAlgebraFctNgb` (2D) is a function that takes a raster and a text value and returns a raster of the same type and resolution. The text value is used to specify the function to be applied to each cell. The function is applied to the cell value and the text value. The function is applied to the cell value and the text value.

**Example 1**

```
SELECT
  rid,
  st_value(
    st_mapalgebrafctngb(rast, 1, '32BF', 1, 1, 'st_mean4ma(float[],text,text)')::
    regprocedure, 'ignore', NULL), 2, 2
  )
FROM dummy_rast
WHERE rid = 2;
rid |      st_value
-----+-----
  2 | 253.222229003906
(1 row)
```

**Example 2**

```
SELECT
  rid,
  st_value(
    ST_MapAlgebra(rast, 1, 'st_mean4ma(double precision[][][], integer[], text ←
    [])'::regprocedure, '32BF', 'FIRST', NULL, 1, 1)
    , 2, 2)
FROM dummy_rast
WHERE rid = 2;
rid |      st_value
-----+-----
  2 | 253.222229003906
(1 row)
```

**Example 3**

`ST_MapAlgebraFctNgb` (2D) is a function that takes a raster and a text value and returns a raster of the same type and resolution. The text value is used to specify the function to be applied to each cell. The function is applied to the cell value and the text value. The function is applied to the cell value and the text value.

### 12.13.5 ST\_Min4ma

ST\_Min4ma — `float8 ST_Min4ma(float8[][] matrix, text nodatamode, text[] VARIADIC args);`  
`double precision ST_Min4ma(double precision[][][] value, integer[][] pos, text[] VARIADIC userargs);`

#### Synopsis

`float8 ST_Min4ma(float8[][] matrix, text nodatamode, text[] VARIADIC args);`  
`double precision ST_Min4ma(double precision[][][] value, integer[][] pos, text[] VARIADIC userargs);`

#### Notes

ST\_Min4ma is a `ST_Min` function that takes a 2D array of float8 values and returns the minimum value. It is similar to `ST_Min` but takes a 2D array of float8 values instead of a single float8 value.

ST\_Min4ma is a `ST_Min` function that takes a 3D array of double precision values and returns the minimum value. It is similar to `ST_Min` but takes a 3D array of double precision values instead of a single double precision value.



#### Note

ST\_Min4ma is a `ST_Min` function that takes a 2D array of float8 values and returns the minimum value. It is similar to `ST_Min` but takes a 2D array of float8 values instead of a single float8 value.



#### Note

ST\_Min4ma is a `ST_Min` function that takes a 2D array of float8 values and returns the minimum value. It is similar to `ST_Min` but takes a 2D array of float8 values instead of a single float8 value.



#### Warning

ST\_Min4ma is a `ST_Min` function that takes a 2D array of float8 values and returns the minimum value. It is similar to `ST_Min` but takes a 2D array of float8 values instead of a single float8 value.

ST\_Min4ma is a `ST_Min` function that takes a 2D array of float8 values and returns the minimum value. It is similar to `ST_Min` but takes a 2D array of float8 values instead of a single float8 value.

#### Example

```
SELECT
  rid,
  st_value(
    st_mapalgebrafctngb(rast, 1, NULL, 1, 1, 'st_min4ma(float8[][],text,text[])':: ↵
    regprocedure, 'ignore', NULL), 2, 2
  )
FROM dummy_rast
WHERE rid = 2;
  rid | st_value
-----+-----
    2 |      250
(1 row)
```



**Synopsis**

float8 **ST\_Range4ma**(float8[][] matrix, text nodatamode, text[] VARIADIC args);  
 double precision **ST\_Range4ma**(double precision[][][] value, integer[][] pos, text[] VARIADIC userargs);

**Parameters**

`matrix`: 2D array of float8 values.  
`nodatamode`: 'NODATA' or 'NULL'.  
`args`: array of text values.  
`value`: 3D array of double precision values.  
`pos`: 2D array of integer values.  
`userargs`: array of text values.



**Note**

ST\_MapAlgebraFctNgb, ST\_Min4ma, ST\_Max4ma, ST\_Sum4ma, ST\_Mean4ma, ST\_Distinct4ma, ST\_StdDev4ma



**Note**

ST\_MapAlgebraFctNgb, ST\_Min4ma, ST\_Max4ma, ST\_Sum4ma, ST\_Mean4ma, ST\_Distinct4ma, ST\_StdDev4ma



**Warning**

ST\_MapAlgebraFctNgb, ST\_Min4ma, ST\_Max4ma, ST\_Sum4ma, ST\_Mean4ma, ST\_Distinct4ma, ST\_StdDev4ma

2.0.0 ST\_MapAlgebraFctNgb, ST\_Min4ma, ST\_Max4ma, ST\_Sum4ma, ST\_Mean4ma, ST\_Distinct4ma, ST\_StdDev4ma

**Examples**

```
SELECT
  rid,
  st_value(
    st_mapalgebrafctngb(rast, 1, NULL, 1, 1, 'st_range4ma(float[],text,text[])'::
      regprocedure, 'ignore', NULL), 2, 2
  )
FROM dummy_rast
WHERE rid = 2;
  rid | st_value
-----+-----
    2 |      4
(1 row)
```

**See also**

ST\_MapAlgebraFctNgb, ST\_Min4ma, ST\_Max4ma, ST\_Sum4ma, ST\_Mean4ma, ST\_Distinct4ma, ST\_StdDev4ma

### 12.13.8 ST\_StdDev4ma

ST\_StdDev4ma — Returns the standard deviation of the values in the raster. The standard deviation is calculated for each pixel by averaging the values of the four nearest pixels (including the pixel itself) and then taking the square root of the variance. The standard deviation is calculated for each pixel by averaging the values of the four nearest pixels (including the pixel itself) and then taking the square root of the variance.

#### Synopsis

float8 ST\_StdDev4ma(float8[][] matrix, text nodatamode, text[] VARIADIC args);  
 double precision ST\_StdDev4ma(double precision[][] value, integer[][] pos, text[] VARIADIC userargs);

#### Parameters

matrix: A 2D array of float8 values representing the raster data.  
 nodatamode: A text string indicating the nodatamode parameter.  
 args: A list of text strings representing the variadic arguments.



#### Note

The standard deviation is calculated for each pixel by averaging the values of the four nearest pixels (including the pixel itself) and then taking the square root of the variance.



#### Note

The standard deviation is calculated for each pixel by averaging the values of the four nearest pixels (including the pixel itself) and then taking the square root of the variance.



#### Warning

2.1.0 The standard deviation is calculated for each pixel by averaging the values of the four nearest pixels (including the pixel itself) and then taking the square root of the variance.

2.0.0 The standard deviation is calculated for each pixel by averaging the values of the four nearest pixels (including the pixel itself) and then taking the square root of the variance.

#### Examples

```
SELECT
  rid,
  st_value(
    st_mapalgebrafctngb(rast, 1, '32BF', 1, 1, 'st_stddev4ma(float[],text,text[])'::
      regprocedure, 'ignore', NULL), 2, 2
  )
FROM dummy_rast
WHERE rid = 2;
  rid |      st_value
-----+-----
    2 | 1.30170822143555
(1 row)
```

**ST\_Min4ma**

**ST\_Max4ma**, **ST\_Sum4ma**, **ST\_Mean4ma**, **ST\_Distinct4ma**, **ST\_StdDev4ma**

### 12.13.9 ST\_Sum4ma

**ST\_Sum4ma** — **ST\_Sum4ma**(float8[][] matrix, text nodatamode, text[] VARIADIC args);  
 double precision **ST\_Sum4ma**(double precision[][][] value, integer[][] pos, text[] VARIADIC userargs);

#### Synopsis

float8 **ST\_Sum4ma**(float8[][] matrix, text nodatamode, text[] VARIADIC args);  
 double precision **ST\_Sum4ma**(double precision[][][] value, integer[][] pos, text[] VARIADIC userargs);

**ST\_Sum4ma**

**ST\_Sum4ma**(float8[][] matrix, text nodatamode, text[] VARIADIC args);  
 double precision **ST\_Sum4ma**(double precision[][][] value, integer[][] pos, text[] VARIADIC userargs);

**ST\_Sum4ma**(double precision[][][] value, integer[][] pos, text[] VARIADIC userargs);



#### Note

**ST\_Sum4ma**(float8[][] matrix, text nodatamode, text[] VARIADIC args);  
 double precision **ST\_Sum4ma**(double precision[][][] value, integer[][] pos, text[] VARIADIC userargs);



#### Note

**ST\_Sum4ma**(float8[][] matrix, text nodatamode, text[] VARIADIC args);  
 double precision **ST\_Sum4ma**(double precision[][][] value, integer[][] pos, text[] VARIADIC userargs);



#### Warning

**ST\_Sum4ma**(float8[][] matrix, text nodatamode, text[] VARIADIC args);  
 double precision **ST\_Sum4ma**(double precision[][][] value, integer[][] pos, text[] VARIADIC userargs);

**ST\_Sum4ma**(float8[][] matrix, text nodatamode, text[] VARIADIC args);  
 double precision **ST\_Sum4ma**(double precision[][][] value, integer[][] pos, text[] VARIADIC userargs);

**ST\_Sum4ma**(double precision[][][] value, integer[][] pos, text[] VARIADIC userargs);

**ST\_Sum4ma**

```
SELECT
  rid,
  st_value(
    st_mapalgebrafctngb(rast, 1, '32BF', 1, 1, 'st_sum4ma(float[][],text,text[])'::
      regprocedure, 'ignore', NULL), 2, 2
  )
```

```
FROM dummy_rast
WHERE rid = 2;
  rid | st_value
-----+-----
    2 |      2279
(1 row)
```

**ST\_MapAlgebraFctNgb**, **ST\_Min4ma**, **ST\_Max4ma**, **ST\_Mean4ma**, **ST\_Range4ma**, **ST\_Distinct4ma**, **ST\_StdDev4ma**

**ST\_MapAlgebraFctNgb**, **ST\_Min4ma**, **ST\_Max4ma**, **ST\_Mean4ma**, **ST\_Range4ma**, **ST\_Distinct4ma**, **ST\_StdDev4ma**

## 12.14 ST\_Aspect

### 12.14.1 ST\_Aspect

**ST\_Aspect** — Returns the aspect (slope and aspect) of a raster. The aspect is calculated as the angle of the slope in degrees, measured clockwise from North. The aspect is returned in degrees, ranging from 0 to 360. The aspect is calculated as the angle of the slope in degrees, measured clockwise from North. The aspect is returned in degrees, ranging from 0 to 360.

#### Synopsis

raster **ST\_Aspect**(raster rast, integer band=1, text pixeltype=32BF, text units=DEGREES, boolean interpolate\_nodata=FALSE);  
 raster **ST\_Aspect**(raster rast, integer band, raster customextent, text pixeltype=32BF, text units=DEGREES, boolean interpolate\_nodata=FALSE);

**ST\_Aspect**

**ST\_Aspect**(rast, band, pixeltype, units, interpolate\_nodata)  
 Returns the aspect (slope and aspect) of a raster. The aspect is calculated as the angle of the slope in degrees, measured clockwise from North. The aspect is returned in degrees, ranging from 0 to 360. The aspect is calculated as the angle of the slope in degrees, measured clockwise from North. The aspect is returned in degrees, ranging from 0 to 360.

**ST\_Aspect**(rast, band, customextent, pixeltype, units, interpolate\_nodata)  
 Returns the aspect (slope and aspect) of a raster. The aspect is calculated as the angle of the slope in degrees, measured clockwise from North. The aspect is returned in degrees, ranging from 0 to 360. The aspect is calculated as the angle of the slope in degrees, measured clockwise from North. The aspect is returned in degrees, ranging from 0 to 360.



#### Note

**ST\_Aspect**(rast, band, pixeltype, units, interpolate\_nodata)  
 Returns the aspect (slope and aspect) of a raster. The aspect is calculated as the angle of the slope in degrees, measured clockwise from North. The aspect is returned in degrees, ranging from 0 to 360. The aspect is calculated as the angle of the slope in degrees, measured clockwise from North. The aspect is returned in degrees, ranging from 0 to 360.

2.0.0 **ST\_Aspect**(rast, band, pixeltype, units, interpolate\_nodata)  
 Returns the aspect (slope and aspect) of a raster. The aspect is calculated as the angle of the slope in degrees, measured clockwise from North. The aspect is returned in degrees, ranging from 0 to 360. The aspect is calculated as the angle of the slope in degrees, measured clockwise from North. The aspect is returned in degrees, ranging from 0 to 360.

**ST\_Aspect**(rast, band, customextent, pixeltype, units, interpolate\_nodata)  
 Returns the aspect (slope and aspect) of a raster. The aspect is calculated as the angle of the slope in degrees, measured clockwise from North. The aspect is returned in degrees, ranging from 0 to 360. The aspect is calculated as the angle of the slope in degrees, measured clockwise from North. The aspect is returned in degrees, ranging from 0 to 360.

2.1.0  
 2.1.0  
 2.1.0

### Example 1

```
WITH foo AS (
  SELECT ST_SetValues(
    ST_AddBand(ST_MakeEmptyRaster(5, 5, 0, 0, 1, -1, 0, 0, 0), 1, '32BF', 0, -9999),
    1, 1, 1, ARRAY[
      [1, 1, 1, 1, 1],
      [1, 2, 2, 2, 1],
      [1, 2, 3, 2, 1],
      [1, 2, 2, 2, 1],
      [1, 1, 1, 1, 1]
    ]::double precision[]
  ) AS rast
)
SELECT
  ST_DumpValues(ST_Aspect(rast, 1, '32BF'))
FROM foo
```

---

```
(1, "{315,341.565063476562,0,18.4349479675293,45},{288.434936523438,315,0,45,71.5650482177734},{270.2227,180,161.565048217773,135}}")
(1 row)
```

### Example 2

PostgreSQL 9.1

```
WITH foo AS (
  SELECT ST_Tile(
    ST_SetValues(
      ST_AddBand(
        ST_MakeEmptyRaster(6, 6, 0, 0, 1, -1, 0, 0, 0),
        1, '32BF', 0, -9999
      ),
      1, 1, 1, ARRAY[
        [1, 1, 1, 1, 1, 1],
        [1, 1, 1, 1, 2, 1],
        [1, 2, 2, 3, 3, 1],
        [1, 1, 3, 2, 1, 1],
        [1, 2, 2, 1, 2, 1],
        [1, 1, 1, 1, 1, 1]
      ]::double precision[]
    ),
    2, 2
  ) AS rast
```

```

)
SELECT
    t1.rast,
    ST_Aspect(ST_Union(t2.rast), 1, t1.rast)
FROM foo t1
CROSS JOIN foo t2
WHERE ST_Intersects(t1.rast, t2.rast)
GROUP BY t1.rast;

```

**ST\_Azimuth**

**ST\_Azimuth(rast, geom)** — Returns the azimuth of the line segment connecting the center of the raster to the center of the geometry. The azimuth is measured in degrees clockwise from North. See [ST\\_Azimuth](#), [ST\\_Tri](#), [ST\\_TPI](#), [ST\\_Roughness](#), [ST\\_HillShade](#), [ST\\_Slope](#)

## 12.14.2 ST\_HillShade

**ST\_HillShade** — Returns a raster with the hillshade of the input raster. The hillshade is calculated using the azimuth and altitude of the sun. The azimuth is measured in degrees clockwise from North. The altitude is measured in degrees from the horizon. The hillshade is calculated using the formula:  $hillshade = 255 * (1 - \cos(\text{azimuth} - \text{azimuth}_{sun})) * \sin(\text{altitude}_{sun})$ . See [ST\\_HillShade](#), [ST\\_Slope](#), [ST\\_Azimuth](#), [ST\\_Tri](#), [ST\\_TPI](#), [ST\\_Roughness](#), [ST\\_HillShade](#), [ST\\_Slope](#)

### Synopsis

raster **ST\_HillShade**(raster rast, integer band=1, text pixeltype=32BF, double precision azimuth=315, double precision altitude=45, double precision max\_bright=255, double precision scale=1.0, boolean interpolate\_nodata=FALSE);

raster **ST\_HillShade**(raster rast, integer band, raster customextent, text pixeltype=32BF, double precision azimuth=315, double precision altitude=45, double precision max\_bright=255, double precision scale=1.0, boolean interpolate\_nodata=FALSE);

**ST\_HillShade**

**ST\_HillShade**(raster rast, integer band=1, text pixeltype=32BF, double precision azimuth=315, double precision altitude=45, double precision max\_bright=255, double precision scale=1.0, boolean interpolate\_nodata=FALSE);

**ST\_HillShade**(raster rast, integer band, raster customextent, text pixeltype=32BF, double precision azimuth=315, double precision altitude=45, double precision max\_bright=255, double precision scale=1.0, boolean interpolate\_nodata=FALSE);

**azimuth** — The azimuth of the sun. The azimuth is measured in degrees clockwise from North. The default value is 315.

**altitude** — The altitude of the sun. The altitude is measured in degrees from the horizon. The default value is 45.

**max\_bright** — The maximum brightness of the hillshade. The default value is 255.

**scale** — The scale of the hillshade. The default value is 1.0.

**interpolate\_nodata** — Whether to interpolate nodata values. The default value is FALSE.

**Note**

How hillshade works

2.0.0 ST\_MapAlgebra() interpolate\_nodata

2.1.0 ST\_MapAlgebra() interpolate\_nodata

2.1.0 ST\_MapAlgebra() interpolate\_nodata

**ST\_Hillshade(1)**

```
WITH foo AS (
  SELECT ST_SetValues(
    ST_AddBand(ST_MakeEmptyRaster(5, 5, 0, 0, 1, -1, 0, 0, 0), 1, '32BF', 0, -9999),
    1, 1, 1, ARRAY[
      [1, 1, 1, 1, 1],
      [1, 2, 2, 2, 1],
      [1, 2, 3, 2, 1],
      [1, 2, 2, 2, 1],
      [1, 1, 1, 1, 1]
    ]::double precision[]
  ) AS rast
)
SELECT
  ST_DumpValues(ST_Hillshade(rast, 1, '32BF'))
FROM foo
```

```
(1, "{NULL,NULL,NULL,NULL,NULL},{NULL,251.32763671875,220.749786376953,147.224319458008,
NULL},{NULL,220.749786376953,180.312225341797,67.7497863769531,NULL},{NULL,
147.224319458008
,67.7497863769531,43.1210060119629,NULL},{NULL,NULL,NULL,NULL,NULL}")
(1 row)
```

**ST\_Hillshade(2)**

PostgreSQL 9.1 ST\_Hillshade(2)

```
WITH foo AS (
  SELECT ST_Tile(
    ST_SetValues(
      ST_AddBand(
        ST_MakeEmptyRaster(6, 6, 0, 0, 1, -1, 0, 0, 0),
```

```

        1, '32BF', 0, -9999
    ),
    1, 1, 1, ARRAY[
        [1, 1, 1, 1, 1, 1],
        [1, 1, 1, 1, 2, 1],
        [1, 2, 2, 3, 3, 1],
        [1, 1, 3, 2, 1, 1],
        [1, 2, 2, 1, 2, 1],
        [1, 1, 1, 1, 1, 1]
    ]::double precision[]
),
    2, 2
) AS rast
)
SELECT
    t1.rast,
    ST_Hillshade(ST_Union(t2.rast), 1, t1.rast)
FROM foo t1
CROSS JOIN foo t2
WHERE ST_Intersects(t1.rast, t2.rast)
GROUP BY t1.rast;

```

**Example 12.14.2**

Example 12.14.2: `ST_Triangulate`, `ST_TPI`, `ST_Roughness`, `ST_Aspect`, `ST_Slope`

### 12.14.3 ST\_Roughness

`ST_Roughness` — DEM (raster), integer nband, raster customextent, text pixeltype='32BF', boolean interpolate\_nodata=FALSE

#### Synopsis

`raster ST_Roughness(raster rast, integer nband, raster customextent, text pixeltype='32BF', boolean interpolate_nodata=FALSE);`

**Example 12.14.3**

Example 12.14.3: `ST_Roughness`, `ST_Triangulate`, `ST_TPI`, `ST_Slope`, `ST_Hillshade`, `ST_Aspect`

2.1.0 `ST_Roughness`, `ST_Triangulate`, `ST_TPI`, `ST_Slope`, `ST_Hillshade`, `ST_Aspect`

**Example 12.14.4**

```
-- Example 12.14.4: ST_Roughness, ST_Triangulate, ST_TPI, ST_Slope, ST_Hillshade, ST_Aspect
```

**Example 12.14.5**

Example 12.14.5: `ST_Triangulate`, `ST_TPI`, `ST_Slope`, `ST_Hillshade`, `ST_Aspect`

## 12.14.4 ST\_Slope

**ST\_Slope** — Compute the slope of a raster. The slope is calculated as the square root of the sum of the squares of the horizontal and vertical derivatives. The slope is returned in degrees by default, but can be returned in radians or percent. The slope is calculated as the square root of the sum of the squares of the horizontal and vertical derivatives. The slope is returned in degrees by default, but can be returned in radians or percent.

### Synopsis

raster **ST\_Slope**(raster rast, integer nband=1, text pixeltype=32BF, text units=DEGREES, double precision scale=1.0, boolean interpolate\_nodata=FALSE);

raster **ST\_Slope**(raster rast, integer nband, raster customextent, text pixeltype=32BF, text units=DEGREES, double precision scale=1.0, boolean interpolate\_nodata=FALSE);

### Parameters

**rast**: raster to be processed.  
**nband**: number of bands to be processed.  
**pixeltype**: pixel type of the output raster.  
**units**: units of the slope. Valid values are RADIANS, DEGREES, and PERCENT.  
**scale**: scale factor for the slope. Valid values are 1.0, 111120, and 370400.  
**interpolate\_nodata**: boolean flag to indicate whether to interpolate nodata values.

### Note



The slope is calculated as the square root of the sum of the squares of the horizontal and vertical derivatives. The slope is returned in degrees by default, but can be returned in radians or percent. The slope is calculated as the square root of the sum of the squares of the horizontal and vertical derivatives. The slope is returned in degrees by default, but can be returned in radians or percent.

2.0.0 **ST\_Slope**(rast, nband, pixeltype, units, scale, interpolate\_nodata);

2.1.0 **ST\_Slope**(rast, nband, customextent, pixeltype, units, scale, interpolate\_nodata);

2.1.0 **ST\_Slope**(rast, nband, customextent, pixeltype, units, scale, interpolate\_nodata);

### Usage

```
WITH foo AS (
  SELECT ST_SetValues(
    ST_AddBand(ST_MakeEmptyRaster(5, 5, 0, 0, 1, -1, 0, 0, 0), 1, '32BF', 0, -9999),
    1, 1, 1, ARRAY[
      [1, 1, 1, 1, 1],
```

```

        [1, 2, 2, 2, 1],
        [1, 2, 3, 2, 1],
        [1, 2, 2, 2, 1],
        [1, 1, 1, 1, 1]
    ]::double precision[][])
    ) AS rast
)
SELECT
    ST_DumpValues(ST_Slope(rast, 1, '32BF'))
FROM foo

          st_dumpvalues
-----
(1, "{10.0249881744385,21.5681285858154,26.5650520324707,21.5681285858154,10.0249881744385},{21.5681285858154,26.5650520324707,36.8698959350586,0,36.8698959350586,26.5650520324707},{21.5681285858154,35.26438905681285858154,26.5650520324707,21.5681285858154,10.0249881744385}}")
(1 row)

```

## ST\_Slope

ST\_Slope(rast, band, format) returns the slope of the raster band as a raster of the same format and resolution. The slope is calculated as the square root of the sum of the squares of the horizontal and vertical derivatives. The format is the same as the raster. The band is the band number of the raster. The format is the same as the raster. The band is the band number of the raster.

```

WITH foo AS (
    SELECT ST_Tile(
        ST_SetValues(
            ST_AddBand(
                ST_MakeEmptyRaster(6, 6, 0, 0, 1, -1, 0, 0, 0),
                1, '32BF', 0, -9999
            ),
            1, 1, 1, ARRAY[
                [1, 1, 1, 1, 1, 1],
                [1, 1, 1, 1, 2, 1],
                [1, 2, 2, 3, 3, 1],
                [1, 1, 3, 2, 1, 1],
                [1, 2, 2, 1, 2, 1],
                [1, 1, 1, 1, 1, 1]
            ]::double precision[]
        ),
        2, 2
    ) AS rast
)
SELECT
    t1.rast,
    ST_Slope(ST_Union(t2.rast), 1, t1.rast)
FROM foo t1
CROSS JOIN foo t2
WHERE ST_Intersects(t1.rast, t2.rast)
GROUP BY t1.rast;

```

**ST\_TPI**

[ST\\_TPI](#), [ST\\_TPI](#), [ST\\_Roughness](#), [ST\\_HillShade](#), [ST\\_Aspect](#)

### 12.14.5 ST\_TPI

**ST\_TPI** — (Topographic Position Index)

#### Synopsis

raster **ST\_TPI**(raster rast, integer nband, raster customextent, text pixeltype="32BF" , boolean interpolate\_nodata=FALSE );

**ST\_TPI**

Calculates the Topographic Position Index, which is defined as the focal mean with radius of one minus the center cell.



#### Note

1 (focalmean radius of one)

2.1.0

**ST\_TPI**

```
-- ST_TPI
```

**ST\_TPI**

[ST\\_TPI](#), [ST\\_Roughness](#), [ST\\_Slope](#), [ST\\_HillShade](#), [ST\\_Aspect](#)

### 12.14.6 ST\_TRI

**ST\_TRI** — (Terrain Ruggedness Index)

#### Synopsis

raster **ST\_TRI**(raster rast, integer nband, raster customextent, text pixeltype="32BF" , boolean interpolate\_nodata=FALSE );

**12.14. Box3D**

Box3D — a 3D bounding box. It is defined by the minimum and maximum coordinates in each dimension. The coordinates are given in the order (xmin, xmax, ymin, ymax, zmin, zmax). The zmin and zmax values are optional and default to -infinity and infinity, respectively. The Box3D type is used to store 3D bounding boxes for spatial data.

**Note**

Box3D is a 3D bounding box. It is defined by the minimum and maximum coordinates in each dimension. The coordinates are given in the order (xmin, xmax, ymin, ymax, zmin, zmax). The zmin and zmax values are optional and default to -infinity and infinity, respectively. The Box3D type is used to store 3D bounding boxes for spatial data.

2.1.0 `BOX3D(xmin, xmax, ymin, ymax, zmin, zmax)`; `BOX3D(xmin, xmax, ymin, ymax)`; `BOX3D(xmin, xmax, ymin, ymax, zmin)`; `BOX3D(xmin, xmax, ymin, ymax, zmin, zmax)`.

**12.15. Box3D**

```
-- BOX3D(xmin, xmax, ymin, ymax, zmin, zmax);
```

**12.15.1. Box3D**

Box3D — a 3D bounding box. It is defined by the minimum and maximum coordinates in each dimension. The coordinates are given in the order (xmin, xmax, ymin, ymax, zmin, zmax). The zmin and zmax values are optional and default to -infinity and infinity, respectively. The Box3D type is used to store 3D bounding boxes for spatial data.

**12.15.1.1. Box3D****12.15.1.1.1. Box3D**

Box3D — a 3D bounding box. It is defined by the minimum and maximum coordinates in each dimension. The coordinates are given in the order (xmin, xmax, ymin, ymax, zmin, zmax). The zmin and zmax values are optional and default to -infinity and infinity, respectively. The Box3D type is used to store 3D bounding boxes for spatial data.

**Synopsis**

`box3d` **Box3D**(raster rast);

**12.15.1.1.1. Box3D**

Box3D — a 3D bounding box. It is defined by the minimum and maximum coordinates in each dimension. The coordinates are given in the order (xmin, xmax, ymin, ymax, zmin, zmax). The zmin and zmax values are optional and default to -infinity and infinity, respectively. The Box3D type is used to store 3D bounding boxes for spatial data.

Box3D — a 3D bounding box. It is defined by the minimum and maximum coordinates in each dimension. The coordinates are given in the order (xmin, xmax, ymin, ymax, zmin, zmax). The zmin and zmax values are optional and default to -infinity and infinity, respectively. The Box3D type is used to store 3D bounding boxes for spatial data.

Box3D — a 3D bounding box. It is defined by the minimum and maximum coordinates in each dimension. The coordinates are given in the order (xmin, xmax, ymin, ymax, zmin, zmax). The zmin and zmax values are optional and default to -infinity and infinity, respectively. The Box3D type is used to store 3D bounding boxes for spatial data.



```

-- &#xc5d4;&#xbca8;&#xb85c;&#xd504;&#xc640; &#xbcf4;&#xb85d; &#xae30;&#xc9c8;&#xc774; &#xc774;&#xc81c; &#xbcf4;&#xb85d; &#xae30;&#xc9c8;&#xcacfc; &#xc5d4;&#xbca8;&#xb85c;&#xd504;&#xc00; &#xc5b4;&#xb5bb;&#xc8c; &#xb2ec;&#xb77c;&#xc9c0;&#xb294;&#xc9c0; &#xb208;&#xc5ec;&#xcaca8; &#xbcf4;&#xc2ed;&#xc2dc;&#xc624;.
SELECT ST_AsText(ST_ConvexHull(rast)) As convhull,
       ST_AsText(ST_Envelope(rast)) As env
FROM dummy_rast WHERE rid=1;

```

| convhull   | env                                |
|--|------------------------------------|
| POLYGON((0.5 0.5,20.5 0.5,20.5 60.5,0.5 60.5,0.5 0.5)) | POLYGON((0 0,20 0,20 60,0 60,0 0)) |

```

-- &#xb798;&#xc2a4;&#xd130;&#xb97c; &#xae30;&#xc6b8;&#xc785;&#xb2c8;&#xb2e4;.
-- &#xc774;&#xc81c; &#xbcf4;&#xb85d; &#xae30;&#xc9c8;&#xcacfc; &#xc5d4;&#xbca8;&#xb85c;&#xd504;&#xc00; &#xc5b4;&#xb5bb;&#xc8c; &#xb2ec;&#xb77c;&#xc9c0;&#xb294;&#xc9c0; &#xb208;&#xc5ec;&#xcaca8; &#xbcf4;&#xc2ed;&#xc2dc;&#xc624;.
SELECT ST_AsText(ST_ConvexHull(rast)) As convhull,
       ST_AsText(ST_Envelope(rast)) As env
FROM (SELECT ST_SetRotation(rast, 0.1, 0.1) As rast
      FROM dummy_rast WHERE rid=1) As foo;

```

| convhull   | env                                |
|--|------------------------------------|
| POLYGON((0.5 0.5,20.5 1.5,22.5 61.5,2.5 60.5,0.5 0.5)) | POLYGON((0 0,22 0,22 61,0 61,0 0)) |

**&#xc38;&#xc0;**

[ST\\_Envelope](#), [ST\\_MinConvexHull](#), [ST\\_ConvexHull](#), [ST\\_AsText](#)

### 12.15.3 ST\_DumpAsPolygons

**ST\_DumpAsPolygons** — [&#xc785;&#xb825;](#) [&#xb798;&#xc2a4;&#xd130;](#) [&#xc34;&#xb4dc;&#xb85c;&#xbd80;&#xd130;](#) [geomval\(geom, val\)](#) [&#xd589;&#xb4e4;&#xc758;](#) [&#xc9d1;&#xd569;&#xc744;](#) [&#xc18;&#xd658;&#xd569;&#xb2c8;&#xb2e4;](#) [&#xc34;&#xb4dc;](#) [&#xc88;&#xd638;&#xb97c;](#) [&#xc124;&#xc815;&#xd558;&#xc9c0;](#) [&#xc54a;&#xc744;](#) [&#xcabd;&#xc6b0;](#) [&#xae30;&#xbcf8;&#xc801;&#xc73c;&#xb85c;](#) [&#xc34;&#xb4dc;](#) [1&#xb85c;](#) [&#xc00;&#xc815;&#xd569;&#xb2c8;&#xb2e4;](#).

#### Synopsis

setof geomval **ST\_DumpAsPolygons**(raster rast, integer band\_num=1, boolean exclude\_nodata\_value=TRUE);

**&#xc124;&#xba85;**

[&#xc774;](#) [&#xd568;&#xc218;&#xb294;](#) [&#xc9d1;&#xd569;](#) [&#xc18;&#xd658;](#) [&#xd568;&#xc218;](#)(SRF; Set-Returning Function)[&#xc785;&#xb2c8;&#xb2e4;](#). [&#xc34;&#xd615;](#)(geom)[&#xcacfc;](#) [&#xd53d;&#xc140;](#) [&#xc34;&#xb4dc;](#) [&#xc12;](#)(val)[&#xc774;&#xb8e8;&#xc5b4;&#xc9c4;](#) geomval [&#xd589;&#xb4e4;&#xc758;](#) [&#xc9d1;&#xd569;&#xc744;](#) [&#xc18;&#xd658;&#xc01;](#) [&#xd3f4;&#xb9ac;&#xc4;](#)[&#xc740;](#) [&#xd574;&#xb2f9;](#) [&#xc34;&#xb4dc;&#xc5d0;&#xc11c;](#) val[&#xc774;](#) [&#xd45c;&#xc12;&#xcacfc;](#) [&#xb3d9;&#xc77c;&#xd55c;](#) [&#xd53d;&#xc140;](#) [&#xc12;&#xc744;](#) [&#xc00;&#xc9c4;](#) [&#xbaa8;&#xb4e0;](#) [&#xd53d;&#xc140;&#xb4e4;&#xc744;](#) [&#xd1b5;&#xd569;&#xd55c;](#) [&#xc83;&#xc785;&#xb2c8;&#xb2e4;](#).

**ST\_DumpAsPolygon**[&#xc740;](#) [&#xb798;&#xc2a4;&#xd130;&#xb97c;](#) [&#xd3f4;&#xb9ac;&#xc4;](#)[&#xd654;&#xd558;&#xb294;](#) [&#xb370;](#) [&#xc720;&#xc6a9;&#xd569;&#xb2c8;&#xb2e4;](#). [&#xc0c8;](#) [&#xd589;&#xb4e4;&#xc744;](#) [&#xc0dd;&#xc131;&#xd55c;&](#)

GROUP BY; &#xc5ed;&#xd568;&#xc218;&#xb77c;&#xace0; &#xd560; &#xc218; &#xc788; &#xc608;&#xb97c; &#xb4e4;&#xc5b4; &#xb2e8;&#xc77c; &#xb798;&#xc2a4;&#xd130;&#xb97c; &#xbcf5;&#xc218;&#xc758; &#xd3f4;&#xb9ac;&#xace4;/&#xba40;&#xd2f0;&#xd3f4;&#xb9ac;&#xace4;&#xc73c;&#xb85c; &#xd655;&#xc7a5;&#xd558;&#xb370; &#xc774; &#xd568;&#xc218;&#xb97c; &#xc4f8; &#xc218; &#xc788;&#xc2b5;&#xb2c8;&#xb2e4;.

Changed 3.3.0, validation and fixing is disabled to improve performance. May result invalid geometries.

GDAL 1.7 &#xc774;&#xc0c1; &#xbc84;&#xc804;&#xc774; &#xd544;&#xc694;&#xd569;&#xb2c8;&#xb2e4;.



#### Note

If there is a no data value set for a band, pixels with that value will not be returned except in the case of `exclude_nodata_value=false`.



#### Note

&#xb798;&#xc2a4;&#xd130; &#xc548;&#xc5d0; &#xc788;&#xb294; &#xd574;&#xb2f9; &#xac12;&#xc744; &#xac00;&#xc9c4; &#xd53d;&#xc140;&#xb4e4;&#xc758; &#xac1c;&#xc218;&#xc5d0;&#xb9cc; &#xad00;&#xc2ec;&#xc774; &#xc788;&#xb2e4;&#xba74;,, **ST\_ValueCount** &#xd568;&#xc218;&#xac00; &#xb354; &#xbe60;&#xb985;&#xb2c8;&#xb2e4;.



#### Note

&#xc774; &#xd568;&#xc218;&#xb294; &#xd53d;&#xc140; &#xac12;&#xacc; &#xc0c1;&#xad00;&#xc5c6;&#xc774; &#xac01; &#xd53d;&#xc140;&#xc5d0; &#xb300;&#xd574; &#xb3c4;&#xd615; &#xd558;&#xb098;&#xb97c; &#xbc18;&#xd658;&#xd558;&#xb294; **ST\_PixelAsPolygons** &#xd568;&#xc218;&#xc640; &#xb2e4;&#xb985;&#xb2c8;&#xb2e4;.

### &#xc608;&#xc2dc;

```
-- this syntax requires PostgreSQL 9.3+
SELECT val, ST_AsText(geom) As geomwkt
FROM (
  SELECT dp.*
  FROM dummy_rast, LATERAL ST_DumpAsPolygons(rast) AS dp
  WHERE rid = 2
) As foo
WHERE val BETWEEN 249 and 251
ORDER BY val;
```

| val | geomwkt  |
|-----|--|
| 249 | POLYGON((3427927.95 5793243.95,3427927.95 5793243.85,3427928 5793243.85,3427928 5793243.95,3427927.95 5793243.95))   |
| 250 | POLYGON((3427927.75 5793243.9,3427927.75 5793243.85,3427927.8 5793243.85,3427927.8 5793243.9,3427927.75 5793243.9))  |
| 250 | POLYGON((3427927.8 5793243.8,3427927.8 5793243.75,3427927.85 5793243.75,3427927.85 5793243.8,3427927.8 5793243.8))   |
| 251 | POLYGON((3427927.75 5793243.85,3427927.75 5793243.8,3427927.8 5793243.8,3427927.8 5793243.85,3427927.75 5793243.85)) |

### &#xcc38;&#xace0;

**geomval, ST\_Value, ST\_Polygon, ST\_ValueCount**

## 12.15.4 ST\_Envelope

`ST_Envelope` — Returns the bounding box of a geometry. The bounding box is a rectangle that encloses the geometry. The bounding box is defined by the minimum and maximum X and Y coordinates of the geometry.

### Synopsis

geometry `ST_Envelope`(raster rast);

#### Parameters

`rast` — A raster. The bounding box is calculated based on the non-null pixels of the raster. The bounding box is defined by the minimum and maximum X and Y coordinates of the non-null pixels.

`SRID` — The SRID of the bounding box. If not specified, the SRID of the raster is used. The SRID is a numeric value that identifies the coordinate system of the geometry.

#### Examples

```
SELECT rid, ST_AsText(ST_Envelope(rast)) As envgeomwkt
FROM dummy_rast;
```

| rid | envgeomwkt   |
|-----|--|
| 1   | POLYGON((0 0,20 0,20 60,0 60,0 0))   |
| 2   | POLYGON((3427927 5793243,3427928 5793243,3427928 5793244,3427927 5793244,3427927 5793243)) |

#### See Also

[ST\\_Envelope](#), [ST\\_AsText](#), [ST\\_SRID](#)

## 12.15.5 ST\_MinConvexHull

`ST_MinConvexHull` — Returns the minimum convex hull of a geometry. The minimum convex hull is the smallest convex polygon that encloses the geometry. The minimum convex hull is defined by the minimum and maximum X and Y coordinates of the geometry.

### Synopsis

geometry `ST_MinConvexHull`(raster rast, integer nband=NULL);

#### Parameters

`rast` — A raster. The minimum convex hull is calculated based on the non-null pixels of the raster. The minimum convex hull is defined by the minimum and maximum X and Y coordinates of the non-null pixels.

`nband` — The number of bands to consider. If not specified, all bands are considered. The number of bands is a numeric value that identifies the number of bands in the raster.

2.1.0 Returns the bounding box of a geometry. The bounding box is a rectangle that encloses the geometry. The bounding box is defined by the minimum and maximum X and Y coordinates of the geometry.



## Synopsis

geometry **ST\_Polygon**(raster rast, integer band\_num=1);

### Changes

Changed 3.3.0, validation and fixing is disabled to improve performance. May result invalid geometries.

0.1.6 GDAL 1.7

2.1.0

2.1.0

### Examples

```
-- NODATA
SELECT ST_AsText(ST_Polygon(rast)) As geomwkt
FROM dummy_rast
WHERE rid = 2;

geomwkt
-----
MULTIPOLYGON(((3427927.75 5793244,3427928 5793244,3427928 5793243.75,3427927.75
5793243.75,3427927.75 5793244)))

-- NODATA
UPDATE dummy_rast SET rast = ST_SetBandNoDataValue(rast,1,254)
WHERE rid = 2;
SELECT rid, ST_BandNoDataValue(rast)
from dummy_rast where rid = 2;

-- ST_Polygon
SELECT ST_AsText(ST_Polygon(rast)) As geomwkt
FROM dummy_rast
WHERE rid = 2;

geomwkt
-----
MULTIPOLYGON(((3427927.9 5793243.95,3427927.85 5793243.95,3427927.85 5793244,3427927.9
5793244,3427927.9 5793243.95)),((3427928 5793243.85,3427928 5793243.8,3427927.95
5793243.8,3427927.95 5793243.85,3427927.9 5793243.85,3427927.9 5793243.9,3427927.9
5793243.95,3427927.95 5793243.95,3427928 5793243.95,3427928 5793243.85)),((3427927.8
5793243.75,3427927.75 5793243.75,3427927.75 5793243.8,3427927.75 5793243.85,3427927.75
5793243.9,3427927.75 5793244,3427927.8 5793244,3427927.8 5793243.9,3427927.8
5793243.85,3427927.85 5793243.85,3427927.85 5793243.8,3427927.85 5793243.75,3427927.8
5793243.75)))
```

```
-- NODATA;
SELECT ST_AsText (
  ST_Polygon(
    ST_SetBandNoDataValue (rast, 1, 252)
  ) As geomwkt
FROM dummy_rast
WHERE rid =2;

geomwkt
-----
MULTIPOLYGON(((3427928 5793243.85,3427928 5793243.8,3427928 5793243.75,3427927.85
5793243.75,3427927.8 5793243.75,3427927.8 5793243.8,3427927.75 5793243.8,3427927.75
5793243.85,3427927.75 5793243.9,3427927.75 5793244,3427927.8 5793244,3427927.85
5793244,3427927.9 5793244,3427928 5793244,3427928 5793243.95,3427928 5793243.85)
,(3427927.9 5793243.9,3427927.9 5793243.85,3427927.95 5793243.85,3427927.95
5793243.9,3427927.9 5793243.9)))
```

**ST\_Value**

**ST\_DumpAsPolygons**

## 12.16 ST\_Value, ST\_DumpAsPolygons

### 12.16.1 ST\_Value

**ST\_Value** ( raster A , geometry B )  
**ST\_DumpAsPolygons** ( raster A , geometry B , boolean TRUE )

#### Synopsis

boolean **ST\_Value**( raster A , raster B );  
boolean **ST\_Value**( raster A , geometry B );  
boolean **ST\_DumpAsPolygons**( geometry B , raster A );

#### ST\_Value

**ST\_Value** ( raster A , geometry B )  
**ST\_DumpAsPolygons** ( raster A , geometry B , boolean TRUE )



#### Note

**ST\_Value** ( raster A , geometry B )  
**ST\_DumpAsPolygons** ( raster A , geometry B , boolean TRUE )

2.0.0 **ST\_Value**( raster A , raster B );

**Intersection**

```
SELECT A.rid As a_rid, B.rid As b_rid, A.rast && B.rast As intersect
FROM dummy_rast AS A CROSS JOIN dummy_rast AS B LIMIT 3;
```

| a_rid | b_rid | intersect |
|-------|-------|-----------|
| 2     | 2     | t         |
| 2     | 3     | f         |
| 2     | 1     | f         |

**12.16.2 &<**

**&<** — A **&&** B; TRUE; FALSE

**Synopsis**

boolean **&<**( raster A , raster B );

**Parameters**

**&<** — A; B; TRUE; FALSE

**Note**

**&<** — A; B; TRUE; FALSE

**Intersection**

```
SELECT A.rid As a_rid, B.rid As b_rid, A.rast &< B.rast As overleft
FROM dummy_rast AS A CROSS JOIN dummy_rast AS B;
```

| a_rid | b_rid | overleft |
|-------|-------|----------|
| 2     | 2     | t        |
| 2     | 3     | f        |
| 2     | 1     | f        |
| 3     | 2     | t        |
| 3     | 3     | t        |
| 3     | 1     | f        |
| 1     | 2     | t        |
| 1     | 3     | t        |
| 1     | 1     | t        |

### 12.16.3 &>

**&>** — A & B; TRUE & FALSE;.

#### Synopsis

boolean **&>**( raster A , raster B );

#### Notes

**&>** A & B; TRUE & FALSE;.



#### Note

**&>** (operand) & B; TRUE & FALSE;.

#### Example

```
SELECT A.rid As a_rid, B.rid As b_rid, A.rast &> B.rast As overright
FROM dummy_rast AS A CROSS JOIN dummy_rast AS B;

a_rid | b_rid | overright
-----+-----+-----
2 | 2 | t
2 | 3 | t
2 | 1 | t
3 | 2 | f
3 | 3 | t
3 | 1 | f
1 | 2 | f
1 | 3 | t
1 | 1 | t
```

### 12.16.4 =

**=** — A & B; TRUE & FALSE;.

#### Synopsis

boolean **=**( raster A , raster B );



**ST\_Intersection**

~

## 12.16.6 ~ =

**~ =** — A & B; B & A; TRUE; FALSE;

### Synopsis

boolean **~**( raster A , raster B );

**ST\_Intersection**

**~ =** **ST\_Intersection**( raster A , raster B );



#### Note

**ST\_Intersection**( raster A , raster B ) returns TRUE if the two rasters have the same value for the same cell, and FALSE otherwise.

2.0.0 **ST\_Intersection**( raster A , raster B );

**ST\_Intersection**

**ST\_Intersection**( raster A , raster B ) returns TRUE if the two rasters have the same value for the same cell, and FALSE otherwise.

```
SELECT ST_AddBand(prec.rast, alt.rast) As new_rast
FROM prec INNER JOIN alt ON (prec.rast ~ = alt.rast);
```

**ST\_Intersection**

**ST\_AddBand**, =

## 12.16.7 ~

**~** — A & B; B & A; TRUE; FALSE;

### Synopsis

boolean **~**( raster A , raster B );  
boolean **~**( geometry A , raster B );  
boolean **~**( raster B , geometry A );

**12.17 ST\_Contains**

~ **ST\_Contains**( raster rastA , integer nbandA , raster rastB , integer nbandB );  
**ST\_Contains**( raster rastA , raster rastB );

**Note**

**ST\_Contains**( raster rastA , integer nbandA , raster rastB , integer nbandB );  
**ST\_Contains**( raster rastA , raster rastB );

2.0.0 **ST\_Contains**( raster rastA , integer nbandA , raster rastB , integer nbandB );

**12.17.1 ST\_Contains**

@

## 12.17 ST\_Contains

### 12.17.1 ST\_Contains

**12.17.1 ST\_Contains**

**ST\_Contains** — **ST\_Contains**( raster rastA , integer nbandA , raster rastB , integer nbandB );  
**ST\_Contains**( raster rastA , raster rastB );

**Synopsis**

boolean **ST\_Contains**( raster rastA , integer nbandA , raster rastB , integer nbandB );  
boolean **ST\_Contains**( raster rastA , raster rastB );

**12.17.1 ST\_Contains**

**ST\_Contains**( raster rastA , integer nbandA , raster rastB , integer nbandB );  
**ST\_Contains**( raster rastA , raster rastB );

**Note**

**ST\_Contains**( raster rastA , integer nbandA , raster rastB , integer nbandB );  
**ST\_Contains**( raster rastA , raster rastB );

**Note**

`ST_Contains(ST_Polygon(raster), geometry)` is equivalent to `ST_Contains(geometry, ST_Polygon(raster))`.

**Note**

`ST_Contains()` and `ST_Within()` are equivalent. `ST_Contains(rastA, rastB)` is equivalent to `ST_Within(rastB, rastA)`.

2.1.0

**Example 1**

```
-- Example 1
SELECT r1.rid, r2.rid, ST_Contains(r1.rast, 1, r2.rast, 1) FROM dummy_rast r1 CROSS JOIN
  dummy_rast r2 WHERE r1.rid = 1;
```

NOTICE: The first raster provided has no bands

| rid | rid | st_contains |
|-----|-----|-------------|
| 1   | 1   | t           |
| 1   | 2   | f           |

```
-- Example 2
SELECT r1.rid, r2.rid, ST_Contains(r1.rast, r2.rast) FROM dummy_rast r1 CROSS JOIN
  dummy_rast r2 WHERE r1.rid = 1;
```

| rid | rid | st_contains |
|-----|-----|-------------|
| 1   | 1   | t           |
| 1   | 2   | f           |

**Example 3**

[ST\\_Intersects](#), [ST\\_Within](#)

**12.17.2 ST\_ContainsProperly**

`ST_ContainsProperly` — `rastB` is properly contained within `rastA`.

**Synopsis**

boolean `ST_ContainsProperly`( raster `rastA` , integer `nbandA` , raster `rastB` , integer `nbandB` );  
 boolean `ST_ContainsProperly`( raster `rastA` , raster `rastB` );





**ST\_CoveredBy**

ST\_CoveredBy(rastA, nbandA, rastB, nbandB) → boolean  
 ST\_CoveredBy(rastA, rastB) → boolean  
 ST\_CoveredBy(rastA, rastB, nbandA, nbandB) → boolean  
 ST\_CoveredBy(geometry, geometry) → boolean  
 ST\_CoveredBy(geometry, geometry, nbandA, nbandB) → boolean  
 ST\_CoveredBy(rastA, rastB, nbandA, nbandB, mode) → boolean  
 ST\_CoveredBy(rastA, rastB, nbandA, nbandB, mode, tolerance) → boolean  
 (NODATA) → boolean



**Note**

ST\_CoveredBy(rastA, rastB, nbandA, nbandB) → boolean  
 ST\_CoveredBy(rastA, rastB, nbandA, nbandB, mode) → boolean  
 ST\_CoveredBy(rastA, rastB, nbandA, nbandB, mode, tolerance) → boolean



**Note**

ST\_CoveredBy(geometry, geometry) → boolean  
 ST\_CoveredBy(geometry, geometry, nbandA, nbandB) → boolean  
 ST\_CoveredBy(geometry, geometry, nbandA, nbandB, mode) → boolean  
 ST\_CoveredBy(geometry, geometry, nbandA, nbandB, mode, tolerance) → boolean

2.1.0 ST\_CoveredBy(geometry, geometry, nbandA, nbandB, mode, tolerance) → boolean

**ST\_Covers**

```
SELECT r1.rid, r2.rid, ST_CoveredBy(r1.rast, 1, r2.rast, 1) FROM dummy_rast r1 CROSS JOIN
    dummy_rast r2 WHERE r1.rid = 2;
```

| rid | rid | st_coveredby |
|-----|-----|--------------|
| 2   | 1   | f            |
| 2   | 2   | t            |

**ST\_Intersection**

ST\_Intersection, ST\_Covers

**12.17.5 ST\_Disjoint**

ST\_Disjoint — ST\_Disjoint(rastA, nbandA, rastB, nbandB) → boolean  
 ST\_Disjoint(rastA, rastB) → boolean  
 ST\_Disjoint(rastA, rastB, nbandA, nbandB) → boolean  
 ST\_Disjoint(geometry, geometry) → boolean  
 ST\_Disjoint(geometry, geometry, nbandA, nbandB) → boolean  
 ST\_Disjoint(rastA, rastB, nbandA, nbandB, mode) → boolean  
 ST\_Disjoint(rastA, rastB, nbandA, nbandB, mode, tolerance) → boolean  
 (NODATA) → boolean

**Synopsis**

boolean ST\_Disjoint( raster rastA , integer nbandA , raster rastB , integer nbandB );  
 boolean ST\_Disjoint( raster rastA , raster rastB );

**ST\_Difference**

```

rastA ST_Difference(rastB, rastA)
rastB ST_Difference(rastA, rastB)
ST_Difference(rastA, rastB)
ST_Difference(rastA, NULL)
ST_Difference(NULL, rastA)
ST_Difference(rastA, rastB, 'NODATA')

```

**Note**

ST\_Difference(rastA, rastB) returns NULL if either raster is NULL or if the rasters have different SRIDs or different units.

**Note**

ST\_Difference(rastA, rastB) returns NULL if either raster is NULL or if the rasters have different SRIDs or different units. The function also returns NULL if the rasters have different units.

2.1.0 ST\_Difference(rastA, rastB, 'NODATA')

**ST\_Difference**

```

-- rid = 1 ST_Difference(rastA, rastB)
-- rid = 2 ST_Difference(rastA, rastB, 'NODATA')
SELECT r1.rid, r2.rid, ST_Difference(r1.rast, 1, r2.rast, 1) FROM dummy_rast r1 CROSS JOIN
dummy_rast r2 WHERE r1.rid = 2;

```

NOTICE: The second raster provided has no bands

| rid | rid | st_disjoint |
|-----|-----|-------------|
| 2   | 1   | t           |
| 2   | 2   | f           |

```

-- ST_Difference(rastA, rastB, 'NODATA')
SELECT r1.rid, r2.rid, ST_Difference(r1.rast, r2.rast) FROM dummy_rast r1 CROSS JOIN
dummy_rast r2 WHERE r1.rid = 2;

```

| rid | rid | st_disjoint |
|-----|-----|-------------|
| 2   | 1   | t           |
| 2   | 2   | f           |

**ST\_Intersection****ST\_Intersection**



## 12.17.7 ST\_Overlaps

**ST\_Overlaps** — Returns true if the two input geometries overlap, that is, if they share a common area. The input geometries must be of the same SRID and must be of the same dimensionality. The input geometries must be non-empty and must not be NULL.

### Synopsis

boolean **ST\_Overlaps**( raster rastA , integer nbandA , raster rastB , integer nbandB );

boolean **ST\_Overlaps**( raster rastA , raster rastB );

### Parameters

**rastA**, **rastB**: Raster objects to be compared.  
**nbandA**, **nbandB**: The number of bands in the input rasters. If not specified, the number of bands in the input rasters is used.  
**geomA**, **geomB**: Geometry objects to be compared.  
**geomA\_srid**, **geomB\_srid**: The SRID of the input geometries. If not specified, the SRID of the input geometries is used.  
**geomA**, **geomB**, **geomA\_srid**, **geomB\_srid**: The input geometries and their SRIDs.  
**geomA**, **geomB**, **geomA\_srid**, **geomB\_srid**, **geomA\_srid**, **geomB\_srid**: The input geometries and their SRIDs, with the SRID of the input geometries specified twice.



#### Note

ST\_Overlaps(geomA, geomB) is equivalent to ST\_Overlaps(geomA, geomB, ST\_SRID(geomA), ST\_SRID(geomB)).



#### Note

ST\_Overlaps(geomA, geomB) is equivalent to ST\_Overlaps(geomA, geomB, ST\_SRID(geomA), ST\_SRID(geomB)).

2.1.0 ST\_Overlaps(geomA, geomB) is equivalent to ST\_Overlaps(geomA, geomB, ST\_SRID(geomA), ST\_SRID(geomB)).

### Examples

```
-- Create two overlapping polygons
CREATE TABLE dummy_rast (
  rid integer,
  geom geometry,
  PRIMARY KEY (rid)
);
INSERT INTO dummy_rast VALUES (1, ST_Polygon('POLYGON((0 0, 100 0, 100 100, 0 100, 0 0))'));
INSERT INTO dummy_rast VALUES (2, ST_Polygon('POLYGON((50 0, 150 0, 150 100, 50 100, 50 0))'));

SELECT ST_Overlaps(geom, 1, geom, 2) FROM dummy_rast WHERE rid = 2;
```

```
st_overlaps
-----
f
```

### See Also

[ST\\_Intersects](#)

## 12.17.8 ST\_Touches

**ST\_Touches** — `rastA` `rastB`; `TRUE`

### Synopsis

boolean **ST\_Touches**( raster `rastA` , integer `nbandA` , raster `rastB` , integer `nbandB` );  
 boolean **ST\_Touches**( raster `rastA` , raster `rastB` );

### Notes

`rastA` `rastB`; `TRUE` `FALSE` `NULL`



#### Note

`ST_Touches` returns `TRUE` if the two rasters touch at any point, `FALSE` if they do not, and `NULL` if either raster is `NULL`.



#### Note

`ST_Touches` uses the `ST_Touches` function from the `PostGIS` library.

2.1.0 `ST_Touches` returns `TRUE` if the two rasters touch at any point, `FALSE` if they do not, and `NULL` if either raster is `NULL`.

### Examples

```
SELECT r1.rid, r2.rid, ST_Touches(r1.rast, 1, r2.rast, 1) FROM dummy_rast r1 CROSS JOIN
  dummy_rast r2 WHERE r1.rid = 2;
```

| rid | rid | st_touches |
|-----|-----|------------|
| 2   | 1   | f          |
| 2   | 2   | f          |

### ST\_Intersects

## 12.17.9 ST\_SameAlignment

ST\_SameAlignment — Returns true if the two input geometries have the same alignment. The alignment is defined by the SRID, the bounding box, and the bounding box of the geometry. The bounding box of the geometry is defined by the bounding box of the geometry and the bounding box of the geometry.

### Synopsis

boolean **ST\_SameAlignment**( raster rastA , raster rastB );

boolean **ST\_SameAlignment**( double precision ulx1 , double precision uly1 , double precision scalex1 , double precision scaley1 , double precision skewx1 , double precision skewy1 , double precision ulx2 , double precision uly2 , double precision scalex2 , double precision scaley2 , double precision skewx2 , double precision skewy2 );

boolean **ST\_SameAlignment**( raster set rastfield );

### Parameters

**rastA**, **rastB**: Raster objects to be compared.  
**ulx1**, **uly1**, **scalex1**, **scaley1**, **skewx1**, **skewy1**: Bounding box and scale of the first raster.  
**ulx2**, **uly2**, **scalex2**, **scaley2**, **skewx2**, **skewy2**: Bounding box and scale of the second raster.  
**set rastfield**: A raster set containing the rasters to be compared.  
**SRID**: The SRID of the rasters.  
**NOTICE**: A notice is issued if the rasters have different SRIDs.

**2.0.0**: The function was introduced in PostGIS 2.0.0. It was previously implemented in the `ST_SameAlignment` function in the `postgis` extension.  
**2.1.0**: The function was updated to support the `ST_SameAlignment` function in the `postgis` extension.  
**3.0.0**: The function was updated to support the `ST_SameAlignment` function in the `postgis` extension.

**2.0.0**: The function was introduced in PostGIS 2.0.0. It was previously implemented in the `ST_SameAlignment` function in the `postgis` extension.

**2.1.0**: The function was updated to support the `ST_SameAlignment` function in the `postgis` extension.

### Examples

```
SELECT ST_SameAlignment (
  ST_MakeEmptyRaster (1, 1, 0, 0, 1, 1, 0, 0),
  ST_MakeEmptyRaster (1, 1, 0, 0, 1, 1, 0, 0)
) as sm;

sm
----
t
```

```
SELECT ST_SameAlignment(A.rast,b.rast)
FROM dummy_rast AS A CROSS JOIN dummy_rast AS B;

NOTICE: The two rasters provided have different SRIDs
NOTICE: The two rasters provided have different SRIDs
st_samealignment
-----
t
f
f
f
```

**Section 11.1, [ST\\_NotSameAlignmentReason](#), [ST\\_MakeEmptyRaster](#)**

Section 11.1, [ST\\_NotSameAlignmentReason](#), [ST\\_MakeEmptyRaster](#)

### 12.17.10 ST\_NotSameAlignmentReason

**ST\_NotSameAlignmentReason** — Returns the reason why two rasters do not have the same alignment. The function returns a text value that is one of the following: 't' (true), 'f' (false), 's' (same), 'a' (aligned), 'b' (border), 'c' (corner), 'd' (different), 'e' (empty), 'f' (false), 'g' (grid), 'h' (horizontal), 'i' (integer), 'j' (jagged), 'k' (key), 'l' (label), 'm' (margin), 'n' (name), 'o' (origin), 'p' (padding), 'q' (quarter), 'r' (radius), 's' (same), 't' (true), 'u' (unit), 'v' (value), 'w' (width), 'x' (x-axis), 'y' (y-axis), 'z' (z-axis).

#### Synopsis

text **ST\_NotSameAlignmentReason**(raster rastA, raster rastB);

**Notes**

The function returns a text value that is one of the following: 't' (true), 'f' (false), 's' (same), 'a' (aligned), 'b' (border), 'c' (corner), 'd' (different), 'e' (empty), 'f' (false), 'g' (grid), 'h' (horizontal), 'i' (integer), 'j' (jagged), 'k' (key), 'l' (label), 'm' (margin), 'n' (name), 'o' (origin), 'p' (padding), 'q' (quarter), 'r' (radius), 's' (same), 't' (true), 'u' (unit), 'v' (value), 'w' (width), 'x' (x-axis), 'y' (y-axis), 'z' (z-axis).



#### Note

The function returns a text value that is one of the following: 't' (true), 'f' (false), 's' (same), 'a' (aligned), 'b' (border), 'c' (corner), 'd' (different), 'e' (empty), 'f' (false), 'g' (grid), 'h' (horizontal), 'i' (integer), 'j' (jagged), 'k' (key), 'l' (label), 'm' (margin), 'n' (name), 'o' (origin), 'p' (padding), 'q' (quarter), 'r' (radius), 's' (same), 't' (true), 'u' (unit), 'v' (value), 'w' (width), 'x' (x-axis), 'y' (y-axis), 'z' (z-axis).

2.1.0 **ST\_SameAlignment**(raster rastA, raster rastB);

**Notes**

```
SELECT
  ST_SameAlignment (
    ST_MakeEmptyRaster (1, 1, 0, 0, 1, 1, 0, 0),
    ST_MakeEmptyRaster (1, 1, 0, 0, 1.1, 1.1, 0, 0)
  ),
  ST_NotSameAlignmentReason (
    ST_MakeEmptyRaster (1, 1, 0, 0, 1, 1, 0, 0),
    ST_MakeEmptyRaster (1, 1, 0, 0, 1.1, 1.1, 0, 0)
  )
;
```

```

st_samealignment | st_notsamealignmentreason
-----+-----
f | The rasters have different scales on the X axis
(1 row)
    
```

**ST\_SameAlignment**

Section 11.1, [ST\\_SameAlignment](#)

**12.17.11 ST\_Within**

**ST\_Within** — Returns true if the geometry of `rastA` is within the geometry of `rastB`. Both `rastA` and `rastB` must be of the same data type (e.g., both `float4` or both `float8`). Both `rastA` and `rastB` must be of the same SRID. `rastA` must be a `POINT`, `POLYGON`, or `MULTIPOLYGON`. `rastB` must be a `POLYGON` or `MULTIPOLYGON`. `rastA` must be a `POINT`, `POLYGON`, or `MULTIPOLYGON`. `rastB` must be a `POLYGON` or `MULTIPOLYGON`. `rastA` must be a `POINT`, `POLYGON`, or `MULTIPOLYGON`. `rastB` must be a `POLYGON` or `MULTIPOLYGON`.

**Synopsis**

boolean **ST\_Within**( raster `rastA` , integer `nbandA` , raster `rastB` , integer `nbandB` );  
 boolean **ST\_Within**( raster `rastA` , raster `rastB` );

**ST\_Within(rastA, rastB)**

`ST_Within(rastA, rastB)` returns true if the geometry of `rastA` is within the geometry of `rastB`. Both `rastA` and `rastB` must be of the same data type (e.g., both `float4` or both `float8`). Both `rastA` and `rastB` must be of the same SRID. `rastA` must be a `POINT`, `POLYGON`, or `MULTIPOLYGON`. `rastB` must be a `POLYGON` or `MULTIPOLYGON`. `rastA` must be a `POINT`, `POLYGON`, or `MULTIPOLYGON`. `rastB` must be a `POLYGON` or `MULTIPOLYGON`.



**Note**

`ST_Within(rastA, rastB)` returns true if the geometry of `rastA` is within the geometry of `rastB`. Both `rastA` and `rastB` must be of the same data type (e.g., both `float4` or both `float8`). Both `rastA` and `rastB` must be of the same SRID. `rastA` must be a `POINT`, `POLYGON`, or `MULTIPOLYGON`. `rastB` must be a `POLYGON` or `MULTIPOLYGON`.



**Note**

`ST_Within(ST_Polygon(raster), geometry)` returns true if the geometry of `raster` is within the geometry of `geometry`. Both `raster` and `geometry` must be of the same data type (e.g., both `float4` or both `float8`). Both `raster` and `geometry` must be of the same SRID. `raster` must be a `POINT`, `POLYGON`, or `MULTIPOLYGON`. `geometry` must be a `POLYGON` or `MULTIPOLYGON`.



**Note**

`ST_Within(rastA, rastB)` returns true if the geometry of `rastA` is within the geometry of `rastB`. Both `rastA` and `rastB` must be of the same data type (e.g., both `float4` or both `float8`). Both `rastA` and `rastB` must be of the same SRID. `rastA` must be a `POINT`, `POLYGON`, or `MULTIPOLYGON`. `rastB` must be a `POLYGON` or `MULTIPOLYGON`.

2.1.0 `ST_Within(rastA, rastB)` returns true if the geometry of `rastA` is within the geometry of `rastB`. Both `rastA` and `rastB` must be of the same data type (e.g., both `float4` or both `float8`). Both `rastA` and `rastB` must be of the same SRID. `rastA` must be a `POINT`, `POLYGON`, or `MULTIPOLYGON`. `rastB` must be a `POLYGON` or `MULTIPOLYGON`.

**ST\_DWithin**

```
SELECT r1.rid, r2.rid, ST_Within(r1.rast, 1, r2.rast, 1) FROM dummy_rast r1 CROSS JOIN
  dummy_rast r2 WHERE r1.rid = 2;
```

```
rid | rid | st_within
-----+-----+-----
  2 |  1 | f
  2 |  2 | t
```

**ST\_Contains**

[ST\\_Intersects](#), [ST\\_Contains](#), [ST\\_DWithin](#), [ST\\_DFullyWithin](#)

**12.17.12 ST\_DWithin**

**ST\_DWithin** — Returns true if the two rasters intersect within a specified distance. The distance is measured in raster units. The distance is measured from the center of each pixel to the center of the other pixel. The distance is measured from the center of each pixel to the center of the other pixel.

**Synopsis**

boolean **ST\_DWithin**( raster rastA , integer nbandA , raster rastB , integer nbandB , double precision distance\_of\_srid );  
 boolean **ST\_DWithin**( raster rastA , raster rastB , double precision distance\_of\_srid );

**ST\_DFullyWithin**

Returns true if the two rasters intersect within a specified distance. The distance is measured in raster units. The distance is measured from the center of each pixel to the center of the other pixel. The distance is measured from the center of each pixel to the center of the other pixel.

**ST\_DFullyWithin** — Returns true if the two rasters intersect within a specified distance. The distance is measured in raster units. The distance is measured from the center of each pixel to the center of the other pixel. The distance is measured from the center of each pixel to the center of the other pixel.

**Note**

**ST\_DFullyWithin**( raster rastA , integer nbandA , raster rastB , integer nbandB , double precision distance\_of\_srid );  
**ST\_DFullyWithin**( raster rastA , raster rastB , double precision distance\_of\_srid );

**Note**

**ST\_DWithin**( raster rastA , integer nbandA , raster rastB , integer nbandB , double precision distance\_of\_srid );  
**ST\_DWithin**( raster rastA , raster rastB , double precision distance\_of\_srid );

2.1.0 **ST\_DFullyWithin**( raster rastA , integer nbandA , raster rastB , integer nbandB , double precision distance\_of\_srid );



**ST\_DFullyWithin**

```
SELECT r1.rid, r2.rid, ST_DFullyWithin(r1.rast, 1, r2.rast, 1, 3.14) FROM dummy_rast r1
  CROSS JOIN dummy_rast r2 WHERE r1.rid = 2;
```

| rid | rid | st_dfullywithin |
|-----|-----|-----------------|
| 2   | 1   | f               |
| 2   | 2   | t               |

**ST\_Within, ST\_DWithin**

[ST\\_Within](#), [ST\\_DWithin](#)

## 12.18 Raster Tips

### 12.18.1 Out-DB Rasters

#### 12.18.1.1 Directory containing many files

When GDAL opens a file, GDAL eagerly scans the directory of that file to build a catalog of other files. If this directory contains many files (e.g. thousands, millions), opening that file becomes extremely slow (especially if that file happens to be on a network drive such as NFS).

To control this behavior, GDAL provides the following environment variable: `GDAL_DISABLE_READDIR_ON_OPEN`. Set `GDAL_DISABLE_READDIR_ON_OPEN` to `TRUE` to disable directory scanning.

In Ubuntu (and assuming you are using PostgreSQL's packages for Ubuntu), `GDAL_DISABLE_READDIR_ON_OPEN` can be set in `/etc/postgresql/POSTGRESQL_VERSION/CLUSTER_NAME/environment` (where `POSTGRESQL_VERSION` is the version of PostgreSQL, e.g. 9.6 and `CLUSTER_NAME` is the name of the cluster, e.g. maindb). You can also set PostGIS environment variables here as well.

```
# environment variables for postmaster process
# This file has the same syntax as postgresql.conf:
# VARIABLE = simple_value
# VARIABLE2 = 'any value!'
# I. e. you need to enclose any value which does not only consist of letters,
# numbers, and '-', '_', '.' in single quotes. Shell commands are not
# evaluated.
POSTGIS_GDAL_ENABLED_DRIVERS = 'ENABLE_ALL'

POSTGIS_ENABLE_OUTDB_RASTERS = 1

GDAL_DISABLE_READDIR_ON_OPEN = 'TRUE'
```

#### 12.18.1.2 Maximum Number of Open Files

The maximum number of open files permitted by Linux and PostgreSQL are typically conservative (typically 1024 open files per process) given the assumption that the system is consumed by human users. For Out-DB Rasters, a single valid query can easily exceed this limit (e.g. a dataset of 10 year's worth of rasters with one raster for each day containing minimum and maximum temperatures and we want to know the absolute min and max value for a pixel in that dataset).

The easiest change to make is the following PostgreSQL setting: `max_files_per_process`. The default is set to 1000, which is far too low for Out-DB Rasters. A safe starting value could be 65536 but this really depends on your datasets and the queries run against those datasets. This setting can only be made on server start and probably only in the PostgreSQL configuration file (e.g. `/etc/postgresql/POSTGRESQL_VERSION/CLUSTER_NAME/postgresql.conf` in Ubuntu environments).

```
...
# - Kernel Resource Usage -

max_files_per_process = 65536          # min 25
                                       # (change requires restart)
...
```

The major change to make is the Linux kernel's open files limits. There are two parts to this:

- Maximum number of open files for the entire system
- Maximum number of open files per process

#### 12.18.1.2.1 Maximum number of open files for the entire system

You can inspect the current maximum number of open files for the entire system with the following example:

```
$ sysctl -a | grep fs.file-max
fs.file-max = 131072
```

If the value returned is not large enough, add a file to `/etc/sysctl.d/` as per the following example:

```
$ echo "fs.file-max = 6145324" >> /etc/sysctl.d/fs.conf

$ cat /etc/sysctl.d/fs.conf
fs.file-max = 6145324

$ sysctl -p --system
* Applying /etc/sysctl.d/fs.conf ...
fs.file-max = 2097152
* Applying /etc/sysctl.conf ...

$ sysctl -a | grep fs.file-max
fs.file-max = 6145324
```

#### 12.18.1.2.2 Maximum number of open files per process

We need to increase the maximum number of open files per process for the PostgreSQL server processes.

To see what the current PostgreSQL service processes are using for maximum number of open files, do as per the following example (make sure to have PostgreSQL running):

```
$ ps aux | grep postgres
postgres 31713  0.0  0.4 179012 17564 pts/0    S   Dec26   0:03 /home/dustymugs/devel/ ↵
    postgresql/sandbox/10/usr/local/bin/postgres -D /home/dustymugs/devel/postgresql/sandbox ↵
    /10/pgdata
postgres 31716  0.0  0.8 179776 33632 ?        Ss  Dec26   0:01 postgres: checkpointer ↵
    process
postgres 31717  0.0  0.2 179144  9416 ?        Ss  Dec26   0:05 postgres: writer process
postgres 31718  0.0  0.2 179012  8708 ?        Ss  Dec26   0:06 postgres: wal writer ↵
    process
postgres 31719  0.0  0.1 179568  7252 ?        Ss  Dec26   0:03 postgres: autovacuum ↵
    launcher process
postgres 31720  0.0  0.1  34228  4124 ?        Ss  Dec26   0:09 postgres: stats collector ↵
    process
postgres 31721  0.0  0.1 179308  6052 ?        Ss  Dec26   0:00 postgres: bgworker: ↵
    logical replication launcher

$ cat /proc/31718/limits
```

| Limit                 | Soft Limit  | Hard Limit  | Units        |
|-----------------------|-------------|-------------|--------------|
| Max cpu time          | unlimited   | unlimited   | seconds      |
| Max file size         | unlimited   | unlimited   | bytes        |
| Max data size         | unlimited   | unlimited   | bytes        |
| Max stack size        | 8388608     | unlimited   | bytes        |
| Max core file size    | 0           | unlimited   | bytes        |
| Max resident set      | unlimited   | unlimited   | bytes        |
| Max processes         | 15738       | 15738       | processes    |
| <b>Max open files</b> | <b>1024</b> | <b>4096</b> | <b>files</b> |
| Max locked memory     | 65536       | 65536       | bytes        |
| Max address space     | unlimited   | unlimited   | bytes        |
| Max file locks        | unlimited   | unlimited   | locks        |
| Max pending signals   | 15738       | 15738       | signals      |
| Max msgqueue size     | 819200      | 819200      | bytes        |
| Max nice priority     | 0           | 0           |              |
| Max realtime priority | 0           | 0           |              |
| Max realtime timeout  | unlimited   | unlimited   | us           |

In the example above, we inspected the open files limit for Process 31718. It doesn't matter which PostgreSQL process, any of them will do. The response we are interested in is *Max open files*.

We want to increase *Soft Limit* and *Hard Limit* of *Max open files* to be greater than the value we specified for the PostgreSQL setting `max_files_per_process`. In our example, we set `max_files_per_process` to 65536.

In Ubuntu (and assuming you are using PostgreSQL's packages for Ubuntu), the easiest way to change the *Soft Limit* and *Hard Limit* is to edit `/etc/init.d/postgresql` (SysV) or `/lib/systemd/system/postgresql*.service` (systemd).

Let's first address the SysV Ubuntu case where we add `ulimit -H -n 262144` and `ulimit -n 131072` to `/etc/init.d/postgresql`.

```
...
case "$1" in
  start|stop|restart|reload)
    if [ "$1" = "start" ]; then
      create_socket_directory
    fi
    if [ -z "`pg_lsclusters -h`" ]; then
      log_warning_msg 'No PostgreSQL clusters exist; see "man pg_createcluster"'
      exit 0
    fi

    ulimit -H -n 262144
    ulimit -n 131072

    for v in $versions; do
      $1 $v || EXIT=$?
    done
    exit ${EXIT:-0}
    ;;
  status)
...

```

Now to address the systemd Ubuntu case. We will add `LimitNOFILE=131072` to every `/lib/systemd/system/postgresql*.service` file in the `[Service]` section.

```
...
[Service]

LimitNOFILE=131072

...

[Install]
WantedBy=multi-user.target

```

...

After making the necessary systemd changes, make sure to reload the daemon

```
systemctl daemon-reload
```

---

## Chapter 13

# PostGIS Raster: FAQ

1. *PostGIS Raster: FAQ* (Jorge Arévalo) (2013). [PostGIS Raster / Oracle GeoRaster Series](#). PostGIS in Action - Raster chapter. [PostGIS raster applied to land classification urban forestry](#).

[PostGIS Raster: FAQ](#) (2013). [PostGIS raster applied to land classification urban forestry](#).

2. *PostGIS Raster: FAQ* (Jorge Arévalo) (2013). [PostGIS Raster / Oracle GeoRaster Series](#). PostGIS in Action - Raster chapter. [PostGIS raster applied to land classification urban forestry](#). Here is a review from a PostGIS Raster user [PostGIS raster applied to land classification urban forestry](#).
3. *PostGIS Raster* (Jorge Arévalo) (2013). [PostGIS Raster / Oracle GeoRaster Series](#). PostGIS in Action - Raster chapter. [PostGIS raster applied to land classification urban forestry](#).

PostGIS Raster is part of the PostGIS codebase and generally available with most PostGIS binary distributions. Starting with PostGIS 3.0, PostGIS raster is now a separate extension and requires: `CREATE EXTENSION postgis_raster;` to enable it in your database. If you are compiling your own PostGIS, you will need to compile with GDAL otherwise `postgis_raster` extension will not be built. Refer to [Download PostGIS binaries](#) for popular distributions of PostGIS that include raster support.

4. *PostGIS Raster: FAQ* (Jorge Arévalo) (2013). [PostGIS Raster / Oracle GeoRaster Series](#). PostGIS in Action - Raster chapter. [PostGIS raster applied to land classification urban forestry](#).

The latest version of PostGIS comes packaged with a `raster2pgsql` raster loader executable capable of loading many kinds of rasters and also generating lower resolution overviews without any additional software. Please refer to [Section 11.1.1](#) for more details.

- 5. &#xc0ac;&#xc6a9;&#xc790; &#xb370;&#xc774;&#xd130;&#xbca0;&#xc774;&#xc2a4;&#xc5d0; &#xc5b4;&#xb5a4;&#xc885;&#xb958;&#xc758; &#xb798;&#xc2a4;&#xd130; &#xd30c;&#xc77c; &#xd615;&#xc2dd;&#xc744; &#xb85c;&#xb4&#xc218; &#xc788;&#xc2b5;&#xb2c8;&#xae4c;?

&#xc0ac;&#xc6a9;&#xc790;&#xc758; GDAL &#xb77c;&#xc774;&#xbe0c;&#xb7ec;&#xb9ac;&#xac00; &#xc9c0;&#xc6d0;&#xbaa8;&#xb4e0; &#xd615;&#xc2dd;&#xc744; &#xb85c;&#xb4dc;&#xd560; &#xc218; &#xc788;&#xc2b5;&#xb2c8;&#xb2 GDAL&#xc774; &#xc9c0;&#xc6d0;&#xd558;&#xb294; &#xd615;&#xc2dd;&#xc740; **GDAL File Formats** &#xc5d0; &#xc798; &#xc815;&#xb9ac;&#xb3fc; &#xc788;&#xc2b5;&#xb2c8;&#xb2e4;. &#xc0ac;&#xc6a9;&#xc790;&#xac00; &#xc124;&#xce58;&#xd55c; &#xd2b9;&#xc815; GDAL&#xc774; &#xbaa8;&#xb4e0; &#xd615;&#xc2dd;&#xc744; &#xc9c0;&#xc6d0;&#xd558;&#xc9c0; &#xc54a;&#xc744; &#xc218;&#xb3c4; &#xc788;&#xc2b5;&#xb2c8;&#xb2e4;. &#xc0ac;&#xc6a9;&#xc790;&#xc758; &#xd2b9;&#xc815; GDAL&#xc774; &#xc9c0;&#xc6d0;&#xd558;&#xb294; &#xd615; &#xd655;&#xc778;&#xd558;&#xb824;&#xba74;. &#xb2e4;&#xc74c; &#xba85;&#xb839;&#xc5b4;&#xb97c; &#xc2e4;&#xd5

```
raster2pgsql -G
```

- 6. &#xb0b4; PostGIS &#xb798;&#xc2a4;&#xd130; &#xb370;&#xc774;&#xd130;&#xb97c; &#xb2e4;&#xb978; &#xb798;&#xc2a4;&#xd615;&#xc2dd;&#xc73c;&#xb85c; &#xb0b4;&#xbcf4;&#xb0b4;&#xae30;&#xd560; &#xc218; &#xc788;&#xc2b5;&#xb2c8;

&#xb124;. PostGIS raster has a function **ST\_AsGDALRaster** that will allow you to use SQL to export to any raster format supported by your GDAL. You can get a list of these using the **ST\_GDALDrivers** SQL function. You can also use GDAL commandline tools to export PostGIS raster to other formats. GDAL has a PostGIS raster driver, but is only compiled in if you choose to compile with PostgreSQL support. &#xc0ac;&#xc6a9;&#xc790;&#xac00; &#xbe44;&#xc815;&#xaddc;&#xc801; &#xbe14;&#xb85d;&#xd654;&#xb41c; &#xb798;&#xc2a4;&#xd130;&#xb97c; PostGIS &#xb798;&#xc2a4;&#xd130; &#xb370;&#xc774;&#xd130;&#xd615;&#xc73c;&#xb85c; &#xc800;&#xc7a5;&#xd560; &#xc218; &#xc788;&#xc74c;&#xc5 &#xbd88;&#xad6c;&#xd558;&#xae0;. &#xd574;&#xb2f9; &#xb4dc;&#xb77c;&#xc774;&#xbc84;&#xb294; &#xd604;&#xc7 &#xbe44;&#xc815;&#xaddc;&#xc801;&#xc73c;&#xb85c; &#xbe14;&#xb85d;&#xd654;&#xb41c; &#xb798;&#xc2a4;&#xd13 &#xc9c0;&#xc6d0;&#xd558;&#xc9c0; &#xc54a;&#xc2b5;&#xb2c8;&#xb2e4;. &#xc0ac;&#xc6a9;&#xc790;&#xac00; &#xc18 &#xcfe4;&#xd30c;&#xc77c;&#xd560; &#xacbd;&#xc6b0;. &#xb4dc;&#xb77c;&#xc774;&#xbc84;&#xb97c; &#xd65c;&#xc1 &#xc0ac;&#xc6a9;&#xc790; &#xc124;&#xc815;&#xc5d0;

```
--with-pg=path/to/pg_config
```

&#xb97c; &#xd3ec;&#xd568;&#xc2dc;&#xcdf1c;&#xc57c; &#xd569;&#xb2c8;&#xb2e4;. &#xc5ec;&#xb7ec; OS &#xd50c;&# &#xb300;&#xd574; GDAL&#xc744; &#xbe4c;&#xb4dc;&#xd558;&#xb294; &#xb370; &#xb300;&#xd55c; &#xbc29;&#xbce9 &#xc54c;&#xae0; &#xc2f6;&#xb2e4;&#xba74; **GDAL Build Hints** &#xb97c; &#xcc38;&#xc870;&#xd558;&#xc2ed;&#xc2dc &#xc0ac;&#xc6a9;&#xc790;&#xc758; GDAL &#xbc84;&#xc804;&#xc744; PostGIS &#xb798;&#xc2a4;&#xd130; &#xb4dc;&# &#xd568;&#xaed8; &#xcfe4;&#xd30c;&#xc77c;&#xd588;&#xb2e4;&#ba74; &#xb2e4;&#xc74c; &#xba85;&#xb839;&#xc5b &#xc2e4;&#xd589;&#xd588;&#xc744; &#xb54c; &#xbaa9;&#xb85d;&#xc5d0; PostGIS &#xb798;&#xc2a4;&#xd130;&#xac0 &#xbcf4;&#xc5ec;&#xc57c; &#xd569;&#xb2c8;&#xb2e4;:

```
gdalinfo --formats
```

gdalinfo&#xb97c; &#xc774;&#xc6a9;&#xd574;&#xc11c; GDAL&#xc744; &#xd1b5;&#xd574; &#xc0ac;&#xc6a9;&#xc790; &#xb798;&#xc2a4;&#xd130;&#xc5d0; &#xb300;&#xd55c; &#xc694;&#xc57d;&#xc744; &#xbcf4;&#xb824;&#ba74; &#xb2e4;&#xc74c; &#ba85;&#xb839;&#xc5b4;&#xb97c; &#xc2e4;&#xd589;&#xd558;&#xc2ed;&#xc2dc;&#xc624;:

```
gdalinfo "PG:host=localhost port=5432 dbname='mygisdb' user='postgres' password=' ← whatever' schema='someschema' table=sometable"
```

&#xb370;&#xc774;&#xd130;&#xb97c; &#xb2e4;&#xb978; &#xb798;&#xc2a4;&#xd130; &#xd615;&#xc2dd;&#xc73c;&#xb8 &#xb0b4;&#xbcf4;&#xb0b4;&#xae30;&#xd558;&#xb824;&#ba74;. gdal\_translate&#xb97c; &#xc774;&#xc6a9;&#xd558;&# &#xb2e4;&#xc74c; &#ba85;&#xb839;&#xc5b4;&#xb294; &#xd14c;&#xc774;&#xbe14;&#xc5d0;&#xc11c; &#baa8;&#xb4 &#xb370;&#xc774;&#xd130;&#xb97c; 10% &#xd06c;&#xae30;&#xc758; PNG &#xd30c;&#xc77c;&#xb85c; &#xb0b4;&#xb &#xac83;&#xc785;&#xb2c8;&#xb2e4;. &#xc0ac;&#xc6a9;&#xc790;&#xc758; &#xd53d;&#xc140; &#xbc34;&#xb4dc; &#xc720;&#xd615;&#xc5d0; &#xb530;&#xb77c;. &#xb0b4;&#xbcf4;&#xb0b4;&#xae30;&#xd560; &#xd615;&#xc2dd;&#xc7 &#xd574;&#xb2f9; &#xd53d;&#xc140; &#xc720;&#xd615;&#xc744; &#xc9c0;&#xc6d0;&#xd558;&#xc9c0; &#xc54a;&#xc7 &#xacbd;&#xc6b0; gdal\_translate&#xc774; &#xc791;&#xb3d9;&#xd558;&#xc9c0; &#xc54a;&#xc744; &#xc218;&#xb3c4; &#xc788;&#xc2b5;&#xb2c8;&#xb2e4;. &#xc608;&#xb97c; &#xb4e4;&#xc5b4; &#xbd80;&#xb3d9;&#xc18c;&#xc218;&#xc8 &#xbc34;&#xb4dc; &#xc720;&#xd615; &#xbcof;&#xbd80;&#xd638;&#xac00; &#xc5c6;&#xb294; 32&#xbe44;&#xd2b8; &#xc815;&#xc218;&#xd615;&#xc740; JPG&#xb610;&#xb294; &#xba87;&#xba87; &#xb2e4;&#xb978; &#xd615;&#xc2dd;&

```
gdal_translate -of PNG -outsize 10% 10% "PG:host=localhost port=5432 dbname='mygisdb'
user='postgres' password='whatever' schema='someschema' table=sometable" C:\
somefile.png
```

```
gdal_translate -of PNG -outsize 10% 10% "PG:host=localhost port=5432 dbname='mygisdb'
user='postgres' password='whatever' schema='someschema' table=sometable where='
filename='abcd.sid'" " C:\somefile.png
```

```
gdal_translate -of PNG -outsize 10% 10% "PG:host=localhost port=5432 dbname='mygisdb'
user='postgres' password='whatever' schema='someschema' table=sometable where='
ST_Intersects(rast, ST_SetSRID(ST_Point(-71.032, 42.3793), 4326) )" " C:\
intersectregion.png
```

```
gdal_translate -of PNG -outsize 10% 10% "PG:host=localhost port=5432 dbname='mygisdb'
user='postgres' password='whatever' schema='someschema' table=sometable where='
ST_Intersects(rast, ST_SetSRID(ST_Point(-71.032, 42.3793), 4326) )" " C:\
intersectregion.png
```

```
gdal_translate -of PNG -outsize 10% 10% "PG:host=localhost port=5432 dbname='mygisdb'
user='postgres' password='whatever' schema='someschema' table=sometable where='
ST_Intersects(rast, ST_SetSRID(ST_Point(-71.032, 42.3793), 4326) )" " C:\
intersectregion.png
```

Reading Raster Data of PostGIS Raster section

- GDAL Binaries

PostgreSQL, GDAL, PostGIS, MapServer, Visual Studio, Nightly, SDK, VS, and MapServer compiled with GDAL to view Raster data. QGIS supports viewing of PostGIS Raster if you have PostGIS raster driver installed.

- You can use MapServer compiled with GDAL to view Raster data. QGIS supports viewing of PostGIS Raster if you have PostGIS raster driver installed.

MapServer compiled with GDAL to view Raster data. QGIS supports viewing of PostGIS Raster if you have PostGIS raster driver installed.

- MapServer compiled with GDAL to view Raster data. QGIS supports viewing of PostGIS Raster if you have PostGIS raster driver installed.

GDAL 1.7 &#xc774;&#xc0c1; &#xb84;&#xc804;&#xc774; &#xd544;&#xc694;&#xd569;&#xb2c8;&#xb2e4;. GDAL 1.8 &#xb9ce;&#xc740; &#xbb38;&#xc81c;&#xc810;&#xb4e4;&#xc774; &#xd574;&#xb54c;&#xb38;&#xc5d0; 1.8 &#xc774;&#xc0c1; &#xb84;&#xc804;&#xc744; &#xc4f0;&#xb294; &#xd3b8;&#xc774; &#xc88b;&#xc2b5;&#xb2c8;&#xb2e4;. &#xd2b8;&#xb801;&#xd06c; &#xb84;&#xc804;&#xc5d0;&#xc11c;&#xb294; &#xb354; &#xb9ce;&#xc740; PostGIS &#xb798;&#xc2a4;&#xd130; &#xbb38;&#xc81c;&#xc810;&#xb4e4;&#xc774; &#xd574;&#xacb0;&#xb410;&#xc2b5;&#xb2c8;&#xb2e4;. &#xc5b4;&#xb5a4; &#xb2e4;&#xb978; &#xb798;&#xc2a4;&#xd130; &#xb9c8;&#xc2c2;&#xac00;&#xc9c0;&#xb85c; &#xcd94;&#xac00;&#xd560; &#xc218; &#xc788;&#xc2b5;&#xb2c8;&#xb2e4; MapServer &#xb798;&#xc2a4;&#xd130; &#xb808;&#xc774;&#xc5b4;&#xc640; &#xd568;&#xae8; &#xc774;&#xc6a9;&#xd130; &#xc218; &#xc788;&#xb294; &#xc5ec;&#xb7ec; &#xacf5;&#xac04; &#xc98;&#xb9ac; &#xd568;&#xc218;&#xb4e4;&#xc774; &#xaa9;&#xb85d;&#xc744; &#xbcf4;&#xb824;&#ba74; **MapServer Raster processing options** &#xb97c; &#xc38;&#xc870; &#xb798;&#xc2a4;&#xd130; &#xb370;&#xc774;&#xd130;&#xb97c; &#xd2b9;&#xd788; &#xd765;&#xbbf8;&#xb86d;&#xac00; &#xb9cc;&#xb4dc;&#xb294; &#xc810;&#xc740;. &#xac01; &#xd0c0;&#xc77c;&#xc774; &#xb2e4;&#xc591;&#xd55c; &#xd45c;&#xc900; &#xb370;&#xc774;&#xd130;&#xbca0;&#xc774;&#xc2a4; &#xc5f4;&#xb4e4;&#xc744; &#xac00;&#xc9c0; &#xc218; &#xc788;&#xae30; &#xb54c;&#bb38;&#xc5d0; &#xc0ac;&#xc6a9;&#xc790; &#xb370;&#xc774;&#xd130;&#xc11c; &#xd0c0;&#xc77c;&#xc744; &#xbd80;&#bd84;&#xc73c;&#xb85c; &#xb098;&#xb20c; &#xc218; &#xc788;&#xb2e4;&#xb2c8;&#xc810;&#xc785;&#xb2c8;&#xb2e4;. &#xb2e4;&#xc74c;&#xc740; MapServer&#xc5d0;&#xc11c; PostGIS &#xb798;&#xc2a4;&#xb808;&#xc774;&#xc5b4;&#xb97c; &#xc5b4;&#xb5bb;&#xac8c; &#xc815;&#xc758;&#xd558;&#xb294;&#xac00;&#xc5d0;&#xb300;&#xd55c; &#xc608;&#xc2dc;&#xc785;&#xb2c8;&#xb2e4;.

**Note**

&#xd0c0;&#xc77c;&#xd654;&#xb41c; &#xb798;&#xc2a4;&#xd130;&#xc758; &#xacbd;&#xc6b0; mode=2 &#xc124;&#xc815;&#xc774; &#xd544;&#xc694;&#xd55c;&#xb370;. &#xc774; &#xc124;&#xc815;&#xc740; PostGIS 2.0 &#xb0f; GDAL 1.8 &#xb4dc;&#xb77c;&#xc774;&#xb84;&#xc5d0; &#xcd94;&#xac00;&#xb410;&#xc2b5;&#xb2c8;&#xb2e4;. &#xc774; &#xc124;&#xc815;&#xc740; GDAL 1.7 &#xb4dc;&#xb77c;&#xc774;&#xb84;&#xc5d0;&#xb294; &#xc874;&#xc7ac;&#xd558;&#xc9c0; &#xc54a;&#xc2b5;&#xb2c8;&#xb2e4;.

```
-- &#xb798;&#xc2a4;&#xd130;&#xb97c; &#xd45c;&#xc900; &#xb798;&#xc2a4;&#xd130; ←
  &#xc635;&#xc158;&#xc73c;&#xb85c; &#xd45c;&#xcd9c;&#xd558;&#xae30;
LAYER
  NAME coolwktraster
  TYPE raster
  STATUS ON
  DATA "PG:host=localhost port=5432 dbname='somedb' user='someuser' password=' ←
    whatever'
    schema='someschema' table='coolttable' mode='2'"
  PROCESSING "NODATA=0"
  PROCESSING "SCALE=AUTO"
  #... &#xb2e4;&#xb978; &#xd45c;&#xc900; &#xb798;&#xc2a4;&#xd130; ←
    &#xacf5;&#xac04; &#xc98;&#xb9ac; &#xd568;&#xc218;&#xb97c; &#xcd94;&#xac00;
  #... &#xd074;&#xb798;&#xc2a4;&#xb294; ←
    &#xc120;&#xd0dd;&#xc801;&#xc774;&#xc9c0;&#xb9cc; &#xb2e8;&#xc77c; ←
    &#xb34;&#xb4dc; &#xb370;&#xc774;&#xd130;&#xc5d0; &#xc720;&#xc6a9;&#xd568;
  CLASS
    NAME "boring"
    EXPRESSION ([pixel] < 20)
    COLOR 250 250 250
  END
  CLASS
    NAME "mildly interesting"
    EXPRESSION ([pixel] > 20 AND [pixel] < 1000)
    COLOR 255 0 0
  END
  CLASS
    NAME "very interesting"
    EXPRESSION ([pixel] >= 1000)
    COLOR 0 255 0
```

```

END
END

-- &#xb798;&#xc2a4;&#xd130;&#xb97c; &#xd45c;&#xc900; &#xb798;&#xc2a4;&#xd130; &#xc635;&#xc158; &#xbc0f; WHERE &#xc808;&#xb85c; &#xd45c;&#xcd9c;&#xd558;&#xae30;
LAYER
  NAME soil_survey2009
  TYPE raster
  STATUS ON
  DATA "PG:host=localhost port=5432 dbname='somedb' user='someuser' password='
  whatever'
  schema='someschema' table='cooltable' where='survey_year=2009' mode &#xc2e2;
  PROCESSING "NODATA=0"
  #... other standard raster processing functions here
  #... classes are optional but useful for 1 band data
END

```

10. [Chapter 12](#) [PostGIS &#xb798;&#xc2a4;&#xd130;&#xb97c; &#xd45c;&#xc900; &#xb798;&#xc2a4;&#xd130; &#xc635;&#xc158; &#xbc0f; WHERE &#xc808;&#xb85c; &#xd45c;&#xcd9c;&#xd558;&#xae30; &#xc624; &#xb958; &#xac00; &#xbc1c; &#xc0dd; &#xd569; &#xb2c8; &#xb2e4; . &#xc5b4; &#xb5bb; &#xac8c; &#xd574; &#xacbd; &#xc6b0; ;](#)

Chapter 12 [PostGIS &#xb798;&#xc2a4;&#xd130;&#xb97c; &#xd45c;&#xc900; &#xb798;&#xc2a4;&#xd130; &#xc635;&#xc158; &#xbc0f; WHERE &#xc808;&#xb85c; &#xd45c;&#xcd9c;&#xd558;&#xae30; &#xc624; &#xb958; &#xac00; &#xbc1c; &#xc0dd; &#xd569; &#xb2c8; &#xb2e4; . &#xc5b4; &#xb5bb; &#xac8c; &#xd574; &#xacbd; &#xc6b0; ;](#)

11. *ERROR: function st\_intersects(raster, unknown) is not unique or st\_union(geometry, text) is not unique* [PostGIS &#xb798;&#xc2a4;&#xd130;&#xb97c; &#xd45c;&#xc900; &#xb798;&#xc2a4;&#xd130; &#xc635;&#xc158; &#xbc0f; WHERE &#xc808;&#xb85c; &#xd45c;&#xcd9c;&#xd558;&#xae30; &#xc624; &#xb958; &#xac00; &#xbc1c; &#xc0dd; &#xd569; &#xb2c8; &#xb2e4; . &#xc5b4; &#xb5bb; &#xac8c; &#xd574; &#xacbd; &#xc6b0; ;](#)?

The function is not unique error happens if one of your arguments is a textual representation of a geometry instead of a geometry. In these cases, PostgreSQL marks the textual representation as an unknown type, which means it can fall into the `st_intersects(raster, geometry)` or `st_intersects(raster, raster)` thus resulting in a non-unique case since both functions can in theory support your request. To prevent this, you need to cast the textual representation of the geometry to a geometry. [PostGIS &#xb798;&#xc2a4;&#xd130;&#xb97c; &#xd45c;&#xc900; &#xb798;&#xc2a4;&#xd130; &#xc635;&#xc158; &#xbc0f; WHERE &#xc808;&#xb85c; &#xd45c;&#xcd9c;&#xd558;&#xae30; &#xc624; &#xb958; &#xac00; &#xbc1c; &#xc0dd; &#xd569; &#xb2c8; &#xb2e4; . &#xc5b4; &#xb5bb; &#xac8c; &#xd574; &#xacbd; &#xc6b0; ;](#)

```

SELECT rast
FROM my_raster
WHERE ST_Intersects(rast, 'SRID=4326;POINT(-10 10)');

```

[PostGIS &#xb798;&#xc2a4;&#xd130;&#xb97c; &#xd45c;&#xc900; &#xb798;&#xc2a4;&#xd130; &#xc635;&#xc158; &#xbc0f; WHERE &#xc808;&#xb85c; &#xd45c;&#xcd9c;&#xd558;&#xae30; &#xc624; &#xb958; &#xac00; &#xbc1c; &#xc0dd; &#xd569; &#xb2c8; &#xb2e4; . &#xc5b4; &#xb5bb; &#xac8c; &#xd574; &#xacbd; &#xc6b0; ;](#)

```

SELECT rast
FROM my_raster
WHERE ST_Intersects(rast, 'SRID=4326;POINT(-10 10)::geometry');

```

12. *PostGIS &#xb798;&#xc2a4;&#xd130;&#xb97c; &#xd45c;&#xc900; &#xb798;&#xc2a4;&#xd130; &#xc635;&#xc158; &#xbc0f; WHERE &#xc808;&#xb85c; &#xd45c;&#xcd9c;&#xd558;&#xae30; &#xc624; &#xb958; &#xac00; &#xbc1c; &#xc0dd; &#xd569; &#xb2c8; &#xb2e4; . &#xc5b4; &#xb5bb; &#xac8c; &#xd574; &#xacbd; &#xc6b0; ;* [Oracle GeoRaster and PostGIS Raster: First impressions](#) [PostGIS &#xb798;&#xc2a4;&#xd130;&#xb97c; &#xd45c;&#xc900; &#xb798;&#xc2a4;&#xd130; &#xc635;&#xc158; &#xbc0f; WHERE &#xc808;&#xb85c; &#xd45c;&#xcd9c;&#xd558;&#xae30; &#xc624; &#xb958; &#xac00; &#xbc1c; &#xc0dd; &#xd569; &#xb2c8; &#xb2e4; . &#xc5b4; &#xb5bb; &#xac8c; &#xd574; &#xacbd; &#xc6b0; ;](#)

```

&#xcee4;&#xbc84;&#xb9ac;&#xc9c0;&#xac00; &#xbc18;&#xb4dc;&#xc2dc; &#xc0ac;&#xac01;&#xd615;&#xc774; &#xc544;
&#xb429;&#xb2c8;&#xb2e4;. (&#xb113;&#xc740; &#xbc94;&#xc704;&#xb97c; &#xcee4;&#xbc84;&#xd558;&#xb294;
&#xb798;&#xc2a4;&#xd130; &#xcee4;&#xbc84;&#xb9ac;&#xc9c0;&#xb294; &#xc0ac;&#xac01;&#xd615;&#xc774;
&#xc544;&#xb2cc; &#xacbd;&#xc6b0;&#xac00; &#xb9ce;&#xc2b5;&#xb2c8;&#xb2e4;. &#xb798;&#xc2a4;&#xd130;&#xb9
&#xc98;&#xb9ac;&#xd560; &#xc218; &#xc788;&#xb294; &#bc29;&#xc2dd;&#xc5d0; &#xb300;&#xd574;&#xc11c;
&#xbb38;&#xc11c;&#xb97c; &#xcc3e;&#xc544;&#xbcf4;&#xc2ed;&#xc2dc;&#xc624;)* &#xb798;&#xc2a4;&#xd130;&#xb9
&#xc911;&#xc9a9;&#xc2dc;&#xd0ac; &#xc218; &#xc788;&#xc2b5;&#xb2c8;&#xb2e4;. (&#xbca1;&#xd130;&#xb97c;
&#xc190;&#xc2e4; &#xc5c6;&#xc774; &#xb798;&#xc2a4;&#xd130;&#xb85c; &#xbcc0;&#xd658;&#xc2dc;&#xd0a4;&#xb29
&#xb370; &#xd544;&#xc694;&#xd569;&#xb2c8;&#xb2e4;. &#xc624;&#xb77c;&#xd074;&#xc5d0;&#xc11c;&#xb3c4;
&#xc774;&#xb7f0; &#xcc98;&#xb9ac; &#bc29;&#xc2dd;&#xc744; &#xc4f8; &#xc218; &#xc788;&#xc9c0;&#xb9cc;
&#xac19;&#xc740; &#xac1c;&#xc218;&#xc758; SDO_RASTER &#xd14c;&#xc774;&#xbe14;&#xc5d0; &#xb9c1;&#xd06c;&
&#xbcf5;&#xc218;&#xc758; SDO_GEOASTER &#xac1d;&#xc6b4;&#xb97c; &#xc800;&#xc7a5;&#xd55c;&#xb2e4;&#xb2
&#xb73b;&#xb3c4; &#xb429;&#xb2c8;&#xb2e4;. &#bcf5;&#xc7a1; &#xbcc0;&#xd658; &#xc791;&#xc5c5;&#xc73c;&#xb8
&#xb370;&#xc774;&#xd130;&#xbca0;&#xc774;&#xc2a4;&#xc5d0; &#xc218;&#bc31; &#xac1c;&#xc758; &#xd14c;&#xc77
&#xc0dd;&#xc131;&#xd560; &#xc218;&#xb3c4; &#xc788;&#xc2b5;&#xb2c8;&#xb2e4;. PostGIS &#xb798;&#xc2a4;&#xd13
&#xc0ac;&#xc6a9;&#xd558;&#xba74;. &#be44;&#xc2b7;&#xd55c; &#xb798;&#xc2a4;&#xd130; &#xcc98;&#xb9ac;
&#bc29;&#xc2dd;&#xc744; &#xb2e8;&#xc77c; &#xd14c;&#xc774;&#xbe14;&#xc5d0; &#xc800;&#xc7a5;&#xd560;
&#xc218; &#xc788;&#xc2b5;&#xb2c8;&#xb2e4;. &#xc624;&#xb77c;&#xd074;&#xc758; &#xb798;&#xc2a4;&#xd130;
&#xc720;&#xd615;&#xc740; PostGIS &#xac00; &#xd2c8;(gap)&#xc774;&#xb098; &#xc911;&#xc9a9;&#xc774; &#xc5c6;&#
&#xc644;&#xc804;&#xd55c; &#xc0ac;&#xac01;&#xd615; &#bca1;&#xd130; &#xcee4;&#xbc84;&#xb9ac;&#xc9c0;(&#xc7
&#xc644;&#xbcbd;&#xd55c; &#xc0ac;&#xac01;&#xd615; &#xb808;&#xc774;&#xc5b4;)&#xb9cc; &#xc800;&#xc7a5;&#xd5
&#xac15;&#xc81c;&#xd558;&#xb294; &#xac83;&#xacfc; &#be44;&#xc2b7;&#xd569;&#xb2c8;&#xb2e4;. &#ba87;&#xba
&#xc2e4;&#xc81c; &#xc0ac;&#xb840;&#xc5d0;&#xc11c; &#xc774;&#xb294; &#xb9e4;&#xc6b0; &#xc2e4;&#xc6a9;&#xc8
&#xb300;&#xbd80;&#xbd84;&#xc758; &#xc9c0;&#xb9ac;&#xd559;&#xc801; &#xcee4;&#xbc84;&#xb9ac;&#xc9c0;&#xc75
&#xacbd;&#xc6b0; &#xadf8;&#xb2e4;&#xc9c0; &#xd604;&#xc2e4;&#xc801;&#xc774;&#xac70;&#xb098; &#bc14;&#xb78
&#xbcf; &#xc218; &#xc5c6;&#xc2b5;&#xb2c8;&#xb2e4;. &#bd88;&#xc5f0;&#xc18d;&#xc801;&#xc774;&#xace0;
&#xc0ac;&#xac01;&#xd615;&#xc774; &#xc544;&#xb2cc; &#xcee4;&#xbc84;&#xb9ac;&#xc9c0;&#xb97c; &#xc800;&#xc7a
&#xbca1;&#xd130; &#xad6c;&#xc870;&#xc5d0; &#xc720;&#xc5f0;&#xc131;&#xc774; &#xd544;&#xc694;&#xd569;&#xb2c
&#xb798;&#xc2a4;&#xd130; &#xad6c;&#xc870; &#xb610;&#xd55c; &#xc774;&#xb7f0; &#xd6a8;&#xc6a9;&#xc744;
&#bc1b;&#xb294; &#xac83;&#xc774; &#xb300;&#xb2e8;&#xd55c; &#xc7a5;&#xc810;&#xc774;&#xb77c;&#xace0;
&#xc0dd;&#xac01;&#xd569;&#xb2c8;&#xb2e4;.

```

13. `raster2pgsql` &#xc774; &#xb300;&#xc6a9;&#xb7c9; &#xd30c;&#xc77c;&#xc744; &#xb85c;&#xb4dc;&#xd560; &#xb54c;  
*String of N bytes is too long for encoding conversion* &#xc774; &#xb77c;&#xb294; &#xc624;&#xb958;&#xb97c; &#xb0b4; &#xb  
`raster2pgsql` &#xc740; &#xb85c;&#xb4dc;&#xd560; &#xd30c;&#xc77c;&#xc744; &#xc0dd;&#xc131;&#xd560; &#xb54c;  
&#xc0ac;&#xc6a9;&#xc790; &#xb370;&#xc774;&#xd130;&#xbca0;&#xc774;&#xc2a4;&#xc640; &#xc5b4;&#xb5a4;  
&#xc5f0;&#xacb0;&#xb3c4; &#xd558;&#xc9c0; &#xc54a;&#xc2b5;&#xb2c8;&#xb2e4;. &#xc0ac;&#xc6a9;&#xc790;  
&#xb370;&#xc774;&#xd130;&#xbca0;&#xc774;&#xc2a4;&#xac00; &#xc0ac;&#xc6a9;&#xc790; &#xb370;&#xc774;&#xd130  
&#xc778;&#xc54;&#xb529;&#xacfc; &#ba85;&#bc31;&#xd788; &#xb2e4;&#xb978; &#xd074;&#xb77c;&#xc774;&#xc5b  
&#xc778;&#xc54;&#xb529;&#xc744; &#xc124;&#xc815;&#xd55c; &#xacbd;&#xc6b0;. &#xb300;&#xc6a9;&#xb7c9;  
(30MB &#xc774;&#xc0c1;) &#xb798;&#xc2a4;&#xd130; &#xd30c;&#xc77c;&#xc744; &#xb85c;&#xb4dc;&#xd560;  
&#xb54c; bytes is too long for encoding conversion &#xc624;&#xb958;&#xac00; &#bcl;&#xc0dd;&#xd  
&#xc218;&#xb3c4; &#xc788;&#xc2b5;&#xb2c8;&#xb2e4;. &#xc608;&#xb97c; &#b4e4;&#xc5b4; &#xc0ac;&#xc6a9;&#xc7  
&#xb370;&#xc774;&#xd130;&#xbca0;&#xc774;&#xc2a4;&#xc758; &#xc778;&#xc54;&#xb529;&#xc774; UTF8&#xc778;&  
&#xc708;&#xb3c4;&#xc6b0; &#xc751;&#xc6a9; &#xd504;&#xb85c;&#xadf8;&#xb7a8;&#xc744; &#xc9c0;&#xc6d0;&#xd55  
&#xc704;&#xd574; &#xd074;&#xb77c;&#xc774;&#xc5b8;&#xd2b8; &#xc778;&#xc54;&#xb529;&#xc744; WIN1252  
&#xc73c;&#xb85c; &#xc124;&#xc815;&#xd55c; &#xacbd;&#xc6b0;. &#xc77c;&#bc18;&#xc801;&#xc73c;&#xb85c;  
&#xc774; &#xc624;&#xb958;&#xac00; &#bc1c;&#xc0dd;&#xd569;&#xb2c8;&#xb2e4;. &#xc774; &#xc624;&#xb958;&#xb9  
&#xd53c;&#xd558;&#xb824;&#xba74; &#xb85c;&#xb4dc; &#xc791;&#xc5c5;&#xc911; &#xc0ac;&#xc6a9;&#xc790;  
&#xb370;&#xc774;&#xd130;&#xbca0;&#xc774;&#xc2a4;&#xc758; &#xc778;&#xc54;&#xb529;&#xacfc; &#xd074;&#xb77  
&#xc778;&#xc54;&#xb529;&#xc774; &#xb3d9;&#xc77c;&#xd55c;&#xc9c0; &#xd655;&#xc778;&#xd558;&#xc2ed;&#xc2d  
&#xc0ac;&#xc6a9;&#xc790;&#xc758; &#xb85c;&#xb4dc; &#xc2a4;&#xd06c;&#xb9bd;&#xd2b8;&#xc5d0; &#xd655;&#xc2e  
&#xc124;&#xc815;&#xd574;&#xb450;&#ba74; &#xb429;&#xb2c8;&#xb2e4;. &#xb2e4;&#xc74c;&#xc740; &#xc708;&#xb  
&#xc608;&#xc2dc;&#xc785;&#xb2c8;&#xb2e4;:

```
set PGCLIENTENCODING=UTF8
```

```
&#xc720;&#xb2c9;&#xc2a4;/&#xb9ac;&#xb205;&#xc2a4;&#xc758; &#xacbd;&#xc6b0;:
```

```
export PGCLIENTENCODING=UTF8
```

<http://trac.osgeo.org/postgis/ticket/2209>

14. *ST\_FromGDALRaster* `ERROR: RASTER_fr`  
*Could not open bytea with GDAL. Check that the bytea is of a GDAL supported format.*  
*ST\_AsPNG* `ERROR: rt_raster_to_gdal: Could not load the output GDAL driver`

PostGIS 2.1.3, 2.0.5, GDAL, DB, [PostGIS 2.0.6, 2.1.3 security release](#), Section 2.1

# Chapter 14

## PostGIS Extras

This chapter documents features found in the extras folder of the PostGIS source tarballs and source repository. These are not always packaged with PostGIS binary releases, but are usually PL/pgSQL based or standard shell scripts that can be run as is.

### 14.1 &#xc8fc;&#xc18c; &#xd45c;&#xc900;&#xd654; &#xb3c4;&#xad6c;

&#xc774; &#xb3c4;&#xad6c;&#xb294; **PAGC standardizer** &#xc758; &#xd55c; &#xc08;&#xb798;(fork)&#xc785;&#xb2c8;&#xb2e8;&#xc08;&#xb798;&#xc758; &#xc6d0;&#xbcf8; &#xcf54;&#xb4dc;&#xb294; **PAGC PostgreSQL &#xc8fc;&#xc18c; &#xd45c;&#xb3c4;&#xad6c;** &#xc600;&#xc2b5;&#xb2c8;&#xb2e4;).

&#xc8fc;&#xc18c; &#xd45c;&#xc900;&#xd654; &#xb3c4;&#xad6c;&#xb294; &#xb2e8;&#xc77c; &#xb77c;&#xc778; &#xc8fc;&#xd30c;&#xc2f1; &#xb3c4;&#xad6c;&#xb85c; &#xc8fc;&#xc18c;&#xb97c; &#xc785;&#xb825;&#xbcb1b;&#xc544; &#xd14c;&#xc800;&#xc7a5;&#xb41c; &#xc77c;&#xb828;&#xc758; &#xaddc;&#xce59; &#xbcb0f; &#xb3c4;&#xc6b0;&#xbbf8; &#xc5b4;&#xbaa9;&#xb85d;(lexicon; lex) &#xd14c;&#xc774;&#xbe14; &#xacfc; &#xc9c0;&#xba85; &#xc0c9;&#xc778;(gazetteer; gaz) &#xd14c;&#xc774;&#xbe14; &#xc5d0; &#xae30;&#xbcb18;&#xd574;&#xc11c; &#xc815;&#xaddc;&#xd654;&#xd569;&#xb2c8;&#xb2e4;).

```
CREATE EXTENSION address_standardizer; &#xba85;&#xb839;&#xc73c;&#xb85c; &#xc124;&#xce58;&#xd560; &#xc218; &#xc788;&#xb294; address_standardizer &#xb77c;&#xb294; &#xb2e8;&#xc77c; PostgreSQL &#xd655;&#xc7a5;&#xd504;&#xb85c;&#xadf8;&#xb7a8; &#xb77c;&#xc774;&#xbe0c;&#xb7ec;&#xb9ac;&#xb85c; &#xcf54;&#xb4dc;&#xb97c; &#xbe4c;&#xb4dc;&#xd569;&#xb2c8;&#xb2e4;. address_standardizer &#xd655;&#xc7a5; &#xd504;&#xb85c;&#xadf8;&#xb354;&#xd574;. address_standardizer_data_us &#xb77c;&#xb294; &#xc0d8;&#xd50c; &#xb370;&#xc774;&#xd130; &#xd655;&#xc7a5; &#xd504;&#xb85c;&#xadf8;&#xb7a8;&#xb3c4; &#xbe4c;&#xb4dc;&#xd558;&#xb294;&#xb370;. &#xbbf8;&#xb370;&#xc774;&#xd130;&#xc5d0; &#xb300;&#xd55c; &#xc9c0;&#xba85; &#xc0c9;&#xc778;. &#xc5b4;&#xd718; &#xbaa9;&#xaddc;&#xce59; &#xd14c;&#xc774;&#xbe14;&#xc744; &#xb2f4;&#xace0; &#xc788;&#xc2b5;&#xb2c8;&#xb2e4;. CREATE EXTENSION address_standardizer_data_us; &#xba85;&#xb839;&#xc73c;&#xb85c; &#xc774; &#xd655;&#xc7a5; &#xd504;&#xb85c;&#xadf8;&#xb7a8;&#xc744; &#xc124;&#xce58;&#xd560; &#xc218; &#xc788;&#xc2b5;&#xb2c8;&#xb2e4;.
```

```
PostGIS extensions/address_standardizer &#xc5d0;&#xc11c; &#xc774; &#xd655;&#xc7a5; &#xd504;&#xb85c;&#xadf8;&#xc704;&#xd55c; &#xcf54;&#xb4dc;&#xb97c; &#xcc3e;&#xc744; &#xc218; &#xc788;&#xc73c;&#xba70;. &#xd604;&#xc7ac; &#xb3c5;&#xb9bd;&#xc801;&#xc73c;&#xb85c; &#xb3fc; &#xc788;&#xc2b5;&#xb2c8;&#xb2e4;.
```

```
&#xc124;&#xce58;&#xc5d0; &#xb300;&#xd55c; &#xc9c0;&#xce68;&#xc744; &#xbcf4;&#xace0; &#xc2f6;&#xb2e4;&#xba74; Section 2.3 &#xb97c; &#xcc38;&#xc870;&#xd558;&#xc2ed;&#xc2dc;&#xc624;.
```

#### 14.1.1 &#xd30c;&#xc2f1; &#xb3c4;&#xad6c;&#xc758; &#xc791;&#xb3d9; &#xbcb9;&#xc2dd;

```
&#xd30c;&#xc2f1; &#xb3c4;&#xad6c;&#xb294; &#xc624;&#xb978;&#xcabd;&#xc5d0;&#xc11c; &#xc67c;&#xcabd; &#xbcb9;&#xba3c;&#xc800; &#xc6b0;&#xd3b8;&#xbcb88;&#xd638;. &#xc8fc;/&#xb3c4;. &#xc2dc;&#xad70;&#xad6c;&#xc5d0; &#xd574;&#xc70;&#xc2dc;&#xc801;(macro) &#xc694;&#xc18c;&#xb97c; &#xd0d0;&#xc0c9;&#xd55c; &#xb2e4;&#xc74c;. &#xac00;&#xbcb88;&#xc9c0;&#xc218; &#xb610;&#xb294; &#xad50;&#xcc28;&#xc810; &#xb610;&#xb294; &#xb79c;&#xb4dc;&#xb9c8;&#xc00;&#xc6b4;&#xb370; &#xc5b4;&#xb5a4; &#xac83;&#xc744; &#xcc98;&#xb9ac;&#xd558;&#xace0; &#xc788;&#xb294;&#xc774;.
```

`&#xc815;&#xd558;&#xae30; &#xc704;&#xd55c; &#xbbf8;&#xc2dc;&#xc801;(micro) &#xc694;&#xc18c;&#xb97c; &#xd0d0;&#xc0c9;&#xd569;&#xb2c8;&#xb2e4;. &#xd604;&#xc7ac; &#xad6d;&#xac00; &#xcf54;&#xb4dc; &#xb610;&#xb294; &#xad6d;&#xac00;&#xba85;&#xc744; &#xd0d0;&#xc0c9;&#xd558;&#xc9c0;&#xb294; &#xc54a;&#xc9c0;&#xb9cc;,&#xd5a5;&#&#xcd94;&#xac00;&#xb420; &#xc218;&#xb3c4; &#xc788;&#xc2b5;&#xb2c8;&#xb2e4;.`

`&#xad6d;&#xac00; &#xcf54;&#xb4dc; &#xbbf8;&#xad6d; &#xb610;&#xb294; &#xce90;&#xb098;&#xb2e4;&#xc758; &#xc8fc;/&#xbbf8;&#xad6d; &#xb610;&#xb294; &#xce90;&#xb098;&#xb2e4;&#xc758; &#xc6b0;&#xd3b8;&#xbc88;&#xd638;&#xb9 &#xbc14;&#xd0d5;&#xc73c;&#xb85c; &#xbbf8;&#xad6d; &#xb610;&#xb294; &#xce90;&#xb098;&#xb2e4;&#xb85c; &#xac00;&#xc815;&#xd569;&#xb2c8;&#xb2e4;.`

`&#xc6b0;&#xd3b8;&#xbc88;&#xd638;/&#xc9d1;&#xcf54;&#xb4dc;(zip code) &#xd384;(Perl) &#xd638;&#xd658; &#xc815;&#&#xd45c;&#xd604;&#xc2dd;&#xc744; &#xc774;&#xc6a9;&#xd574;&#xc11c; &#xc6b0;&#xd3b8;&#xbc88;&#xd638;&#xb9 &#xc778;&#xc2dd;&#xd569;&#xb2c8;&#xb2e4;. &#xc774; &#xc815;&#xaddd; &#xd45c;&#xd604;&#xc2dd;&#xc740; &#xd604;&#xc7ac; parseaddress-api.c &#xd30c;&#xc77c;&#xc5d0; &#xb2f4;&#xaca8; &#xc788;&#xace0;,&#xd544;&#xc69 &#xacbd;&#xc6b0; &#xc0c1;&#xb300;&#xc801;&#xc73c;&#xb85c; &#xc27d;&#xac8c; &#xbcc0;&#xacbd;&#xd560; &#xc218; &#xc788;&#xc2b5;&#xb2c8;&#xb2e4;.`

`&#xc8fc;/&#xb3c4; &#xd384;(Perl) &#xd638;&#xd658; &#xc815;&#xaddd; &#xd45c;&#xd604;&#xc2dd;&#xc744; &#xc774;&#xc &#xc6b0;&#xd3b8;&#xbc88;&#xd638;&#xb97c; &#xc778;&#xc2dd;&#xd569;&#xb2c8;&#xb2e4;. &#xc774; &#xc815;&#xad &#xd45c;&#xd604;&#xc2dd;&#xc740; &#xd604;&#xc7ac; parseaddress-api.c &#xd30c;&#xc77c;&#xc5d0; &#xb2f4;&#xaca8 &#xc788;&#xb294;&#xb370;,&#xd5a5;&#xd6c4; &#xc720;&#xc9c0;&#xbcf4;&#xc218;&#xb97c; &#xb354; &#xc27d;&#xad &#xd558;&#xae30; &#xc704;&#xd574; "includes" &#xb85c; &#xc774;&#xb3d9;&#xd560; &#xc218;&#xb3c4; &#xc788;&#xc`

## 14.1.2 `&#xc8fc;&#xc18c; &#xd45c;&#xc900;&#xd654; &#xb3c4;&#xad6c; &#xc720;&#xd615;`

### 14.1.2.1 `stdaddr`

`stdaddr` — `&#xc8fc;&#xc18c;&#xc758; &#xc694;&#xc18c;&#xb4e4;&#xb85c; &#xc774;&#xb8e8;&#xc5b4;&#xc9c4; &#xd569;&#&#xc720;&#xd615;&#xc785;&#xb2c8;&#xb2e4;. standardize_address &#xd568;&#xc218;&#xac00; &#xc774; &#xc720;&#&#xb8; &#xd658;&#xd569;&#xb2c8;&#xb2e4;.`

### `&#xc124;&#xba85;`

`&#xc8fc;&#xc18c;&#xc758; &#xc694;&#xc18c;&#xb4e4;&#xb85c; &#xc774;&#xb8e8;&#xc5b4;&#xc9c4; &#xd569;&#xc131; &#xc720;&#xd615;&#xc785;&#xb2c8;&#xb2e4;. standardize_address &#xd568;&#xc218;&#xac00; &#xc774; &#xc720;&#xd615;&#&#xb8; &#xd658;&#xd569;&#xb2c8;&#xb2e4;. PAGC Postal Attributes &#xc5d0;&#xc11c; &#xc694;&#xc18c;&#xb4e4;&#xc5d0 &#xb300;&#xd55c; &#xba87;&#xba87; &#xc124;&#xba85;&#xc744; &#xbe4c;&#xb824;&#xc654;&#xc2b5;&#xb2c8;&#xb2e4;.`

`&#xaddd;&#xce59; &#xd14c;&#xc774;&#xbe14; &#xc5d0;&#xc11c; &#xcd9c;&#xb825; &#xcc38;&#xc870; &#xbc88;&#xd638;&#&#xd45c;&#xc2dc;&#xd558;&#xb294; &#xd1a0;&#xd070; &#xbc88;&#xd638;&#xb97c; &#xcc3e;&#xc544;&#xbcf4; &#xc218; &#xc788;&#xc2b5;&#xb2c8;&#xb2e4;.`



This method needs `address_standardizer` extension.

`building` `&#xbb38;&#xc790;&#xd615;(&#xd1a0;&#xd070; &#xbc88;&#xd638; 0)&#xc785;&#xb2c8;&#xb2e4;: &#xac74;&#xbb3c &#xbc88;&#xd638; &#xb610;&#xb294; &#xac74;&#xbb3c;&#xba85;&#xc744; &#xcc38;&#xc870;&#xd569;&#xb2c8;&#xb2 &#xd30c;&#xc2f1;&#xb418;&#xc9c0; &#xc54a;&#xc740; &#xac74;&#xbb3c; &#xc2dd;&#xbcc4;&#xc790; &#bc0f; &#xc720;&#xd615;&#xc785;&#xb2c8;&#xb2e4;. &#xc8fc;&#xc18c; &#xb300;&#xbd80;&#xbd84;&#xc758; &#xacbd;&#xc6 &#xc77c;&#xbc18;&#xc801;&#xc73c;&#xb85c; &#xbe44;&#xc5b4; &#xc788;&#xc2b5;&#xb2c8;&#xb2e4;.`

`house_num` `&#xbb38;&#xc790;&#xd615;(&#xd1a0;&#xd070; &#xbc88;&#xd638; 1)&#xc785;&#xb2c8;&#xb2e4;: &#xb3c4;&#xb &#xbc88;&#xc9c0;&#xc218;&#xc785;&#xb2c8;&#xb2e4;. &#xc608;: 75 State Street &#xc758; 75 &#bc88;&#xc9c0;`

`predir` `&#xbb38;&#xc790;&#xd615;(&#xd1a0;&#xd070; &#xbc88;&#xd638; 2)&#xc785;&#xb2c8;&#xb2e4;: North, South, East, West &#xb4f1;&#xacfc; &#xac19;&#xc774; &#xbc29;&#xd5a5;&#xc744; &#xb098;&#xd0c0;&#xb0b4;&#xb294; &#xb3c4;&#xb85c;&#xba85; &#xc811;&#xb450;&#xc0ac;(STREET NAME PRE-DIRECTIONAL)&#xc785;&#xb2c8;&#xb2`

**qual** `&#x33;(&#xd1a0;&#xd070; &#xbc88;&#xd638; 3)&#xc785;&#xb2c8;&#xb2e4;: &#xb3c4;&#xb85c;&#xc804;&#xce58;&#xc218;&#xc2dd;&#xc5b4;(STREET NAME PRE-MODIFIER)&#xc785;&#xb2c8;&#xb2e4;: &#xc608;: 3715 OLD HIGHWAY 99 &#xc5d0;&#xc11c; OLD`

**pretype** `&#x33;(&#xd1a0;&#xd070; &#xbc88;&#xd638; 4)&#xc785;&#xb2c8;&#xb2e4;: &#xb3c4;&#xb85c;&#xc811;&#xb450;&#xc0ac; &#xc720;&#xd615;(STREET PREFIX TYPE)&#xc785;&#xb2c8;&#xb2e4;:`

**name** `&#x33;(&#xd1a0;&#xd070; &#xbc88;&#xd638; 5)&#xc785;&#xb2c8;&#xb2e4;: &#xb3c4;&#xb85c;&#xc785;&#xb2c8;&#xb2e4;: NAME)&#xc785;&#xb2c8;&#xb2e4;:`

**suftype** `&#x33;(&#xd1a0;&#xd070; &#xbc88;&#xd638; 6)&#xc785;&#xb2c8;&#xb2e4;: St, Ave, Cir&#xc640;&#xc19;&#xc740; &#xb3c4;&#xb85c; &#xc811;&#xbbbf8;&#xc0ac; &#xc720;&#xd615;(STREET POST TYPE)&#xc785;&#xb3c4;&#xb85c;&#xba85;&#xc758; &#xbab8;&#xd1b5; &#xb4a4;&#xc5d0; &#xbd99;&#xb294; &#xb3c4;&#xb85c; &#xc720;&#xd615;&#xc744; &#xb73b;&#xd569;&#xb2c8;&#xb2e4;: &#xc608;: 75 State Street &#xc5d0;&#xc11c; STREET`

**sufdir** `&#x33;(&#xd1a0;&#xd070; &#xbc88;&#xd638; 7)&#xc785;&#xb2c8;&#xb2e4;: &#xb29;&#xd5a5;&#xb098;&#xd0c0;&#xb0b4;&#xb294; &#xb3c4;&#xb85c; &#xc811;&#xbbbf8;&#xc0ac;(STREET POST-DIRECTIONAL)&#xb3c4;&#xb85c;&#xba85; &#xb4a4;&#xc5d0; &#xbd99;&#xc5b4; &#xb29;&#xd5a5;&#xc744; &#xb098;&#xd0c0;&#xb0b4;&#xc218;&#xc2dd;&#xc5b4;&#xc785;&#xb2c8;&#xb2e4;: &#xc608;: 3715 TENTH AVENUE WEST &#xc5d0;&#xc11c; WEST`

**ruralroute** is text (token number 8): RURAL ROUTE . Example 7 in RR 7.

**extra** `&#x33;&#xc785;&#xb2c8;&#xb2e4;: &#xce35;&#xc218; &#xc19;&#xc740; &#xcd94;&#xc00; &#xc815;&#xbcf4;&#xc785;&#xb2c8;&#xb2e4;:`

**city** `&#x33;(&#xd1a0;&#xd070; &#xbc88;&#xd638; 10)&#xc785;&#xb2c8;&#xb2e4;: &#xc608;: &#xbcf4;&#xc2dc;`

**state** `&#x33;(&#xd1a0;&#xd070; &#xbc88;&#xd638; 11)&#xc785;&#xb2c8;&#xb2e4;: &#xc608;: &#xb9e4;&#xc8fc;`

**country** `&#x33;(&#xd1a0;&#xd070; &#xbc88;&#xd638; 12)&#xc785;&#xb2c8;&#xb2e4;: &#xc608;: USA`

**postcode** `&#x33;(&#xd1a0;&#xd070; &#xbc88;&#xd638; 13)&#xc6b0;&#xd3b8;&#xbc88;&#xd638;(postal code, zip code)&#xc785;&#xb2c8;&#xb2e4;: &#xc608;: 02109`

**box** `&#x33;(&#xd1a0;&#xd070; &#xbc88;&#xd638; 14, 15) &#xc0ac;&#xc11c;&#xd568; &#xbc88;&#xd638; BOX NUMBER)&#xc785;&#xb2c8;&#xb2e4;: &#xc608;: 02109`

**unit** `&#x33;(&#xd1a0;&#xd070; &#xbc88;&#xd638; 17) &#xc544;&#xd30c;&#xd2b8; &#xb3d9;&#xc218; &#xb610;&#xb294; &#xd638;&#xc218;&#xc785;&#xb2c8;&#xb2e4;: &#xc608;: APT 3B &#xc5d0;&#xc11c; 3B`

### 14.1.3 `&#xc8fc;&#xc18c;&#xd45c;&#xc900;&#xd654; &#xb3c4;&#xad6c; &#xd14c;&#xc774;&#xbe14;`

#### 14.1.3.1 `&#xaddc;&#xce59; &#xd14c;&#xc774;&#xbe14;`

`&#xaddc;&#xce59; &#xd14c;&#xc774;&#xbe14; — &#xaddc;&#xce59; &#xd14c;&#xc774;&#xbe14;&#xc740; &#xc8fc;&#xc18c; &#xc785;&#xb825; &#xb30;&#xc5f4; &#xd1a0;&#xd070;&#xc744; &#xd45c;&#xc900;&#xd654;&#xb41c; &#xcd9c;&#xb825; &#xb30;&#xc5f4; &#xd1a0;&#xd070;&#xc5d0; &#xb9e4;&#xd551;&#xd558;&#xb294; &#xc77c;&#xb828;&#xc758; &#xaddc;&#xb2f4;&#xcace0; &#xc788;&#xc2b5;&#xb2c8;&#xb2e4;: &#xc785;&#xb825; &#xd1a0;&#xd070;&#xb4e4;&#xc758; &#xc9d1;&#xadf8; &#xb4a4;&#xc5d0; -1(&#xc885;&#xcacb0;&#xc790;: terminator), &#xadf8; &#xb4a4;&#xc5d0; &#xcd9c;&#xb825; &#xd1a0;&#xd070;&#xb4e4;&#xc758; &#xc9d1;&#xd569;:, &#xadf8; &#xb4a4;&#xc5d0; -1, &#xadf8; &#xb4a4;&#xc5d0; &#xaddc;&#xce59;&#xc758; &#xc885;&#xb958;&#xb97c; &#xd45c;&#xc2dc;&#xd558;&#xb294; &#xbc88;&#xd638;:, &#xadf8; &#xb4a4;&#xc5d0; &#xaddc;&#xce59;&#xc758; &#xc21c;&#xc704;&#xb97c; &#xb098;&#xc5f4;&#xd574;&#xc11c; &#xd558;&#xaddc;&#xce59;&#xc744; &#xc815;&#xc758;&#xd569;&#xb2c8;&#xb2e4;:`

**PGC**

PGC: PGC Address Standardizer Rule records

**id** PGC Address Standardizer Rule records

**rule** PGC Address Standardizer Rule records

**PGC Input Tokens**

PGC Input Tokens

**PGC Input Tokens**

**AMPERS** (13). PGC Input Tokens

**DASH** (9). PGC Input Tokens

**DOUBLE** (21). PGC Input Tokens

**FRACT** (25). PGC Input Tokens

**MIXED** (23). PGC Input Tokens

**NUMBER** (0). PGC Input Tokens

**ORD** (15). PGC Input Tokens

**ORD** (18). *Order*

**WORD** (1). *Word*

**BOXH** (14). *Box*

**BOXH** (14). *Box*

**BUILDH** (19). *Build*

**BUILD** (19). *Build*

**DIRECT** (22). *Direct*

**MILE** (20). *Mile*

**ROAD** (6). *Road*

**RR** (8). *RR*

**TYPE** (2). *Type*

**UNITH** (16). *Unit*

**QUINT** (28). *Quint*

**QUAD** (29). *Quad*

**PCH** (27). *PCH*

**PCT** (26). *PCT*

**&#xbd88;&#xc6a9;&#xc5b4; (&#xf967;&#x7528;&#xa9e;; stopword)**

STOPWORD&#xb294; WORD&#xc640; &#xacb0;&#xd569;&#xd569;&#xb2c8;&#xb2e4;. &#xaddc;&#xce59; &#xc0c1; &#xbcf5;&#xc0f; STOPWORD &#xc2a4;&#xd2b8;&#xb9c1;&#xc744; &#xb2e8;&#xc77c; WORD &#xd1a0;&#xd070;&#xc73c;&#xd45c;&#xd604;&#xd560; &#xac83;&#xc785;&#xb2c8;&#xb2e4;.

**STOPWORD** (7). &#xc5b4;&#xd718; &#xc0c1; &#xc911;&#xc694;&#xb3c4;&#xac00; &#xb0ae;&#xc544; &#xd30c;&#xc2f1; &#xc791;&#xc5c5;&#xc2dc; &#xc0dd;&#xb7b5;&#xd560; &#xc218; &#xc788;&#xb294; &#xb2e8;&#xc5b4;&#xc785;&#xb2e4;&#xc608;; *THE*

**&#xcd9c;&#xb825; &#xd1a0;&#xd070;**

&#xccab; &#xbc88;&#xc9f8; -1(&#xc885;&#xacb0;&#xc790;) &#xb2e4;&#xc74c;&#xc5d0; &#xcd9c;&#xb825; &#xd1a0;&#xd070;&#xbcf5;&#xc0f; &#xadf8; &#xc21c;&#xc11c;&#xac00; &#xc624;&#xace0;; &#xadf8; &#xb2e4;&#xc74c;&#xc5d0; &#xc885;&#xacb0;&#xb2e4;-1 &#xc774; &#xb530;&#xb77c;&#xc635;&#xb2c8;&#xb2e4;. **stdaddr** &#xc5d0;&#xc11c; &#xb9ce;&#xc740; &#xd574;&#xb2f9; &#xcd9c;&#xb825; &#xd1a0;&#xd070;&#xb4e4;&#xc758; &#xbaa9;&#xb85d;&#xc744; &#xc18c;&#xac1c;&#xd558;&#xace0; &#xc788;&#xc2b5;&#xb2c8;&#xb2e4;. &#xaddc;&#xce59;&#xc758; &#xc720;&#xd615;&#xc5d0; &#xb530;&#xb77c; &#xc0ac;&#xc218; &#xc788;&#xb294; &#xcd9c;&#xb825; &#xd1a0;&#xd070;&#xb4e4;&#xb3c4; &#xb2ec;&#xb77c;&#xc9d1;&#xb2c8;&#xb2e4;. the section called “&#xaddc;&#xce59; &#xc720;&#xd615; &#xbcf5;&#xc21c;&#xc704;.” &#xc5d0;&#xc11c; &#xac01; &#xaddc;&#xc720;&#xd615;&#xc5d0; &#xb300;&#xd574; &#xc720;&#xd6a8;&#xd55c; &#xcd9c;&#xb825; &#xd1a0;&#xd070; &#xbaa9;&#xbcf5;&#xc218; &#xc788;&#xc2b5;&#xb2c8;&#xb2e4;.

**&#xaddc;&#xce59; &#xc720;&#xd615; &#xbcf5; &#xc21c;&#xc704;**

&#xaddc;&#xce59;&#xc744; &#xad6c;&#xc131;&#xd558;&#xb294; &#xb9c8;&#xc9c0;&#xb9c9; &#bd80;&#bd84;&#xc740; &#xb2e4;&#xc74c; &#xac00;&#xc6b4;&#xb370; &#xd558;&#xb098;&#xb85c; &#xd45c;&#xc2dc;&#xb418;&#xb294; &#xaddc;&#xc720;&#xd615;&#xacfc; &#xadf8; &#xb2e4;&#xc74c;&#xc5d0; &#xc624;&#xb294; &#xaddc;&#xce59; &#xc21c;&#xc704;&#xb2e4;&#xaddc;&#xce59;&#xc758; &#xc21c;&#xc704;&#xb294; (&#xcd5c;&#xc800;) 0&#bd80;&#xd130; (&#xcd5c;&#xace0;) 17&#xae4c;&#xc9c0;&#xc785;&#xb2c8;&#xb2e4;.

**MACRO\_C**

(&#xd1a0;&#xd070; &#xbc88;&#xd638; = "0"). *PLACE STATE ZIP* &#xac19;&#xc740; **MACRO** &#xc808;&#xc744; &#xd30c;&#xc704;&#xd55c; &#xaddc;&#xce59; &#xd074;&#xb798;&#xc2a4;&#xc785;&#xb2c8;&#xb2e4;.

**MACRO\_C &#xcd9c;&#xb825; &#xd1a0;&#xd070;** (<http://www.pagcgeo.org/docs/html/pagc-12.html#--r-tyt--> &#xc5d0;&#xc11c;&#xbcf5;&#xc1c;&#xcdcc;)

**CITY** (&#xd1a0;&#xd070; &#xbc88;&#xd638; "10"). &#xc608;; "Albany"

**STATE** (&#xd1a0;&#xd070; &#xbc88;&#xd638; "11"). &#xc608;; "NY"

**NATION** (&#xd1a0;&#xd070; &#xbc88;&#xd638; "12"). &#xc774; &#xc18d;&#xc131;&#xc740; &#xb300;&#bd80;&#bd84;&#xbcf5;&#xc38;&#xc870; &#xd30c;&#xc77c;&#xc5d0;&#xc11c; &#xc4f0;&#xc774;&#xc9c0; &#xc54a;&#xc2b5;&#xb2c8;&#xb2e4;&#xc608;; "USA"

**POSTAL** (&#xd1a0;&#xd070; &#xbc88;&#xd638; "13"). (*SADS* &#xc694;&#xc18c; "ZIP CODE", "PLUS 4"). &#xbbf8;&#xad6d; &#xc9d1;&#xcf54;&#xb4dc; &#xbcf5; &#xc90;&#xb098;&#xb2e4; &#xc6b0;&#xd3b8; &#xbc88;&#xd638; &#xbaa8;&#xb44c;&#xc774; &#xc18d;&#xc131;&#xc744; &#xc774;&#xc6a9;&#xd569;&#xb2c8;&#xb2e4;.

**MICRO\_C**

(&#xd1a0;&#xd070; &#xbc88;&#xd638; = "1"). (&#xac00;&#xad6c; &#xb3c4;&#xb85c;; *sudir*, *predir*, *pretyp*, *suftype*, *qualif* &#xc640; &#xac19;&#xc740;) &#xc644;&#xc804;&#xd55c; **MICRO** &#xc808;&#xc744; &#xd30c;&#xc2f1;&#xd558;&#xae3c;&#xc704;&#xd55c; &#xaddc;&#xce59; &#xd074;&#xb798;&#xc2a4;&#xc785;&#xb2c8;&#xb2e4; (&#xc608;; *ARC\_C* &#xbcf5; *CIVIC\_C*). &#xbe4c;&#xb4dc; &#xc2dc;&#xae30;&#xc5d0;&#xb294; &#xc774; &#xaddc;&#xce59;&#xb4e4;&#xc744; &#xc774;&#xc54a;&#xc2b5;&#xb2c8;&#xb2e4;.

**MICRO\_C &#xcd9c;&#xb825; &#xd1a0;&#xd070;** (<http://www.pagcgeo.org/docs/html/pagc-12.html#--r-tyt--> &#xc5d0;&#xc11c;&#xbcf5;&#xc1c;&#xcdcc;).

**HOUSE** (&#xd1a0;&#xd070; &#xbc88;&#xd638; 1)&#xc785;&#xb2c8;&#xb2e4;: &#xb3c4;&#xb85c;&#xbc88;&#xc9c0;&#xc218;&#xc785;&#xb2c8;&#xb2e4;. &#xc608;: 75 State Street &#xc758; 75 &#xbc88;&#xc9c0;

**predir** &#xbb38;&#xc790;&#xd615;(&#xd1a0;&#xd070; &#xbc88;&#xd638; 2)&#xc785;&#xb2c8;&#xb2e4;: North, South, East, West &#xb4f1;&#xacfc; &#xac19;&#xc774; &#xbc29;&#xd5a5;&#xc744; &#xb098;&#xd0c0;&#xb0b4;&#xb294; &#xb3c4;&#xb85c;&#xba85; &#xc811;&#xb450;&#xc0ac;(STREET NAME PRE-DIRECTIONAL)&#xc785;&#xb2c8;&#xb2e4;

**qual** &#xbb38;&#xc790;&#xd615;(&#xd1a0;&#xd070; &#xbc88;&#xd638; 3)&#xc785;&#xb2c8;&#xb2e4;: &#xb3c4;&#xb85c;&#xc804;&#xce58;&#xc218;&#xc2dd;&#xc5b4;(STREET NAME PRE-MODIFIER)&#xc785;&#xb2c8;&#xb2e4;. &#xc608;: 3715 OLD HIGHWAY 99 &#xc5d0;&#xc11c; *OLD*

**pretype** &#xbb38;&#xc790;&#xd615;(&#xd1a0;&#xd070; &#xbc88;&#xd638; 4)&#xc785;&#xb2c8;&#xb2e4;: &#xb3c4;&#xb85c;&#xc811;&#xb450;&#xc0ac; &#xc720;&#xd615;(STREET PREFIX TYPE)&#xc785;&#xb2c8;&#xb2e4;.

**street** &#xbb38;&#xc790;&#xd615;(&#xd1a0;&#xd070; &#xbc88;&#xd638; 5)&#xc785;&#xb2c8;&#xb2e4;: &#xb3c4;&#xb85c;&#xc0ac; &#xc785;&#xb2c8;&#xb2e4;: &#xc785;&#xb2c8;&#xb2e4;.

**suftype** &#xbb38;&#xc790;&#xd615;(&#xd1a0;&#xd070; &#xbc88;&#xd638; 6)&#xc785;&#xb2c8;&#xb2e4;: St, Ave, Cir&#xc640;&#xac19;&#xc740; &#xb3c4;&#xb85c; &#xc811;&#xbbbf8;&#xc0ac; &#xc720;&#xd615;(STREET POST TYPE)&#xc785;&#xb2c8;&#xb2e4; &#xb3c4;&#xb85c;&#xba85;&#xc758; &#xbab8;&#xd1b5; &#xb4a4;&#xc5d0; &#xbd99;&#xb294; &#xb3c4;&#xb85c; &#xc720;&#xd615;&#xc744; &#xb73b;&#xd569;&#xb2c8;&#xb2e4;. &#xc608;: 75 State Street &#xc5d0;&#xc11c; *STREET*

**sufdir** &#xbb38;&#xc790;&#xd615;(&#xd1a0;&#xd070; &#xbc88;&#xd638; 7)&#xc785;&#xb2c8;&#xb2e4;: &#xbc29;&#xd5a5;&#xb098;&#xd0c0;&#xb0b4;&#xb294; &#xb3c4;&#xb85c; &#xc811;&#xbbbf8;&#xc0ac;(STREET POST-DIRECTIONAL)&#xc785;&#xb2c8;&#xb2e4; &#xb3c4;&#xb85c;&#xba85; &#xb4a4;&#xc5d0; &#xbd99;&#xc5b4; &#xbc29;&#xd5a5;&#xc744; &#xb098;&#xd0c0;&#xb0b4;&#xb294; &#xc218;&#xc2dd;&#xc5b4;&#xc785;&#xb2c8;&#xb2e4;. &#xc608;: 3715 TENTH AVENUE WEST &#xc5d0;&#xc11c; *WEST*

## ARC\_C

(&#xd1a0;&#xd070; &#xbc88;&#xd638; = "2"). HOUSE &#xc18d;&#xc131;&#xc744; &#xc81c;&#xc678;&#xd558;&#xace0; MICRO &#xc808;&#xc744; &#xd30c;&#xc2f1;&#xd558;&#xae30; &#xc704;&#xd55c; &#xaddc;&#xce59; &#xd074;&#xb798;&#xc2a4; &#xb530;&#xb77c;&#xc11c; HOUSE &#xd1a0;&#xd070;&#xc744; &#xbe80; MICRO\_C &#xcd9c;&#xb825; &#xd1a0;&#xd070; &#xc9d1;&#xd569;&#xc744; &#xc774;&#xc6a9;&#xd569;&#xb2c8;&#xb2e4;.

## CIVIC\_C

(&#xd1a0;&#xd070; &#xbc88;&#xd638; = "3"). HOUSE &#xc18d;&#xc131;&#xc744; &#xd30c;&#xc2f1;&#xd558;&#xae30; &#xc704;&#xd55c; &#xaddc;&#xce59; &#xd074;&#xb798;&#xc2a4;&#xc785;&#xb2c8;&#xb2e4;.

## EXTRA\_C

(&#xd1a0;&#xd070; &#xbc88;&#xd638; = "4"). EXTRA &#xc18d;&#xc131; - &#xc9c0;&#xc624;&#xcdf54;&#xb529;&#xc5d0;&#xc81c;&#xc678;&#xb41c; &#xc18d;&#xc131; - &#xc744; &#xd30c;&#xc2f1;&#xd558;&#xae30; &#xc704;&#xd55c; &#xaddc;&#xc2a4;&#xc785;&#xb2c8;&#xb2e4;. &#xbe4c;&#xb4dc; &#xc2dc;&#xae30;&#xc5d0;&#xb294; &#xc774; &#xaddc;&#xce59;&#xb4e4;&#xc744; &#xc774;&#xc6a9;&#xd558;&#xc9c0; &#xc54a;&#xc2b5;&#xb2c8;&#xb2e4;.

**EXTRA\_C &#xcd9c;&#xb825; &#xd1a0;&#xd070;** (<http://www.pagegeo.org/docs/html/page-12.html#--r-ty--> &#xc5d0;&#xc11c; &#xbc1c;&#xcdcc;).

**BLDNG** (&#xd1a0;&#xd070; &#xbc88;&#xd638; 0): &#xd30c;&#xc2f1;&#xb418;&#xc9c0; &#xc54a;&#xc740; &#xac74;&#xbb38;&#xc2dd;&#xbcc4;&#xc790; &#xbc0f; &#xc720;&#xd615;&#xc785;&#xb2c8;&#xb2e4;.

**BOXH** (token number 14): The **BOX** in BOX 3B

**BOXT** (&#xd1a0;&#xd070; &#xbc88;&#xd638; 15): BOX 3B &#xc5d0;&#xc11c; **3B**

**RR** (&#xd1a0;&#xd070; &#xbc88;&#xd638; 8): RR 7 &#xc5d0;&#xc11c; **RR**

**UNITH** (&#xd1a0;&#xd070; &#xbc88;&#xd638; 16): APT 3B &#xc5d0;&#xc11c; **APT**

**UNITT** (&#xd1a0;&#xd070; &#xbc88;&#xd638; 17): APT 3B &#xc5d0;&#xc11c; **3B**

**UNKNWN** (&#xd1a0;&#xd070; &#xbc88;&#xd638; 9): &#xb530;&#xb85c; &#xbd84;&#xb958;&#xb418;&#xc9c0; &#xc54a;&#xc0ac; &#xcd9c;&#xb825;&#xbb3c;&#xc785;&#xb2c8;&#xb2e4;.

**14.1.3.2 &#xc5b4;&#xd718; &#xbaa9;&#xb85d; &#xd14c;&#xc774;&#xbe14;**

&#xc5b4;&#xd718; &#xbaa9;&#xb85d; &#xd14c;&#xc774;&#xbe14; — &#xc5b4;&#xd718; &#xbaa9;&#xb85d;(lex) &#xd14c;&#xc601;&#xc22b;&#xc790; &#xc785;&#xb825;&#xc744; &#xbd84;&#xb958;&#xd558;&#xace0;, &#xd574;&#xb2f9; &#xc785;&#xb825;&#xc744; (1) &#xc785;&#xb825; &#xd1a0;&#xd070;(the section called “&#xc785;&#xb825; &#xd1a0;&#xd070;” &#xc38;&#xc870;) &#xcbc0f; (2) &#xd45c;&#xc900;&#xd654;&#xb41c; &#xd45c;&#xd604;&#xc2dd;&#xacfc; &#xc5f0;&#xad00;&#xc9d3;&#xb294; &#xb370; &#xc4f0;&#xc785;&#xb2c8;&#xb2e4;.

**&#xc124;&#xba85;**

&#xc5b4;&#xd718; &#xbaa9;&#xb85d;(lexicon) &#xd14c;&#xc774;&#xbe14;&#xc740; &#xc601;&#xc22b;&#xc790; &#xc785;&#xb825;&#xbd84;&#xb958;&#xd558;&#xace0;, &#xd574;&#xb2f9; &#xc785;&#xb825;&#xc744; (1) &#xc785;&#xb825; &#xd1a0;&#xd070; section called “&#xc785;&#xb825; &#xd1a0;&#xd070;” &#xc38;&#xc870;) &#xcbc0f; (2) &#xd45c;&#xc900;&#xd654;&#xb41c; &#xd45c;&#xd604;&#xc2dd;&#xacfc; &#xc5f0;&#xad00;&#xc9d3;&#xb294; &#xb370; &#xc4f0;&#xc785;&#xb2c8;&#xb2e4;. &#xc774; &#xd14c;&#xc774;&#xbe14;&#xc5d0;&#xc11c;, &#xc608;&#xb97c; &#xb4e4;&#xc5b4;, &#xc601;&#xb2e8;&#xc5b4; ONE &#xc774; stdword 1 &#xb85c; &#xb9e4;&#xd551;&#xb41c; &#xac83;&#xc744; &#xc3e;&#xc544;&#xcbcfc; &#xc218; &#xc788;&#xc2b5;&#xb2c8;&#xb2e4;.

&#xc5b4;&#xd718; &#xbaa9;&#xb85d; &#xd14c;&#xc774;&#xbe14;&#xc740; &#xcd5c;&#xc18c;&#xd55c; &#xb2e4;&#xc74c; &#xc5f4;&#xc744; &#xac00;&#xc9c0;&#xace0; &#xc788;&#xc5b4;&#xc57c; &#xd569;&#xb2c8;&#xb2e4;. &#xc0ac;&#xc6a9;&#xc9c0;&#xc815; &#xc5f4;&#xc744; &#xcd94;&#xac00;&#xd560; &#xc218;&#xb3c4; &#xc788;&#xc2b5;&#xb2c8;&#xb2e4;.

**id** &#xd14c;&#xc774;&#xbe14;&#xc758; &#xae30;&#xcfc8; &#xd0a4;

**seq** &#xc815;&#xc218;&#xd615;: &#xc815;&#xc758; &#xc88;&#xd638;?

**word** &#xbb38;&#xc790;&#xd615;: &#xc785;&#xb825; &#xb2e8;&#xc5b4;

**stdword** &#xbb38;&#xc790;&#xd615;: &#xd45c;&#xc900;&#xd654;&#xb41c; &#xb300;&#xc38;&#xb2e8;&#xc5b4;

**token** &#xc815;&#xc218;&#xd615;: &#xb2e8;&#xc5b4;&#xc758; &#xc720;&#xd615;&#xc744; &#xb098;&#xd0c0;&#xb0c5;&#xb2e8;&#xc5b4;&#xac00; &#xd574;&#xb2f9; &#xb9e5;&#xb77d;&#xc5d0;&#xc11c; &#xc4f0;&#xc600;&#xc744; &#xacbd;&#xc6b0;&#xc5d0;&#xb9cc; &#xb300;&#xc38;&#xb420; &#xac83;&#xc785;&#xb2c8;&#xb2e4;. **PAGC Tokens** &#xb97c; &#xc38;&#xc870;&#xd558;&#xc2ed;&#xc2dc;&#xc624;.

**14.1.3.3 &#xc9c0;&#xba85; &#xc0c9;&#xc778; &#xd14c;&#xc774;&#xbe14;**

&#xc9c0;&#xba85; &#xc0c9;&#xc778; &#xd14c;&#xc774;&#xbe14; — &#xc9c0;&#xba85; &#xc0c9;&#xc778;(gaz) &#xd14c;&#xc9c0;&#xba85;&#xc744; &#xd45c;&#xc900;&#xd654;&#xd558;&#xace0;, &#xd574;&#xb2f9; &#xc785;&#xb825;&#xc744; (1) &#xc785;&#xb825; &#xd1a0;&#xd070;(the section called “&#xc785;&#xb825; &#xd1a0;&#xd070;” &#xc38;&#xc870;) &#xcbc0f; (2) &#xd45c;&#xc900;&#xd654;&#xb41c; &#xd45c;&#xd604;&#xc2dd;&#xacfc; &#xc5f0;&#xad00;&#xc9d3;&#xb294; &#xb370; &#xc4f0;&#xc785;&#xb2c8;&#xb2e4;.

**&#xc124;&#xba85;**

A gaz (short for gazeteer) table is used to standardize place names and associate that input with the section called “&#xc785;&#xb825; &#xd1a0;&#xd070;” and (b) standardized representations. For example if you are in US, you may load these with State Names and associated abbreviations.

&#xc9c0;&#xba85; &#xc0c9;&#xc778; &#xd14c;&#xc774;&#xbe14;&#xc740; &#xcd5c;&#xc18c;&#xd55c; &#xb2e4;&#xc74c; &#xc5f4;&#xc744; &#xac00;&#xc9c0;&#xace0; &#xc788;&#xc5b4;&#xc57c; &#xd569;&#xb2c8;&#xb2e4;. &#xc0ac;&#xc6a9;&#xc6a9;&#xc38;&#xc5d0; &#xb530;&#xb77c; &#xb354; &#xb9cc;&#xc740; &#xc5f4;&#xc744; &#xcd94;&#xac00;&#xd560; &#xc218;&#xb3c4; &#xc788;&#xc2b5;&#xb2c8;&#xb2e4;.

**id** &#xd14c;&#xc774;&#xbe14;&#xc758; &#xae30;&#xcfc8; &#xd0a4;

**seq** &#xc815;&#xc218;&#xd615;: &#xc815;&#xc758; &#xc88;&#xd638;? - &#xb2e8;&#xc5b4;&#xc758; &#xd574;&#xb2f9; &#xc778;&#xc2a4;&#xd134;&#xc2a4;&#xc5d0; &#xc4f0;&#xc774;&#xb294; &#xc2dd;&#xc38;&#xc790;

**word** `&#xb38;&#xc790;&#xd615;: &#xc785;&#xb825; &#xb2e8;&#xc5b4;`

**stdword** `&#xb38;&#xc790;&#xd615;: &#xd45c;&#xc900;&#xd654;&#xb41c; &#xb300;&#xc4b4; &#xb2e8;&#xc5b4;`

**token** `&#xc815;&#xc218;&#xd615;: &#xb2e8;&#xc5b4;&#xc758; &#xc720;&#xd615;&#xc744; &#xb098;&#xd0c0;&#xb0c5;&#xb2e8;&#xc5b4;&#xac00; &#xd574;&#xb2f9; &#xb9e5;&#xb77d;&#xc5d0;&#xc11c; &#xc4f0;&#xc600;&#xc744; &#xacbd;&#xc6b0;&#xc5d0;&#xb9cc; &#xb300;&#xc4b4;&#xb420; &#xac83;&#xc785;&#xb2c8;&#xb2e4;. PAGC Tokens &#xb97c; &#xcc38;&#xc870;&#xd558;&#xc2ed;&#xc2dc;&#xc624;.`

## 14.1.4 `&#xc8fc;&#xc18c; &#xd45c;&#xc900;&#xd654; &#xb3c4;&#xad6c; &#xd568;&#xc218;`

### 14.1.4.1 `parse_address`

`parse_address` — `&#xd55c; &#xc904;&#xb85c; &#xb41c; &#xc8fc;&#xc18c;&#xb97c; &#xc785;&#xb825;&#xbc1b;&#xc544; &#xbd80;&#xbd84;&#xb4e4;&#xb85c; &#xbd84;&#xd574;&#xd569;&#xb2c8;&#xb2e4;`

#### Synopsis

record `parse_address`(text address);

#### `&#xc124;&#xba85;`

Returns takes an address as input, and returns a record output consisting of fields *num*, *street*, *street2*, *address1*, *city*, *state*, *zip*, *zipplus*, *country*.

2.2.0 `&#xbc84;&#xc804;&#xbd80;&#xd130; &#xc0ac;&#xc6a9;&#xd560; &#xc218; &#xc788;&#xc2b5;&#xb2c8;&#xb2e4;`



This method needs `address_standardizer` extension.

#### `&#xc608;&#xc2dc;`

`&#xb2e8;&#xc77c; &#xc8fc;&#xc18c;`

```
SELECT num, street, city, zip, zipplus
       FROM parse_address('1 Devonshire Place, Boston, MA 02109-1234') AS a;
```

```
num |      street      | city | zip | zipplus
-----+-----+-----+-----+-----
  1  | Devonshire Place | Boston | 02109 | 1234
```

`&#xc8fc;&#xc18c; &#xd14c;&#xc774;&#xbe14;`

```
-- &#xae30;&#xbcf8; &#xd14c;&#xc774;&#xbe14;
CREATE TABLE places(addid serial PRIMARY KEY, address text);

INSERT INTO places(address)
VALUES ('529 Main Street, Boston MA, 02129'),
       ('77 Massachusetts Avenue, Cambridge, MA 02139'),
       ('25 Wizard of Oz, Walaford, KS 99912323'),
       ('26 Capen Street, Medford, MA'),
       ('124 Mount Auburn St, Cambridge, Massachusetts 02138'),
       ('950 Main Street, Worcester, MA 01610');

-- &#xc8fc;&#xc18c; &#xd30c;&#xc2f1;&#xd558;&#xae30;
-- &#xbaa8;&#xb4e0; &#xd56d;&#xbaa9;&#xc744; &#xc6d0;&#xd55c;&#xb2e4;&#xba74; (a).* ↔
   &#xb97c; &#xc4f0;&#xba74; &#xb429;&#xb2c8;&#xb2e4;.
SELECT addid, (a).num, (a).street, (a).city, (a).state, (a).zip, (a).zipplus
FROM (SELECT addid, parse_address(address) As a
      FROM places) AS p;
```



| house_num | name       | suftype | city   | country | state         | unit            |
|-----------|------------|---------|--------|---------|---------------|-----------------|
| 1         | DEVONSHIRE | PLACE   | BOSTON | USA     | MASSACHUSETTS | # PENTHOUSE 301 |

```
TIGER
&#xc624;&#xc54;&#xb529; &#xb3c4;&#xad6c;&#xb85c; &#xd328;&#xd0a4;&#xc9d5;&#xb41c; &#xd14c;&#xc774;&#xc6a9; (&#xc774; &#xc608;&#xc2dc;&#xb294; &#xc0ac;&#xc6a9;&#xc790;&#xac00; postgis_tiger_geocoder
&#xb97c; &#xc124;&#xce58;&#xd588;&#xc744; &#xacbd;&#xc6b0;&#xc5d0;&#xb9cc; &#xb3d9;&#xc791;&#xd569;&#xb2c8;&#
```

```
SELECT * FROM standardize_address('tiger.pagc_lex',
    'tiger.pagc_gaz', 'tiger.pagc_rules', 'One Devonshire Place, PH 301, Boston, MA 02109-1234');
```

```
&#xb354; &#xc54c;&#xc544;&#xbcf4;&#xae30; &#xc27d;&#xac8c; &#xd558;&#xae30; &#xc704;&#xd574; hstore &#xd655;&#xc7
&#xd504;&#xb85c;&#xadf8;&#xb7a8;&#xc744; &#xc774;&#xc6a9;&#xd574;&#xc11c; &#xcd9c;&#xb825;&#xbb3c;&#xc744;
&#xb364;&#xd504;&#xd560; &#xac83;&#xc785;&#xb2c8;&#xb2e4;. &#xc0ac;&#xc6a9;&#xc790;&#xac00; CREATE EXTENSION
hstore; &#xba85;&#xb839;&#xc73c;&#xb85c; &#xc124;&#xce58;&#xd574;&#xc57c; &#xd569;&#xb2c8;&#xb2e4;.
```

```
SELECT (each(hstore(p))).*
FROM standardize_address('tiger.pagc_lex', 'tiger.pagc_gaz',
    'tiger.pagc_rules', 'One Devonshire Place, PH 301, Boston, MA 02109') As p;
```

| key        | value           |
|------------|-----------------|
| box        |                 |
| city       | BOSTON          |
| name       | DEVONSHIRE      |
| qual       |                 |
| unit       | # PENTHOUSE 301 |
| extra      |                 |
| state      | MA              |
| predir     |                 |
| sufdir     |                 |
| country    | USA             |
| pretype    |                 |
| suftype    | PL              |
| building   |                 |
| postcode   | 02109           |
| house_num  | 1               |
| ruralroute |                 |

(16 rows)

```
&#xbcc0;&#xc885; 2: &#xc8fc;&#xc18c;&#xb97c; &#xb450; &#xbd80;&#xbd84;&#xc73c;&#xb85c; &#xc785;&#xb825;&#xbc1b;&
```

```
SELECT (each(hstore(p))).*
FROM standardize_address('tiger.pagc_lex', 'tiger.pagc_gaz',
    'tiger.pagc_rules', 'One Devonshire Place, PH 301', 'Boston, MA 02109, US') As p;
```

| key     | value           |
|---------|-----------------|
| box     |                 |
| city    | BOSTON          |
| name    | DEVONSHIRE      |
| qual    |                 |
| unit    | # PENTHOUSE 301 |
| extra   |                 |
| state   | MA              |
| predir  |                 |
| sufdir  |                 |
| country | USA             |
| pretype |                 |

```

suftype      | PL
building     |
postcode     | 02109
house_num    | 1
ruralroute   |
(16 rows)

```

**stdaddr, Page\_Normalize\_Address**

stdaddr, Page\_Normalize\_Address

## 14.2 TIGER

TIGER

- Nominatim**, OpenStreetMap, PostgreSQL 8.4, PostGIS 1.5, TIGER
- GIS Graphy**, OSM (OpenStreetMap), Nominatim, Java 1.5, Servlet apps, Solr

### 14.2.1 Drop\_Indexes\_Generate\_Script

Drop\_Indexes\_Generate\_Script — TIGER

## Synopsis

```
text Drop_Indexes_Generate_Script(text param_schema=tiger_data);
```

### Parameters

**param\_schema** text  
 TIGER schema name. Default is tiger\_data.

**param\_indexes** text  
 List of indexes to drop. Default is all indexes in the schema. If the list is empty, all indexes in the schema will be dropped. The list can contain the following values:
 

- all: Drop all indexes in the schema.
- bloat: Drop the bloat index.
- install\_missing\_indexes: Drop the install\_missing\_indexes index.
- tiger\_data: Drop all indexes in the tiger\_data schema.

2.0.0: Drop the tiger\_data schema.

### Return Value

```
SELECT drop_indexes_generate_script() As actionsql;
actionsql
-----
DROP INDEX tiger.idx_tiger_countysub_lookup_lower_name;
DROP INDEX tiger.idx_tiger_edges_countyfp;
DROP INDEX tiger.idx_tiger_faces_countyfp;
DROP INDEX tiger.tiger_place_the_geom_gist;
DROP INDEX tiger.tiger_edges_the_geom_gist;
DROP INDEX tiger.tiger_state_the_geom_gist;
DROP INDEX tiger.idx_tiger_addr_least_address;
DROP INDEX tiger.idx_tiger_addr_tlid;
DROP INDEX tiger.idx_tiger_addr_zip;
DROP INDEX tiger.idx_tiger_county_countyfp;
DROP INDEX tiger.idx_tiger_county_lookup_lower_name;
DROP INDEX tiger.idx_tiger_county_lookup_snd_name;
DROP INDEX tiger.idx_tiger_county_lower_name;
DROP INDEX tiger.idx_tiger_county_snd_name;
DROP INDEX tiger.idx_tiger_county_the_geom_gist;
DROP INDEX tiger.idx_tiger_countysub_lookup_snd_name;
DROP INDEX tiger.idx_tiger_cousub_countyfp;
DROP INDEX tiger.idx_tiger_cousub_cousubfp;
DROP INDEX tiger.idx_tiger_cousub_lower_name;
DROP INDEX tiger.idx_tiger_cousub_snd_name;
DROP INDEX tiger.idx_tiger_cousub_the_geom_gist;
DROP INDEX tiger_data.idx_tiger_data_ma_addr_least_address;
DROP INDEX tiger_data.idx_tiger_data_ma_addr_tlid;
DROP INDEX tiger_data.idx_tiger_data_ma_addr_zip;
DROP INDEX tiger_data.idx_tiger_data_ma_county_countyfp;
DROP INDEX tiger_data.idx_tiger_data_ma_county_lookup_lower_name;
DROP INDEX tiger_data.idx_tiger_data_ma_county_lookup_snd_name;
DROP INDEX tiger_data.idx_tiger_data_ma_county_lower_name;
DROP INDEX tiger_data.idx_tiger_data_ma_county_snd_name;
:
:
```



**Loader\_Generate\_Script**

```

SELECT drop_state_tables_generate_script('PA');
DROP TABLE tiger_data.pa_addr;
DROP TABLE tiger_data.pa_county;
DROP TABLE tiger_data.pa_county_lookup;
DROP TABLE tiger_data.pa_cousub;
DROP TABLE tiger_data.pa_edges;
DROP TABLE tiger_data.pa_faces;
DROP TABLE tiger_data.pa_featnames;
DROP TABLE tiger_data.pa_place;
DROP TABLE tiger_data.pa_state;
DROP TABLE tiger_data.pa_zip_lookup_base;
DROP TABLE tiger_data.pa_zip_state;
DROP TABLE tiger_data.pa_zip_state_loc;

```

2.0.0

**Loader\_Generate\_Script**

```

SELECT drop_state_tables_generate_script('PA');
DROP TABLE tiger_data.pa_addr;
DROP TABLE tiger_data.pa_county;
DROP TABLE tiger_data.pa_county_lookup;
DROP TABLE tiger_data.pa_cousub;
DROP TABLE tiger_data.pa_edges;
DROP TABLE tiger_data.pa_faces;
DROP TABLE tiger_data.pa_featnames;
DROP TABLE tiger_data.pa_place;
DROP TABLE tiger_data.pa_state;
DROP TABLE tiger_data.pa_zip_lookup_base;
DROP TABLE tiger_data.pa_zip_state;
DROP TABLE tiger_data.pa_zip_state_loc;

```

**Loader\_Generate\_Script****Loader\_Generate\_Script****14.2.4 Geocode**

Geocode —

**Synopsis**

setof record **geocode**(varchar address, integer max\_results=10, geometry restrict\_region=NULL, norm\_addy OUT addy, geometry OUT geomout, integer OUT rating);

setof record **geocode**(norm\_addy in\_addy, integer max\_results=10, geometry restrict\_region=NULL, norm\_addy OUT addy, geometry OUT geomout, integer OUT rating);

**Normalized address**

normalized\_address (addy) ... TIGER ... PostgreSQL ...

levenshtein ... TIGER ... PostgreSQL ...

**Geocode**

SELECT g.rating, ST\_X(g.geomout) As lon, ST\_Y(g.geomout) As lat, (addy).address As stno, (addy).streetname As street, (addy).streettypeabbrev As styp, (addy).location As city, (addy).stateabbrev As st, (addy).zip FROM geocode('75 State Street, Boston MA 02109') As g;

| rating | lon               | lat              | stno | street | styp | city   | st | zip   |
|--------|-------------------|------------------|------|--------|------|--------|----|-------|
| 0      | -71.0556722990239 | 42.3589914927049 | 75   | State  | St   | Boston | MA | 02109 |

SELECT g.rating, ST\_AsText(ST\_SnapToGrid(g.geomout,0.00001)) As wktmlonlat, (addy).address As stno, (addy).streetname As street, (addy).streettypeabbrev As styp, (addy).location As city, (addy).stateabbrev As st, (addy).zip FROM geocode('226 Hanover Street, Boston, MA',1) As g;

| rating | wktlonlat                 | stno | street  | styp | city   | st | zip   |
|--------|---------------------------|------|---------|------|--------|----|-------|
| 1      | POINT(-71.05528 42.36316) | 226  | Hanover | St   | Boston | MA | 02113 |

```

&#xc0;&#xc790; &#xc624;&#xb958;&#xb3c4; &#xc98;&#xb9ac;&#xd560; &#xc218; &#xc788;&#xb294;&#xb370; &#xd558;&
&#xc774;&#xc0c1;&#xc758; &#xac00;&#xb2a5;&#xc131; &#xc788;&#xb294; &#xb2f5;&#xc744; &#xc21c;&#xc704;&#xc640;
&#xd568;&#xaed8; &#xc0dd;&#xc131;&#xd558;&#xae30; &#xb54c;&#xbb38;&#xc5d0; &#xc2dc;&#xac04;&#xc774; &#xb354;
&#xac78;&#xb9bd;&#xb2c8;&#xb2e4;(500&#xbc00;&#xb9ac;&#xcd08;).

```

```

SELECT g.rating, ST_AsText(ST_SnapToGrid(g.geomout,0.00001)) As wktlonlat,
      (addy).address As stno, (addy).streetname As street,
      (addy).streettypeabbrev As styp, (addy).location As city, (addy).stateabbrev As st, ( ←
      addy).zip
FROM geocode('31 - 37 Stewart Street, Boston, MA 02116') As g;
rating |          wktlonlat          | stno | street | styp | city | st | zip
-----+-----+-----+-----+-----+-----+-----+-----
70 | POINT(-71.06459 42.35113) | 31 | Stuart | St | Boston | MA | 02116

```

```

&#xc8fc;&#xc18c;&#xb4e4;&#xc744; &#xc9c0;&#xc624;&#xcf54;&#xb529;&#xd558;&#xb294; &#xb370; &#xbc30;&#xce58;(bat
&#xc791;&#xc5c5;&#xc744; &#xc774;&#xc6a9;&#xd569;&#xb2c8;&#xb2e4;. max_results = 1&#xb85c; &#xc124;&#xc815;
&#xd3b8;&#xc774; &#xac00;&#xc7a5; &#xc27d;&#xc2b5;&#xb2c8;&#xb2e4;. &#xc544;&#xc9c1; &#xc9c0;&#xc624;&#xcf54;&#
&#xc54a;&#xc740; (&#xc21c;&#xc704;&#xac00; &#xc5c6;&#xb294;) &#xc8fc;&#xc18c;&#xb4e4;&#xb9cc; &#xc98;&#xb9ac;&#

```

```

CREATE TABLE addresses_to_geocode(addid serial PRIMARY KEY, address text,
      lon numeric, lat numeric, new_address text, rating integer);

```

```

INSERT INTO addresses_to_geocode(address)
VALUES ('529 Main Street, Boston MA, 02129'),
      ('77 Massachusetts Avenue, Cambridge, MA 02139'),
      ('25 Wizard of Oz, Walaford, KS 99912323'),
      ('26 Capen Street, Medford, MA'),
      ('124 Mount Auburn St, Cambridge, Massachusetts 02138'),
      ('950 Main Street, Worcester, MA 01610');

```

```

-- &#xc98;&#xc74c; &#xc138; &#xc8fc;&#xc18c;&#xb9cc; ←
&#xc5c5;&#xb370;&#xc774;&#xd2b8;&#xd569;&#xb2c8;&#xb2e4;(323 ~ 704 ←
&#xbc00;&#xb9ac;&#xcd08; - &#xce90;&#xc2f1; &#xbc0f; &#xacf5;&#xc720; ←
&#xba54;&#xbaa8;&#xb9ac; &#xc601;&#xd5a5; &#xb54c;&#xbb38;&#xc5d0; &#xccab; ←
&#xc9c0;&#xc624;&#xcf54;&#xb529; &#xc791;&#xc5c5;&#xc740; &#xd56d;&#xc0c1; ←
&#xb290;&#xb9bd;&#xb2c8;&#xb2e4;). --
-- &#xc8fc;&#xc18c;&#xac00; &#xb300;&#xb7c9;&#xc77c; &#xacbd;&#xc6b0; ←
&#xd55c;&#xbc88;&#xc5d0; &#xc5c5;&#xb370;&#xc774;&#xd2b8;&#xd558;&#xc9c0; ←
&#xc54a;&#xb294; &#xd3b8;&#xc774; &#xc88b;&#xc2b5;&#xb2c8;&#xb2e4;.
-- &#xc804;&#xccb4; &#xc9c0;&#xc624;&#xcf54;&#xb529; &#xc791;&#xc5c5;&#xc744; ←
&#xd55c;&#xbc88;&#xc5d0; &#xcee4;&#xbc0b;&#xd574;&#xc57c;&#xb9cc; &#xd558;&#xae30; ←
&#xb54c;&#xbb38;&#xc785;&#xb2c8;&#xb2e4;.
-- &#xc774; &#xc608;&#xc2dc;&#xc758; &#xacbd;&#xc6b0; LEFT JOIN&#xc73c;&#xb85c; ←
&#xc7ac;&#xacb0;&#xd569;&#xc2dc;&#xd0a4;&#xace0;
-- &#xb098;&#xc05c; &#xc8fc;&#xc18c;&#xb97c; &#xb2e4;&#xc2dc; ←
&#xc9c0;&#xc624;&#xcf54;&#xb529; &#xc791;&#xc5c5;&#xd558;&#xc9c0; &#xc54a;&#xae30; ←
&#xc704;&#xd574;
-- &#xc77c;&#xce58;&#xd558;&#xc9c0; &#xc54a;&#xc744; &#xacbd;&#xc6b0; ←
&#xc21c;&#xc704;&#xb97c; -1&#xb85c; &#xc124;&#xc815;&#xd569;&#xb2c8;&#xb2e4;.
UPDATE addresses_to_geocode
SET (rating, new_address, lon, lat)
= ( COALESCE((g.geo).rating,-1), pprint_addy((g.geo).addy),
      ST_X((g.geo).geomout)::numeric(8,5), ST_Y((g.geo).geomout)::numeric(8,5) )
FROM (SELECT addid
      FROM addresses_to_geocode
      WHERE rating IS NULL ORDER BY addid LIMIT 3) As a
LEFT JOIN (SELECT addid, (geocode(address,1)) As geo

```





**Geocode, Pprint\_Addy, ST\_AsText**

**Geocode, Pprint\_Addy, ST\_AsText**

## 14.2.6 Get\_Geocode\_Setting

Get\_Geocode\_Setting — tiger.geocode\_settings

### Synopsis

text Get\_Geocode\_Setting(text setting\_name);

**tiger.geocode\_settings**

tiger.geocode\_settings

```

tiger.geocode_settings
  debug_geocode_address          | false | boolean | debug | <
  debug_geocode_intersection     | false | boolean | debug | <
  debug_normalize_address        | false | boolean | debug | <
  debug_reverse_geocode          | false | boolean | debug | <
  reverse_geocode_numbered_roads | 0     | integer | rating | <
  use_pagc_address_parser        | false | boolean | normalize | <
  normalize_address              | <

```

```
address_standardizer <
  &#xc774;&#xc6a9;&#xd558;&#xb824; &#xd560; &#xac83;&#xc785;&#xb2c8;&#xb2e4; .
```

2.2.0 geocode\_settings\_default &#xc774;&#xae30;&#xbcf8; &#xc124;&#xc815;&#xac12;&#xc744; &#xc800;&#xc7a5;&#xd569;&#xb2c8;&#xb2e4;. &#xc0ac;&#xc6a9;&#xc9c0;&#xc815; &#xc124;&#xc815;&#xac12;&#xc740; geocode\_settings &#xd14c;&#xc774;&#xbe14;&#xc5d0; &#xc788;&#xc774; geocode\_settings &#xd14c;&#xc774;&#xbe14;&#xc5d0;&#xb294; &#xc0ac;&#xc6a9;&#xc790;&#xac00; &#xc124;&#xc815;&#xac12;&#xb9cc; &#xc800;&#xc7a5;&#xd569;&#xb2c8;&#xb2e4;.

2.1.0 &#xc804;&#xbd80;&#xd130; &#xc0ac;&#xc6a9;&#xd560; &#xc218; &#xc788;&#xc2b5;&#xb2c8;&#xb2e4;.

**&#xc608;&#xc2dc;: &#xb514;&#xbc84;&#xae45; &#xc124;&#xc815;&#xac12; &#xbc18;&#xd658;**

```
SELECT get_geocode_setting('debug_geocode_address') As result;
result
-----
false
```

**&#xcc38;&#xace0;**

[Set\\_Geocode\\_Setting](#)

## 14.2.7 Get\_Tract

Get\_Tract — &#xb3c4;&#xd615;&#xc774; &#xc704;&#xce58;&#xd574; &#xc788;&#xb294; &#xad6c;&#xc5ed;(tract) &#xd14c;&#xc778;&#xad6c;&#xc870;&#xc0ac; &#xad6c;&#xc5ed; &#xb610;&#xb294; &#xd604;&#xc7a5;(field)&#xc744; &#xbc18;&#xd658;&#xae30;&#xbcf8;&#xc801;&#xc73c;&#xb85c; &#xad6c;&#xc5ed;&#xc758; &#xcd95;&#xc57d;&#xb41c; &#xba85;&#xce6d;&#xbc18;&#xd658;&#xd569;&#xb2c8;&#xb2e4;.

### Synopsis

text **get\_tract**(geometry loc\_geom, text output\_field=name);

**&#xc124;&#xba85;**

&#xb3c4;&#xd615;&#xc744; &#xc785;&#xb825;&#xbc1b;&#xc544; &#xd574;&#xb2f9; &#xb3c4;&#xd615;&#xc774; &#xc704;&#xc778;&#xad6c;&#xc870;&#xc0ac; &#xad6c;&#xc5ed;&#xc744; &#xbc18;&#xd658;&#xd569;&#xb2c8;&#xb2e4;. &#xacf5;&#xc778;&#xc870; &#xc2dc;&#xc2a4;&#xd15c;&#xc744; &#xb530;&#xb85c; &#xc124;&#xc815;&#xd558;&#xc9c0; &#xc54a;&#xacbd;&#xc6b0; NAD83 &#xacbd;&#xc704;&#xb3c4;&#xb77c;&#xace0; &#xac00;&#xc815;&#xd569;&#xb2c8;&#xb2e4;.

#### Note

This function uses the census `tract` which is not loaded by default. If you have already loaded your state table, you can load `tract` as well as `bg`, and `tabblock` using the [Loader\\_Generate\\_Census\\_Script](#) script.

If you have not loaded your state data yet and want these additional tables loaded, do the following

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

then they will be included by the [Loader\\_Generate\\_Script](#).

2.0.0 &#xc804;&#xbd80;&#xd130; &#xc0ac;&#xc6a9;&#xd560; &#xc218; &#xc788;&#xc2b5;&#xb2c8;&#xb2e4;.

**&#xae30;&#xbcf8; &#xc608;&#xc2dc;**

```
SELECT get_tract(ST_Point(-71.101375, 42.31376) ) As tract_name;
tract_name
-----
1203.01
```

```
-- &#xc774; &#xcffc;&#xb9ac;&#xb294; TIGER &#xb3c4;&#xd615; ID(geoid)&#xb97c; ←
   &#xb18;&#xd658;&#xd569;&#xb2c8;&#xb2e4;.
SELECT get_tract(ST_Point(-71.101375, 42.31376), 'tract_id' ) As tract_id;
tract_id
-----
25025120301
```

**&#xcc38;&#xace0;**

[Geocode >](#)

**14.2.8 Install\_Missing\_Indexes**

Install\_Missing\_Indexes — &#xc9c0;&#xc624;&#xc5f4;&#xb529; &#xb3c4;&#xad6c;&#xc758; &#xacb0;&#xd569;(join)&#xc5d0; &#xc774;&#xc6a9;&#xb418;&#xb294; &#xd0a4;(key) &#xc5f4;&#xc744; &#xac00;&#xc9c4; &#xd14c;&#xc774;&#xbe14;&#xc744; &#xaa8;&#xb450; &#xcc3e;&#xc544;&#xc11c; &#xd574;&#xb2f9; &#xc5f4;&#xc5d0; &#xc4f0;&#xc778; &#xc778;&#xb371;&#xc778; &#xc00;&#xc6b4;&#xb370; &#xc5c6;&#xc5b4;&#xc9c4; &#xac83;&#xb4e4;&#xc5d0; &#xb300;&#xd55c; &#xc870;&#xac74;&#xc778; &#xd544;&#xd130;&#xb9c1;&#xd574;&#xc11c; &#xc5c6;&#xc5b4;&#xc9c4; &#xc778;&#xb371;&#xc2a4;&#xb4e4;&#xc744; &#xcd94;&#xac00;&#xd569;&#xb2c8;&#xb2e4;.

**Synopsis**

boolean Install\_Missing\_Indexes();

**&#xc124;&#xba85;**

tiger &#xbcf8; tiger\_data &#xc2a4;&#xd0a4;&#xb9c8;&#xc5d0;&#xc11c; &#xc9c0;&#xc624;&#xc5f4;&#xb529; &#xb3c4;&#xad6c;&#xc758; &#xacb0;&#xd569;&#xc5d0; &#xc774;&#xc6a9;&#xb418;&#xb294; &#xd0a4; &#xc5f4;&#xc744; &#xac00;&#xc9c4; &#xd14c;&#xc774;&#xbe14;&#xc744; &#xaa8;&#xb450; &#xcc3e;&#xc740; &#xb2e4;&#xc74c;, &#xd574;&#xb2f9; &#xc5f4;&#xc5d0;&#xc11c; &#xc5c6;&#xc5b4;&#xc778;&#xb371;&#xc2a4;&#xb4e4;&#xc744; &#xd544;&#xd130;&#xb9c1;&#xd574;&#xc11c;, &#xd574;&#xb2f9; &#xd14c;&#xc744; &#xd55c; &#xc778;&#xb371;&#xc2a4;&#xb97c; &#xc815;&#xc758;&#xd558;&#xae30; &#xc704;&#xd55c; SQL DDL&#xc744; &#xcd9c;&#xb825;&#xd558;&#xace0;, &#xc0dd;&#xc131;&#xb41c; &#xc2a4;&#xd06c;&#xb9bd;&#xd2b8;&#xb97c; &#xc2e4;&#xd589;&#xd569;&#xb2c8;&#xb2e4;. &#xc774; &#xd568;&#xc218;&#xb294; &#xb85c;&#xb4dc; &#xacfc;&#xc815;&#xc5c6;&#xc5b4;&#xc84c;&#xc744; &#xc218;&#xb3c4; &#xc788;&#xb294;, &#xcffc;&#xb9ac;&#xb97c; &#xb354; &#bbe68;&#xd558;&#xae30; &#xc704;&#xd574; &#xd544;&#xc694;&#xd55c; &#xc0c8; &#xc778;&#xb371;&#xc2a4;&#xb4e4;&#xc744; &#xcd94;&#xac00;&#xd558;&#xb294; &#xb3c4;&#xc6b0;&#xbbf8; &#xd568;&#xc218;&#xc785;&#xb2c8;&#xb2e4;. &#xc774; &#xd568;&#xc218;&#xb294; &#xc778;&#xb371;&#xc2a4; &#xc0dd;&#xc131; &#xc2a4;&#xd06c;&#xb9bd;&#xd2b8;&#xb97c; &#xc791;&#xc131;&#xd558;&#xb294; **Missing\_Indexes\_Generate\_Script** &#xd568;&#xc218;&#xc640; &#xb3d9;&#xc77c;&#xd55c; &#xacc4;&#xc5f4;&#xb85c;, &#xc791;&#xc131;&#xb41c; &#xc2a4;&#xd06c;&#xb9bd;&#xd2b8;&#xb97c; &#xc2e4;&#xd589;&#xd569;&#xb2c8;&#xb2e4;. update\_geocode.sql &#xc5c5;&#xadf8;&#xb808;&#xc774;&#xb4dc; &#xc2a4;&#xd06c;&#xb97c; &#xb80;&#xb85c; &#xc774; &#xd568;&#xc218;&#xb97c; &#xd638;&#xcd9c;&#xd569;&#xb2c8;&#xb2e4;.

2.0.0 &#xb84;&#xc804;&#xbd80;&#xd130; &#xc0ac;&#xc6a9;&#xd560; &#xc218; &#xc788;&#xc2b5;&#xb2c8;&#xb2e4;.

**Loader\_Generate\_Script**

```
SELECT install_missing_indexes();
       install_missing_indexes
-----
t
```

**Loader\_Generate\_Script**

[Loader\\_Generate\\_Script](#), [Missing\\_Indexes\\_Generate\\_Script](#)

**14.2.9 Loader\_Generate\_Census\_Script**

**Loader\_Generate\_Census\_Script** — `install_missing_indexes(` `install_missing_indexes` `-----` `t`

`Loader_Generate_Census_Script` — `install_missing_indexes(` `install_missing_indexes` `-----` `t`

**Synopsis**

```
setof text loader_generate_census_script(text[] param_states, text os);
```

**Loader\_Generate\_Census\_Script**

`Loader_Generate_Census_Script` — `install_missing_indexes(` `install_missing_indexes` `-----` `t`

`Loader_Generate_Census_Script` — `install_missing_indexes(` `install_missing_indexes` `-----` `t`

`Loader_Generate_Census_Script` — `install_missing_indexes(` `install_missing_indexes` `-----` `t`

- `loader_variables` - `install_missing_indexes(` `install_missing_indexes` `-----` `t`
- `loader_platform` - `install_missing_indexes(` `install_missing_indexes` `-----` `t`



```
%PSQL% -c "ALTER TABLE tiger_staging.MA_tract10 RENAME geoid10 TO tract_id; SELECT ←
  loader_load_staged_data(lower('MA_tract10'), lower('MA_tract')); "
%PSQL% -c "CREATE INDEX tiger_data_MA_tract_the_geom_gist ON tiger_data.MA_tract USING gist ←
  (the_geom);"
%PSQL% -c "VACUUM ANALYZE tiger_data.MA_tract;"
%PSQL% -c "ALTER TABLE tiger_data.MA_tract ADD CONSTRAINT chk_statefp CHECK (statefp = ←
  '25');"
:
```

```
.sh
```

```
STATEDIR="/gisdata/www2.census.gov/geo/pvs/tiger2010st/25_Massachusetts"
TMPDIR="/gisdata/temp/"
UNZIPTOOL=unzip
WGETTOOL="/usr/bin/wget"
export PGBIN="/usr/pgsql-9.0/bin"
export PGPORT=5432
export PGHOST=localhost
export PGUSER=postgres
export PGPASSWORD=yourpasswordhere
export PGDATABASE=geocoder
PSQL=${PGBIN}/psql
SHP2PGSQL=${PGBIN}/shp2pgsql
cd /gisdata

wget http://www2.census.gov/geo/pvs/tiger2010st/25_Massachusetts/25/ --no-parent --relative ←
  --accept=*bg10.zip,*tract10.zip,*tabblock10.zip --mirror --reject=html
rm -f ${TMPDIR}/*. *
${PSQL} -c "DROP SCHEMA tiger_staging CASCADE;"
${PSQL} -c "CREATE SCHEMA tiger_staging;"
cd $STATEDIR
for z in *.zip; do $UNZIPTOOL -o -d $TMPDIR $z; done
:
```

```
&
```

[Loader\\_Generate\\_Script](#)

## 14.2.10 Loader\_Generate\_Script

Loader\_Generate\_Script — TIGER 2010

### Synopsis

```
setof text loader_generate_script(text[] param_states, text os);
```



**Using psql**

Using psql where gistest is your database and /gisdata/data\_load.sh is the file to create with the shell commands to run.

```
psql -U postgres -h localhost -d gistest -A -t \
-c "SELECT Loader_Generate_Script (ARRAY['MA'], 'gistest') " > /gisdata/data_load.sh;
```

```
SELECT loader_generate_script (ARRAY['MA','RI'], 'windows') AS result;
-- result --
set TMPDIR=\gisdata\temp\
set UNZIPTOOL="C:\Program Files\7-Zip\7z.exe"
set WGETTOOL="C:\wget\wget.exe"
set PGBIN=C:\Program Files\PostgreSQL\9.4\bin\
set PGPORT=5432
set PGHOST=localhost
set PGUSER=postgres
set PGPASSWORD=yourpasswordhere
set PGDATABASE=geocoder
set PSQL="%PGBIN%psql"
set SHP2PGSQL="%PGBIN%shp2pgsql"
cd \gisdata
```

```
cd \gisdata
%WGETTOOL% ftp://ftp2.census.gov/geo/tiger/TIGER2015/PLACE/tl_*_25_* --no-parent --relative ←
--recursive --level=2 --accept=zip --mirror --reject=html
cd \gisdata/ftp2.census.gov/geo/tiger/TIGER2015/PLACE
:
:
```

```
.sh
STATEDIR="/gisdata/www2.census.gov/geo/pvs/tiger2010st/25_Massachusetts"
TMPDIR="/gisdata/temp/"
UNZIPTOOL=unzip
WGETTOOL="/usr/bin/wget"
export PGBIN=/usr/pgsql-9.0/bin
export PGPORT=5432
export PGHOST=localhost
export PGUSER=postgres
export PGPASSWORD=yourpasswordhere
export PGDATABASE=geocoder
PSQL=${PGBIN}/psql
SHP2PGSQL=${PGBIN}/shp2pgsql
cd /gisdata
```

```
wget http://www2.census.gov/geo/pvs/tiger2010st/25_Massachusetts/25/ --no-parent --relative ←
--accept=*bg10.zip,*tract10.zip,*tabblock10.zip --mirror --reject=html
rm -f ${TMPDIR}/*. *
${PSQL} -c "DROP SCHEMA tiger_staging CASCADE;"
${PSQL} -c "CREATE SCHEMA tiger_staging;"
cd $STATEDIR
for z in *.zip; do $UNZIPTOOL -o -d $TMPDIR $z; done
:
:
```

**Section 2.4.1**

Section 2.4.1, [Pprint\\_Addy, ST\\_AsText](#)

## 14.2.11 Loader\_Generate\_Nation\_Script

Loader\_Generate\_Nation\_Script — tiger\_data, county\_all, county\_all\_lookup, state\_all, tiger, county, county\_lookup, state, tiger\_loader\_platform, tiger\_loader\_variables, tiger\_loader\_lookuptables

### Synopsis

```
text loader_generate_nation_script(text os);
```

#### Options

county\_all, county\_all\_lookup, state\_all, tiger, county, county\_lookup, state, tiger\_loader\_platform, tiger\_loader\_variables, tiger\_loader\_lookuptables

unzip (7-zip); wget; Section 4.7.2

OS, tiger\_loader\_platform, tiger\_loader\_variables, tiger\_loader\_lookuptables

1. loader\_variables - (staging)
2. loader\_platform -
3. loader\_lookuptables -

Enhanced: 2.4.1 zip code 5 tabulation area (zcta5) load step was fixed and when enabled, zcta5 data is loaded as a single table called zcta5\_all as part of the nation script load.

2.1.0

**Note**

If you want zip code 5 tabulation area (zcta5) to be included in your nation script load, do the following:

```
UPDATE tiger.loader_lookuptables SET load = true WHERE table_name = 'zcta510';
```

**Note**

```
&#xc0ac;&#xc6a9;&#xc790;&#xac00;tiger_2010 &#xbc84;&#xc804;&#xc744; &#xc0ac;&#xc6a9;&#xc911;&#xc774;&#xc
&#xc8fc;&#xc5dde;&#xb97c; tiger_2011 &#xbc84;&#xc804;&#xc73c;&#xb85c; &#xb2e4;&#xc2dc;
&#xb85c;&#xb4dc;&#xd558;&#xb824; &#xd560; &#xacbd;&#xc6b0; &#xc774; &#xc2a4;&#xd06c;&#xb9bd;&#xd2b8;&#xb97c
&#xc2e4;&#xd589;&#xd558;&#xae30; &#xc804;&#xc5d0; &#xc81c;&#xc77c; &#xba3c;&#xc800; &#xc0ad;&#xc81c;
&#xc120;&#xc5b8;&#xbb38; Drop\_Nation\_Tables\_Generate\_Script &#xb97c; &#xc0dd;&#xc131;&#xd558;&#xae0;
&#xc2e4;&#xd589;&#xd574;&#xc57c; &#xd560; &#xac83;&#xc785;&#xb2c8;&#xb2e4;
```

**&#xc608;&#xc2dc;**

```
&#xc708;&#xb3c4;&#xc6b0; &#xc0c1;&#xc5d0;&#xc11c; &#xad6d;&#xac00; &#xb370;&#xc774;&#xd130;&#xb97c; &#xb85c;&#
&#xc704;&#xd55c; &#xc2a4;&#xd06c;&#xb9bd;&#xd2b8;&#xb97c; &#xc0dd;&#xc131;&#xd569;&#xb2c8;&#xb2e4;
```

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

```
&#xb9ac;&#xb205;&#xc2a4;/&#xc720;&#xb2c9;&#xc2a4; &#xc2dc;&#xc2a4;&#xd15c; &#xc0c1;&#xc5d0;&#xc11c; &#xb370;&#
&#xb85c;&#xb4dc;&#xd558;&#xae30; &#xc704;&#xd55c; &#xc2a4;&#xd06c;&#xb9bd;&#xd2b8;&#xb97c; &#xc0dd;&#xc131;&#
```

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

**&#xc38;&#xae0;**

[Loader\\_Generate\\_Script](#), [Missing\\_Indexes\\_Generate\\_Script](#)

**14.2.12 Missing\_Indexes\_Generate\_Script**

Missing\_Indexes\_Generate\_Script — &#xc9c0;&#xc624;&#xcf54;&#xb529; &#xb3c4;&#xad6c;&#xc758; &#xacb0;&#xd569;(join)&#xc774;&#xc6a9;&#xb418;&#xb294; &#xd0a4;(key) &#xc5f4;&#xc744; &#xac00;&#xc9c4; &#xd14c;&#xc774;&#xbe14; &#xac00;&#xc6b4;&#xb370; &#xd574;&#xb2f9; &#xc5f4;&#xc5d0;&#xc11c; &#xc5c6;&#xc5b4;&#xc9c4; &#xc778;&#xb371;&#xbaa8;&#xb450; &#xc3e;&#xc544;&#xc11c; &#xd574;&#xb2f9; &#xd14c;&#xc774;&#xbe14;&#xc5d0; &#xb300;&#xd55c; &#xc778;&#xb371;&#xc2a4;&#xb97c; &#xc815;&#xc758;&#xd558;&#xb294; SQL DDL&#xc744; &#xcd9c;&#xb825;&#xd569;&#

**Synopsis**

```
text Missing_Indexes_Generate_Script();
```

**&#xc124;&#xba85;**

```
tiger &#xbc0f; tiger_data &#xc2a4;&#xd0a4;&#xb9c8;&#xc5d0;&#xc11c; &#xc9c0;&#xc624;&#xcf54;&#xb529; &#xb3c4;&#
&#xacb0;&#xd569;(join)&#xc5d0; &#xc774;&#xc6a9;&#xb418;&#xb294; &#xd0a4;(key) &#xc5f4;&#xc744; &#xac00;&#xc9c4;
&#xd14c;&#xc774;&#xbe14; &#xac00;&#xc6b4;&#xb370; &#xd574;&#xb2f9; &#xc5f4;&#xc5d0;&#xc11c; &#xc5c6;&#xc5b4;&#xc9c4; &#xc778;&#xb371;&#xc2a4;&#xb97c; &#xbaa8;&#xb450; &#xc3e;&#xc544;&#xc11c; &#xd574;&#xb2f9; &#xd14c;&#xc774;&#
&#xb300;&#xd55c; &#xc778;&#xb371;&#xc2a4;&#xb97c; &#xc815;&#xc758;&#xd558;&#xb294; SQL DDL&#xc744; &#xcd9c;&#
&#xc774; &#xd568;&#xc218;&#xb294; &#xb85c;&#xb4dc; &#xacfc;&#xc815;&#xc5d0;&#xc11c; &#xc5c6;&#xc5b4;&#xc84c;&#
&#xc218;&#xb3c4; &#xc788;&#xb294; &#xcffc;&#xb9ac;&#xb97c; &#xb354; &#xb68;&#xb9ac; &#xd558;&#xae30; &#xc704;&#
```

```

&#xd544;&#xc694;&#xd55c; &#xc0c8; &#xc778;&#xb371;&#xc2a4;&#xb4e4;&#xc744; &#xcd94;&#xac00;&#xd558;&#xb294;
&#xb3c4;&#xc6b0;&#xbbbf8; &#xd568;&#xc218;&#xc785;&#xb2c8;&#xb2e4;. &#xc9c0;&#xc624;&#xcf54;&#xb529; &#xb3c4;&#
&#xd5a5;&#xc0c1;&#xb418;&#xba74;. &#xc0ac;&#xc6a9;&#xb418;&#xb294; &#xc0c8; &#xc778;&#xb371;&#xc2a4;&#xb97c;
&#xc218;&#xc6a9;&#xd558;&#xae30; &#xc704;&#xd574; &#xc774; &#xd568;&#xc218;&#xb3c4; &#xc5c5;&#xb370;&#xc774;&#
&#xac83;&#xc785;&#xb2c8;&#xb2e4;. &#xc774; &#xd568;&#xc218;&#xac00; &#xc544;&#xbb34;&#xac83;&#xb3c4; &#xcd9c;&#
&#xc54a;&#xc744; &#xacbd;&#xc6b0;. &#xbaa8;&#xb4e0; &#xd14c;&#xc774;&#xbe14;&#xc774; &#xc774;&#xbbbf8; &#xd0a4;
&#xc778;&#xb371;&#xc2a4;&#xb85c; &#xac04;&#xc8fc;&#xb418;&#xb294; &#xac83;&#xc744; &#xac00;&#xc9c0;&#xace0;
&#xc788;&#xb2e4;&#xb294; &#xb73b;&#xc785;&#xb2c8;&#xb2e4;.

```

```

2.0.0 &#xbc84;&#xc804;&#xbd80;&#xd130; &#xc0ac;&#xc6a9;&#xd560; &#xc218; &#xc788;&#xc2b5;&#xb2c8;&#xb2e4;.

```

## &#xc608;&#xc2dc;

```

SELECT missing_indexes_generate_script();
-- &#xcd9c;&#xb825;&#xbb3c;: &#xb85c;&#xb4dc; &#xc791;&#xc5c5; ←
  &#xc2a4;&#xd06c;&#xb9bd;&#xd2b8;&#xc5d0; &#xb9ce;&#xc740; &#xc218;&#xc815;&#xc774; ←
  &#xac00;&#xd574;&#xc9c0;&#xae30; &#xc804;&#xc5d0; &#xc0dd;&#xc131;&#xb41c; ←
  &#xb370;&#xc774;&#xd130;&#xbca0;&#xc774;&#xc2a4; &#xc0c1;&#xc5d0;&#xc11c; ←
  &#xc2e4;&#xd589;&#xd55c; &#xac83;&#xc785;&#xb2c8;&#xb2e4;. --
CREATE INDEX idx_tiger_county_countyfp ON tiger.county USING btree(countyfp);
CREATE INDEX idx_tiger_cousub_countyfp ON tiger.cousub USING btree(countyfp);
CREATE INDEX idx_tiger_edges_tfidr ON tiger.edges USING btree(tfidr);
CREATE INDEX idx_tiger_edges_tfidl ON tiger.edges USING btree(tfidl);
CREATE INDEX idx_tiger_zip_lookup_all_zip ON tiger.zip_lookup_all USING btree(zip);
CREATE INDEX idx_tiger_data_ma_county_countyfp ON tiger_data.ma_county USING btree(countyfp ←
);
CREATE INDEX idx_tiger_data_ma_cousub_countyfp ON tiger_data.ma_cousub USING btree(countyfp ←
);
CREATE INDEX idx_tiger_data_ma_edges_countyfp ON tiger_data.ma_edges USING btree(countyfp);
CREATE INDEX idx_tiger_data_ma_faces_countyfp ON tiger_data.ma_faces USING btree(countyfp);

```

## &#xcc38;&#xace0;

### Loader\_Generate\_Script, Install\_Missing\_Indexes

## 14.2.13 Normalize\_Address

Normalize\_Address — &#xbb38;&#xc790;&#xd615; &#xb3c4;&#xb85c; &#xc8fc;&#xc18c;&#xb97c; &#xc785;&#xb825;&#xbc1b; &#xb3c4;&#xb85c; &#xc811;&#xbbbf8;&#xc0ac;. &#xc811;&#xb450;&#xc0ac; &#xbc0f; &#xc720;&#xd615;&#xc744; &#xd45c;&# &#xb3c4;&#xb85c;. &#xb3c4;&#xb85c;&#xba85; &#xb4f1;&#xc744; &#xac1c;&#xbcc4; &#xd544;&#xb4dc;&#xb85c; &#xbd84;&# norm\_addy &#xd569;&#xc131; &#xc720;&#xd615;&#xc744; &#xbc18;&#xd658;&#xd569;&#xb2c8;&#xb2e4;. &#xc774; &#xd568;&#xc218;&#xb294; tiger\_geocoder&#xc640; &#xd568;&#xaed8; &#xd328;&#xd0a4;&#xc9d5;&#xb41c; &#xc0c9;&#xc77 &#xb370;&#xc774;&#xd130;&#xb9cc; &#xc774;&#xc6a9;&#xd574;&#xc11c; (TIGER &#xc778;&#xad6c;&#xc870;&#xc0ac; &#xb370;&#xc774;&#xd130;&#xb294; &#xd544;&#xc694;&#xc5c6;&#xc774;) &#xc791;&#xb3d9;&#xd560; &#xac83;&#xc785;&

### Synopsis

```
norm_addy normalize_address(varchar in_address);
```

## &#xc124;&#xba85;

```

&#xbb38;&#xc790;&#xd615; &#xb3c4;&#xb85c; &#xc8fc;&#xc18c;&#xb97c; &#xc785;&#xb825;&#xbc1b;&#xc544;. &#xb3c4;&#
&#xc811;&#xbbbf8;&#xc0ac;. &#xc811;&#xb450;&#xc0ac; &#xbc0f; &#xc720;&#xd615;&#xc744; &#xd45c;&#xc900;&#xd654;&#
&#xb3c4;&#xb85c;. &#xb3c4;&#xb85c;&#xba85; &#xb4f1;&#xc744; &#xac1c;&#xbcc4; &#xd544;&#xb4dc;&#xb85c; &#xbd84;&#

```

```

norm_addy &#xd569;&#xc131; &#xc720;&#xd615;&#xc744; &#xbc18;&#xd658;&#xd569;&#xb2c8;&#xb2e4;. &#xbaa8;&#xb4e4;
&#xc8fc;&#xc18c;&#xb97c; &#xc815;&#xadcc;&#xd654;&#xb41c; &#xc6b0;&#xd3b8; &#xd615;&#xc2dd;&#xc73c;&#xb85c;
&#xbcc0;&#xd658;&#xd558;&#xae30; &#xc704;&#xd55c; &#xc9c0;&#xc624;&#xcf54;&#xb529; &#xacfc;&#xc815;&#xc758;
&#xccab; &#xbc88;&#xc9f8; &#xb2e8;&#xacc4;&#xc785;&#xb2c8;&#xb2e4;. &#xc9c0;&#xc624;&#xcf54;&#xb529; &#xb3c4;&#xc758;
&#xd568;&#xaed8; &#xd328;&#xd0a4;&#xc9d5;&#xb41c; &#xb370;&#xc774;&#xd130; &#xc678;&#xc5d0; &#xb2e4;&#xb978;
&#xb370;&#xc774;&#xd130;&#xb294; &#xd544;&#xc694;&#xd558;&#xc9c0; &#xc54a;&#xc2b5;&#xb2c8;&#xb2e4;.

&#xc774; &#xd568;&#xc218;&#xb294; tiger &#xc2a4;&#xd0a4;&#xb9c8;&#xc5d0; &#xc788;&#xace0; tiger_geocoder&#xc640;
&#xd568;&#xaed8; &#xbbf8;&#xb9ac; &#xb85c;&#xb4dc;&#xb3fc; &#xc788;&#xb294; &#xc5ec;&#xb7ec; &#xbc29;&#xd5a5;/&#xc9c0;
&#xc778; &#xd14c;&#xc774;&#xbe14;&#xb9cc; &#xc774;&#xc6a9;&#xd569;&#xb2c8;&#xb2e4;. &#xb530;&#xb77c;&#xc774;
&#xc774; &#xd568;&#xc218;&#xb97c; &#xc774;&#xc6a9;&#xd558;&#xae30; &#xc704;&#xd574; TIGER &#xc778;&#xad6c;&#xc9c0;
&#xb370;&#xc774;&#xd130; &#xb610;&#xb294; &#xb2e4;&#xb978; &#xcd94;&#xac00;&#xc801;&#xc778; &#xb370;&#xc774;&#xc774;
&#xb2e4;&#xc6b4;&#xb85c;&#xb4dc;&#xd560; &#xd544;&#xc694;&#xac00; &#xc5c6;&#xc2b5;&#xb2c8;&#xb2e4;. tiger
&#xc2a4;&#xd0a4;&#xb9c8;&#xc5d0; &#xc788;&#xb294; &#xc5ec;&#xb7ec; &#xc9c0;&#xc778; &#xd14c;&#xc774;&#xbe14;&#xc774;
&#xb354; &#xb9ce;&#xc740; &#xc57d;&#xc5b4; &#xb610;&#xb294; &#xb300;&#xccb4; &#xba85;&#xce6d;&#xb4e4;&#xc744;
&#xcd94;&#xac00;&#xd574;&#xc57c; &#xd560; &#xd544;&#xc694;&#xac00; &#xc788;&#xc744; &#xc218;&#xb3c4; &#xc788;&#xc774;
&#xd568;&#xc218;&#xb294; &#xc785;&#xb825; &#xc8fc;&#xc18c;&#xb97c; &#xc815;&#xadcc;&#xd654;&#xd558;&#xc704;&#xd574;
tiger &#xc2a4;&#xd0a4;&#xb9c8;&#xc5d0; &#xc788;&#xb294; &#xc5ec;&#xb7ec; &#xc81c;&#xc5b4; &#xc9c0;&#xc778;
&#xd14c;&#xc774;&#xbe14;&#xc744; &#xc774;&#xc6a9;&#xd569;&#xb2c8;&#xb2e4;.

&#xc774; &#xd568;&#xc218;&#xb294; norm_addy &#xc720;&#xd615; &#xac1d;&#xccb4; &#xc548;&#xc758; &#xd544;&#xb4d4;
&#xb2e4;&#xc74c;&#xacfc; &#xac19;&#xc740; &#xc21c;&#xc11c;&#xb85c; &#xbc18;&#xd658;&#xd569;&#xb2c8;&#xb2e4;.
&#xc774;&#xb54c; ()&#xac00; &#xc9c0;&#xc624;&#xcf54;&#xb529; &#xb3c4;&#xad6c;&#xac00; &#xc694;&#xad6c;&#xd558;&#xc774;
&#xd544;&#xb4dc;&#xb97c; &#xb098;&#xd0c0;&#xb0b4;&#xace0;, []&#xb294; &#xc120;&#xd0dd;&#xc801;&#xc778; &#xd544;&#xb098;
&#xd0c0;&#xb0c5;&#xb2c8;&#xb2e4;:

```

(address) [predirAbbrev] (streetName) [streetTypeAbbrev] [postdirAbbrev] [internal] [location] [stateAbbrev] [zip] [parsed] [zip4] [address\_alphanumeric]

Enhanced: 2.4.0 norm\_addy object includes additional fields zip4 and address\_alphanumeric.

1. address &#xb294; &#xc815;&#xc218;&#xd615;&#xc785;&#xb2c8;&#xb2e4;. &#xb3c4;&#xb85c; &#xbc88;&#xc9c0;&#xc758;
2. predirAbbrev &#xb294; varchar&#xd615;&#xc785;&#xb2c8;&#xb2e4;. N, S, E, W &#xb4f1;&#xacfc; &#xac19;&#xc740; &#xb3c4;&#xb85c;&#xc758; &#xbc29;&#xd5a5;&#xc744; &#xc758;&#xbbf8;&#xd558;&#xb294; &#xc811;&#xb450;&#xc000; direction\_lookup &#xd14c;&#xc774;&#xbe14;&#xc744; &#xc774;&#xc6a9;&#xd574;&#xc11c; &#xc774; &#xd544;&#xc81c;&#xc5b4;&#xd569;&#xb2c8;&#xb2e4;.
3. streetName &#xc740; varchar&#xd615;&#xc785;&#xb2c8;&#xb2e4;.
4. streetTypeAbbrev &#xb294; varchar&#xd615;&#xc73c;&#xb85c;. St, Ave, Cir&#xcc98;&#xb7fc; &#xb3c4;&#xb85c; &#xc720;&#xd615;&#xc758; &#xcd95;&#xc57d; &#xbc84;&#xc804;&#xc785;&#xb2c8;&#xb2e4;. street\_type\_lookup &#xd14c;&#xc774;&#xbe14;&#xc744; &#xc774;&#xc6a9;&#xd574;&#xc11c; &#xc774; &#xd544;&#xb4dc;&#xb97c; &#xc81c;&#xc5b4;&#xd569;&#xb2c8;&#xb2e4;.
5. postdirAbbrev &#xb294; varchar&#xd615;&#xc73c;&#xb85c;. N, S, E, W &#xb4f1;&#xacfc; &#xac19;&#xc740; &#xb3c4;&#xb85c;&#xc758; &#xbc29;&#xd5a5;&#xc744; &#xc758;&#xbbf8;&#xd558;&#xb294; &#xc811;&#xbbf8;&#xc000; &#xcd95;&#xc57d; &#xbc84;&#xc804;&#xc785;&#xb2c8;&#xb2e4;. direction\_lookup &#xd14c;&#xc774;&#xbe14;&#xc744; &#xc774;&#xc6a9;&#xd574;&#xc11c; &#xc774; &#xd544;&#xb4dc;&#xb97c; &#xc81c;&#xc5b4;&#xd569;&#xb2c8;&#xb2e4;.
6. internal &#xc740; varchar&#xd615;&#xc785;&#xb2c8;&#xb2e4;. &#xc544;&#xd30c;&#xd2b8; &#xb610;&#xb294; &#xbe4c;&#xb77c;&#xc758; &#xb3d9;&#xd638;&#xc218;&#xc640; &#xac19;&#xc740; &#xb0b4;&#xbd80; &#xc8fc;&#xc11c;
7. location &#xc740; varchar&#xd615;&#xc73c;&#xb85c;. &#xc77c;&#xbc18;&#xc801;&#xc73c;&#xb85c; &#xb3c4;&#xc000; &#xb610;&#xb294; &#xc9c0;&#xc790;&#xccb4;&#xb97c; &#xb098;&#xd0c0;&#xb0c5;&#xb2c8;&#xb2e4;.
8. stateAbbrev &#xb294; varchar&#xd615;&#xc73c;&#xb85c;. MA, NY, MI&#xcc98;&#xb7fc; &#xb450; &#xae00;&#xc790; &#xd45c;&#xd604;&#xd55c; &#xbbf8;&#xad6d;&#xc758; &#xc8fc;&#xba85; (&#x5dde;&#x540d;)&#xc785;&#xb2c8;&#xb2e4; state\_lookup &#xd14c;&#xc774;&#xbe14;&#xc744; &#xc774;&#xc6a9;&#xd574;&#xc11c; &#xc774; &#xd544;&#xb4dc;&#xc81c;&#xc5b4;&#xd569;&#xb2c8;&#xb2e4;.

9. zip varchar(15);(boolean) (02109;);
10. parsed; (boolean); normalize\_address; zip4 last 4 digits of a 9 digit zip code. Availability: PostGIS 2.4.0.
12. address\_alphanumeric Full street number even if it has alpha characters like 17R. Parsing of this is better using [Pgcode\\_Normalize\\_Address](#) function. Availability: PostGIS 2.4.0.

**Example:**

SELECT address As orig, (g.na).streetname, (g.na).streettypeabbrev  
 FROM (SELECT address, normalize\_address(address) As na  
 FROM addresses\_to\_geocode) As g;

| orig  | streetname    | streettypeabbrev |
|---|---------------|------------------|
| 28 Capen Street, Medford, MA                        | Capen         | St               |
| 124 Mount Auburn St, Cambridge, Massachusetts 02138 | Mount Auburn  | St               |
| 950 Main Street, Worcester, MA 01610                | Main          | St               |
| 529 Main Street, Boston MA, 02129                   | Main          | St               |
| 77 Massachusetts Avenue, Cambridge, MA 02139        | Massachusetts | Ave              |
| 25 Wizard of Oz, Walford, KS 99912323               | Wizard of Oz  |                  |

**Example:**

[Pgcode](#), [Pprint\\_Addy](#)

**14.2.14 Pgcode\_Normalize\_Address**

[Pgcode\\_Normalize\\_Address](#) — (varchar(15);(boolean) (02109;);  
 norm\_addy; tiger\_geocoder; address\_standardizer;

**Synopsis**

norm\_addy [Pgcode\\_Normalize\\_Address](#)(varchar in\_address);

**&#xc124;&#xba85;**

&#xbb38;&#xc790;&#xd615; &#xb3c4;&#xb85c; &#xc8fc;&#xc18c;&#xb97c; &#xc785;&#xb825;&#xbc1b;&#xc544;,. &#xb3c4;&#xc811;&#xbbf8;&#xc0ac;,. &#xc811;&#xb450;&#xc0ac; &#xbc0f; &#xc720;&#xd615;&#xc744; &#xd45c;&#xc900;&#xd654;&#xb3c4;&#xb85c;,. &#xb3c4;&#xb85c;&#xba85; &#xb4f1;&#xc744; &#xac1c;&#xbcc4; &#xd544;&#xb4dc;&#xb85c; &#xbd84;&#xc0ac; norm\_addy &#xd569;&#xc131; &#xc720;&#xd615;&#xc744; &#xbc18;&#xd658;&#xd569;&#xb2c8;&#xb2e4;,. &#xbaa8;&#xb4e4; &#xc8fc;&#xc18c;&#xb97c; &#xc815;&#xaddd;&#xd654;&#xb41c; &#xc6b0;&#xd3b8; &#xd615;&#xc2dd;&#xc73c;&#xb85c; &#xbcc0;&#xd658;&#xd558;&#xae30; &#xc704;&#xd55c; &#xc9c0;&#xc624;&#xcf54;&#xb529; &#xacfc;&#xc815;&#xc758; &#xccab; &#xbc88;&#xc9f8; &#xb2e8;&#xacc4;&#xc785;&#xb2c8;&#xb2e4;,. &#xc9c0;&#xc624;&#xcf54;&#xb529; &#xb3c4;&#xd568;&#xaed8; &#xd328;&#xd0a4;&#xc9d5;&#xb41c; &#xb370;&#xc774;&#xd130; &#xc678;&#xc5d0; &#xb2e4;&#xb978; &#xb370;&#xc774;&#xd130;&#xb294; &#xd544;&#xc694;&#xd558;&#xc9c0; &#xc54a;&#xc2b5;&#xb2c8;&#xb2e4;.

&#xc774; &#xd568;&#xc218;&#xb294; tiger &#xc2a4;&#xd0a4;&#xb9c8;&#xc5d0; &#xc788;&#xace0; tiger\_geocoder&#xc640; &#xd568;&#xaed8; &#xbbf8;&#xb9ac; &#xb85c;&#xb4dc;&#xb3fc; &#xc788;&#xb294; &#xc5ec;&#xb7ec; page\_\* &#xc0c9;&#xc744; &#xd14c;&#xc774;&#xbe14;&#xb9cc; &#xc774;&#xc6a9;&#xd569;&#xb2c8;&#xb2e4;,. &#xb530;&#xb77c;&#xc11c; &#xc774; &#xd568;&#xc218;&#xb97c; &#xc774;&#xc6a9;&#xd558;&#xae30; &#xc704;&#xd574; TIGER &#xc778;&#xad6c;&#xc870;&#xc744; &#xb370;&#xc774;&#xd130; &#xb610;&#xb294; &#xb2e4;&#xb978; &#xcd94;&#xac00;&#xc801;&#xc778; &#xb370;&#xc774; &#xb2e4;&#xc6b4;&#xb85c;&#xb4dc;&#xd560; &#xd544;&#xc694;&#xac00; &#xc5c6;&#xc2b5;&#xb2c8;&#xb2e4;,. tiger &#xc2a4;&#xd0a4;&#xb9c8;&#xc5d0; &#xc788;&#xb294; &#xc5ec;&#xb7ec; &#xc0c9;&#xc778; &#xd14c;&#xc774;&#xbe14;&#xb354; &#xb9cc;&#xc740; &#xc57d;&#xc5b4; &#xb610;&#xb294; &#xb300;&#xccb4; &#xba85;&#xce6d;&#xb4e4;&#xc744; &#xcd94;&#xac00;&#xd574;&#xc57c; &#xd560; &#xd544;&#xc694;&#xac00; &#xc788;&#xc744; &#xc218;&#xb3c4; &#xc788;&#xc774; &#xd568;&#xc218;&#xb294; &#xc785;&#xb825; &#xc8fc;&#xc18c;&#xb97c; &#xc815;&#xaddd;&#xd654;&#xd558;&#xc704;&#xd574; tiger &#xc2a4;&#xd0a4;&#xb9c8;&#xc5d0; &#xc788;&#xb294; &#xc5ec;&#xb7ec; &#xc81c;&#xc5b4; &#xc0c9;&#xc778; &#xd14c;&#xc774;&#xbe14;&#xc744; &#xc774;&#xc6a9;&#xd569;&#xb2c8;&#xb2e4;.

&#xc774; &#xd568;&#xc218;&#xb294; norm\_addy &#xc720;&#xd615; &#xac1d;&#xccb4; &#xc548;&#xc758; &#xd544;&#xb4dc; &#xb2e4;&#xc74c;&#xacfc; &#xac19;&#xc740; &#xc21c;&#xc11c;&#xb85c; &#xbc18;&#xd658;&#xd569;&#xb2c8;&#xb2e4;,. &#xc774;&#xb54c; ()&#xac00; &#xc9c0;&#xc624;&#xcf54;&#xb529; &#xb3c4;&#xad6c;&#xac00; &#xc694;&#xad6c;&#xd558;&#xd544;&#xb4dc;&#xb97c; &#xb098;&#xd0c0;&#xb0b4;&#xace0;,. []&#xb294; &#xc120;&#xd0dd;&#xc801;&#xc778; &#xd544;&#xb098;&#xd0c0;&#xb0c5;&#xb2c8;&#xb2e4;:

**Normalize\_Address** &#xc5d0; &#xb300;&#xd574; &#xb300;&#xc18c;&#xbb38;&#xc790; &#xaddd;&#xce59; &#xbc0f; &#xc11c;&#xc0b4;&#xc9dd; &#xb2e4;&#xb978; &#xbcc0;&#xc885;&#xb4e4;&#xc774; &#xc788;&#xc2b5;&#xb2c8;&#xb2e4;.

2.1.0 &#xbc84;&#xc804;&#xbd80;&#xd130; &#xc0ac;&#xc6a9;&#xd560; &#xc218; &#xc788;&#xc2b5;&#xb2c8;&#xb2e4;.

 This method needs address\_standardizer extension.

(address) [predirAbbrev] (streetName) [streetTypeAbbrev] [postdirAbbrev] [internal] [location] [stateAbbrev] [zip]

&#xc774; &#xc2dc;&#xc810;&#xc5d0;&#xc11c; address\_standardizer &#xd655;&#xc7a5; &#xd504;&#xb85c;&#xadf8;&#xb7a8;&#xc6d0;&#xd615; standardaddr&#xb294; norm\_addy&#xbcf4;&#xb2e4; &#xc57d;&#xac04; &#xb2e4;&#xcc44;&#xb86d;&#xc2b5; (&#xad6d;&#xac00;&#xb97c; &#xd3ec;&#xd568;&#xd55c;) &#xad6d;&#xc81c; &#xc8fc;&#xc18c;&#xb97c; &#xc9c0;&#xc6d0;&#xc124;&#xacc4;&#xb410;&#xae30; &#xb54c;&#xbb38;&#xc785;&#xb2c8;&#xb2e4;,. standardaddr&#xc758; &#xc0c1;&#xc751; &#xd544;&#xb4dc;&#xb4e4;&#xc740; &#xb2e4;&#xc74c;&#xacfc; &#xac19;&#xc2b5;&#xb2c8;&#xb2e4;:

house\_num, predir, name, suftype, sufd, unit, city, state, postcode

Enhanced: 2.4.0 norm\_addy object includes additional fields zip4 and address\_alphanumeric.

1. address &#xb294; &#xc815;&#xc218;&#xd615;&#xc785;&#xb2c8;&#xb2e4;:. &#xb3c4;&#xb85c; &#xbc88;&#xc9c0;&#xc774; &#xc774; &#xc2dc;&#xc810;&#xc5d0;&#xc11c; address\_standardizer &#xd655;&#xc7a5; &#xd504;&#xb85c;&#xadf8;&#xb7a8;&#xc6d0;&#xd615; standardaddr&#xb294; norm\_addy&#xbcf4;&#xb2e4; &#xc57d;&#xac04; &#xb2e4;&#xcc44;&#xb86d;&#xc2b5; (&#xad6d;&#xac00;&#xb97c; &#xd3ec;&#xd568;&#xd55c;) &#xad6d;&#xc81c; &#xc8fc;&#xc18c;&#xb97c; &#xc9c0;&#xc6d0;&#xc124;&#xacc4;&#xb410;&#xae30; &#xb54c;&#xbb38;&#xc785;&#xb2c8;&#xb2e4;,. standardaddr&#xc758; &#xc0c1;&#xc751; &#xd544;&#xb4dc;&#xb4e4;&#xc740; &#xb2e4;&#xc74c;&#xacfc; &#xac19;&#xc2b5;&#xb2c8;&#xb2e4;:
2. predirAbbrev &#xb294; varchar&#xd615;&#xc785;&#xb2c8;&#xb2e4;: N, S, E, W &#xb4f1;&#xacfc; &#xac19;&#xc740; &#xb3c4;&#xb85c;&#xc758; &#xbc29;&#xd5a5;&#xc744; &#xc758;&#xbbf8;&#xd558;&#xb294; &#xc811;&#xb450;&#xc0ac; direction\_lookup &#xd14c;&#xc774;&#xbe14;&#xc744; &#xc774;&#xc6a9;&#xd574;&#xc11c; &#xc774; &#xd544;&#xc81c;&#xc5b4;&#xd569;&#xb2c8;&#xb2e4;.
3. streetName &#xc740; varchar&#xd615;&#xc785;&#xb2c8;&#xb2e4;.
4. streetTypeAbbrev &#xb294; varchar&#xd615;&#xc73c;&#xb85c;,. St, Ave, Cir &#xcc98;&#xb7fc; &#xb3c4;&#xb85c; &#xc720;&#xd615;&#xc758; &#xcd95;&#xc57d; &#xbc84;&#xc804;&#xc785;&#xb2c8;&#xb2e4;,. street\_type\_lookup &#xd14c;&#xc774;&#xbe14;&#xc744; &#xc774;&#xc6a9;&#xd574;&#xc11c; &#xc774; &#xd544;&#xb4dc;&#xb97c; &#xc81c;&#xc5b4;&#xd569;&#xb2c8;&#xb2e4;.

5. `postdirabbrev` `varchar(615,73;85c;`, N, S, E, W `4f1;acfc; ac19;740; 3c4;85c;758; bc29;d5a5;744; 758;bbf8;d558;b294; 811;bbf8;c0a d95;57d; bc84;c804;785; b2c8;b2e4;`. `direction_lookup` `d14c;774;be14;774;6a9;d574;c11c; 774; d544;b4dc;b97c; 81c;5b4;d569;b2c8;b2e4;`
6. `internal` `varchar(615;785;b2c8;b2e4;`. `544;d30c;d2b8; b610;b294; be4c;b77c;758; b3d9;d638;c218;c640; ac19;c740; b0b4;bd80; 8fc;c11c;`
7. `location` `varchar(615;73c;85c;`, `77c;bc18;801;73c;85c; b3c4;c2d b610;b294; 9c0;790;c4b4;b97c; b098;d0c0;b0c5;b2c8;b2e4;`.
8. `stateabbrev` `varchar(615;73c;85c;`, MA, NY, MI `cc98;b7fc; b450; ae00;790; d45c;d604;d55c; bbf8;ad6d;758; 8fc;ba85;(5dde;540d;)785;b2c8;b2e4;` `state_lookup` `d14c;774;be14;744; 774;6a9;d574;c11c; 774; d544;b4dc; 81c;5b4;d569;b2c8;b2e4;`.
9. `zip` `varchar(615;785;b2c8;b2e4;`. 02109 `ac19;c740; b2e4;c12f; 790;b99; 9d1;cf54;b4dc;785;b2c8;b2e4;`.
10. `parsed` `boolean(615;785;b2c8;b2e4;`. `815;addc;d654; cc98;b99; 8fc;c18c;b97c; d615;c131;d588;b294; 9c0; 5ec;bd80;b97c; d45c;c2d normalize_address` `d568;c218;b294; 8fc;c18c;b97c; bc18;d658;d558;xae30; c804;c5d0;774; d544;b4dc;b97c; cc38;c73c;85c; c124;c815;d569;b2e4;`
11. `zip4` last 4 digits of a 9 digit zip code. Availability: PostGIS 2.4.0.
12. `address_alphanumeric` Full street number even if it has alpha characters like 17R. Parsing of this is better using `PgC_Normalize_Address` function. Availability: PostGIS 2.4.0.

**Geocode**

`Geocode`

```
SELECT addy.*
FROM pgc_normalize_address('9000 E ROO ST STE 999, Springfield, CO') AS addy;

address | predirabbrev | streetname | streettypeabbrev | postdirabbrev | internal | location | stateabbrev | zip | parsed
-----+-----+-----+-----+-----+-----+-----+-----+-----+-----
9000 | E | ROO | ST | SUITE 999 | SPRINGFIELD | CO | t
```

`postgis_tiger_geocoder`; `address_standardizer`; `Geocode`; `Normalize_Address`; `address_standardizer`; `standardize_address`

```
WITH g AS (SELECT address, ROW((sa).house_num, (sa).predir, (sa).name, (sa).sufytype, (sa).sufdir, (sa).unit, (sa).city, (sa).state, (sa).postcode, true):: norm_addy As na
FROM (SELECT address, standardize_address('tiger.pgc_lex'
```

```

, 'tiger.pagc_gaz'
, 'tiger.pagc_rules', address) As sa
FROM addresses_to_geocode) As g)
SELECT address As orig, (g.na).streetname, (g.na).streettypeabbrev
FROM g;

```

| orig  | streetname    | streettypeabbrev |
|---|---------------|------------------|
| 529 Main Street, Boston MA, 02129                   | MAIN          | ST               |
| 77 Massachusetts Avenue, Cambridge, MA 02139        | MASSACHUSETTS | AVE              |
| 25 Wizard of Oz, Walaford, KS 99912323              | WIZARD OF     |                  |
| 26 Capen Street, Medford, MA                        | CAPEN         | ST               |
| 124 Mount Auburn St, Cambridge, Massachusetts 02138 | MOUNT AUBURN  | ST               |
| 950 Main Street, Worcester, MA 01610                | MAIN          | ST               |

**Normalize\_Address, Geocode**

**Normalize\_Address, Geocode**

## 14.2.15 Pprint\_Addy

**Pprint\_Addy** — `norm_addy` `&#xc9c0;` `&#xc6a9;` `&#xc790;` `&#xc9c0;` `&#xc68;` `&#xc11c;` `&#xc544;` `&#xc574;` `&#xc2f9;` `&#xc1d;` `&#xc4;` `&#xc758;` `&#xc4;` `&#xc30;` `&#xc88b;` `&#xc740;` `&#xc778;` `&#xc1c4;` `&#xc6a9;` `&#xd45c;` `&#xc18;` `&#xd658;` `&#xd569;` `&#xb2c8;` `&#xb2e4;` `&#xc9c0;` `&#xc624;` `&#xc54;` `&#xb529;` `&#xb3c4;` `&#xad6c;` `&#xc640;` `&#xd568;` `&#xd328;` `&#xd0a4;` `&#xc9d5;` `&#xb41c;` `&#xb370;` `&#xc774;` `&#xd130;` `&#xc774;` `&#xc678;` `&#xc5d0;` `&#xb2e4;` `&#xb978;` `&#xb370;` `&#xbd88;` `&#xd544;` `&#xc694;` `&#xd569;` `&#xb2c8;` `&#xb2e4;` `&#xc77c;` `&#xc18;` `&#xc801;` `&#xc73c;` `&#xb85c;` `Normalize_Address` `&#xd568;` `&#xc218;` `&#xc640;` `&#xc6b0;` `&#xd569;` `&#xd574;` `&#xc4f0;` `&#xc785;` `&#xb2c8;` `&#xb2e4;`.

### Synopsis

`varchar pprint_addy(norm_addy in_addy);`

**norm\_addy** `&#xc9c0;` `&#xc6a9;` `&#xc790;` `&#xc9c0;` `&#xc68;` `&#xc11c;` `&#xc544;` `&#xc574;` `&#xc2f9;` `&#xc1d;` `&#xc4;` `&#xc758;` `&#xc4;` `&#xc30;` `&#xc88b;` `&#xc740;` `&#xc778;` `&#xc1c4;` `&#xc6a9;` `&#xd45c;` `&#xc18;` `&#xd658;` `&#xd569;` `&#xb2c8;` `&#xb2e4;` `&#xc9c0;` `&#xc624;` `&#xc54;` `&#xb529;` `&#xb3c4;` `&#xad6c;` `&#xc640;` `&#xd568;` `&#xd328;` `&#xd0a4;` `&#xc9d5;` `&#xb41c;` `&#xb370;` `&#xc774;` `&#xd130;` `&#xc774;` `&#xc678;` `&#xc5d0;` `&#xb2e4;` `&#xb978;` `&#xb370;` `&#xbd88;` `&#xd544;` `&#xc694;` `&#xd569;` `&#xb2c8;` `&#xb2e4;`.

`&#xc77c;` `&#xc18;` `&#xc801;` `&#xc73c;` `&#xb85c;` `Normalize_Address` `&#xd568;` `&#xc218;` `&#xc640;` `&#xc6b0;` `&#xd569;` `&#xd574;` `&#xc4f0;` `&#xc785;` `&#xb2c8;` `&#xb2e4;`.

**pprint\_addy** `&#xc9c0;` `&#xc6a9;` `&#xc790;` `&#xc9c0;` `&#xc68;` `&#xc11c;` `&#xc544;` `&#xc574;` `&#xc2f9;` `&#xc1d;` `&#xc4;` `&#xc758;` `&#xc4;` `&#xc30;` `&#xc88b;` `&#xc740;` `&#xc778;` `&#xc1c4;` `&#xc6a9;` `&#xd45c;` `&#xc18;` `&#xd658;` `&#xd569;` `&#xb2c8;` `&#xb2e4;` `&#xc9c0;` `&#xc624;` `&#xc54;` `&#xb529;` `&#xb3c4;` `&#xad6c;` `&#xc640;` `&#xd568;` `&#xd328;` `&#xd0a4;` `&#xc9d5;` `&#xb41c;` `&#xb370;` `&#xc774;` `&#xd130;` `&#xc774;` `&#xc678;` `&#xc5d0;` `&#xb2e4;` `&#xb978;` `&#xb370;` `&#xbd88;` `&#xd544;` `&#xc694;` `&#xd569;` `&#xb2c8;` `&#xb2e4;`.

`&#xb2e8;` `&#xc77c;` `&#xc8fc;` `&#xc18c;` `&#xc758;` `&#xc4;` `&#xc30;` `&#xc88b;` `&#xc740;` `&#xc778;` `&#xc1c4;` `&#xc6a9;` `&#xd45c;` `&#xc18;` `&#xd658;` `&#xd569;` `&#xb2c8;` `&#xb2e4;` `&#xc9c0;` `&#xc624;` `&#xc54;` `&#xb529;` `&#xb3c4;` `&#xad6c;` `&#xc640;` `&#xd568;` `&#xd328;` `&#xd0a4;` `&#xc9d5;` `&#xb41c;` `&#xb370;` `&#xc774;` `&#xd130;` `&#xc774;` `&#xc678;` `&#xc5d0;` `&#xb2e4;` `&#xb978;` `&#xb370;` `&#xbd88;` `&#xd544;` `&#xc694;` `&#xd569;` `&#xb2c8;` `&#xb2e4;`.

```

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

```

`&#xc8fc;` `&#xc18c;` `&#xd14c;` `&#xc774;` `&#xc14;` `&#xc758;` `&#xc4;` `&#xc30;` `&#xc88b;` `&#xc740;` `&#xc778;` `&#xc1c4;` `&#xc6a9;` `&#xd45c;` `&#xc18;` `&#xd658;` `&#xd569;` `&#xb2c8;` `&#xb2e4;` `&#xc9c0;` `&#xc624;` `&#xc54;` `&#xb529;` `&#xb3c4;` `&#xad6c;` `&#xc640;` `&#xd568;` `&#xd328;` `&#xd0a4;` `&#xc9d5;` `&#xb41c;` `&#xb370;` `&#xc774;` `&#xd130;` `&#xc774;` `&#xc678;` `&#xc5d0;` `&#xb2e4;` `&#xb978;` `&#xb370;` `&#xbd88;` `&#xd544;` `&#xc694;` `&#xd569;` `&#xb2c8;` `&#xb2e4;`.

`&#xc8fc;` `&#xc18c;` `&#xd14c;` `&#xc774;` `&#xc14;` `&#xc758;` `&#xc4;` `&#xc30;` `&#xc88b;` `&#xc740;` `&#xc778;` `&#xc1c4;` `&#xc6a9;` `&#xd45c;` `&#xc18;` `&#xd658;` `&#xd569;` `&#xb2c8;` `&#xb2e4;` `&#xc9c0;` `&#xc624;` `&#xc54;` `&#xb529;` `&#xb3c4;` `&#xad6c;` `&#xc640;` `&#xd568;` `&#xd328;` `&#xd0a4;` `&#xc9d5;` `&#xb41c;` `&#xb370;` `&#xc774;` `&#xd130;` `&#xc774;` `&#xc678;` `&#xc5d0;` `&#xb2e4;` `&#xb978;` `&#xb370;` `&#xbd88;` `&#xd544;` `&#xc694;` `&#xd569;` `&#xb2c8;` `&#xb2e4;`.

```
SELECT address As orig, pprint_addy(normalize_address(address)) As pretty_address
      FROM addresses_to_geocode;
```

| orig  | pretty_address                            |
|---|---|
| 529 Main Street, Boston MA, 02129                   | 529 Main St, Boston MA, 02129             |
| 77 Massachusetts Avenue, Cambridge, MA 02139        | 77 Massachusetts Ave, Cambridge, MA 02139 |
| 28 Capen Street, Medford, MA                        | 28 Capen St, Medford, MA                  |
| 124 Mount Auburn St, Cambridge, Massachusetts 02138 | 124 Mount Auburn St, Cambridge, MA 02138  |
| 950 Main Street, Worcester, MA 01610                | 950 Main St, Worcester, MA 01610          |

**Reverse\_Geocode**

[Normalize\\_Address](#)

### 14.2.16 Reverse\_Geocode

Reverse\_Geocode — Returns the address for a given geometry point. The function returns the address as a text string. The function can be used to find the address for a given geometry point. The function can be used to find the address for a given geometry point. The function can be used to find the address for a given geometry point.

#### Synopsis

record **Reverse\_Geocode**(geometry pt, boolean include\_strnum\_range=false, geometry[] OUT intpt, norm\_addy[] OUT addy, varchar[] OUT street);

**Reverse\_Geocode**

Reverse\_Geocode — Returns the address for a given geometry point. The function returns the address as a text string. The function can be used to find the address for a given geometry point. The function can be used to find the address for a given geometry point. The function can be used to find the address for a given geometry point.

```

&#xd3ec;&#xc778;&#xd2b8;&#xac00; &#xb450; &#xb3c4;&#xb85c;&#xc758; &#xad50;&#xcc28;&#xc810;&#xc5d0; &#xc704;&#
&#xc218;&#xb3c4; &#xc788;&#xc73c;&#xbbc0;&#xb85c; &#xb85c;&#xc9c1;&#xc774; &#xb450; &#xb3c4;&#xb85c; &#xbaa8;&#
&#xb530;&#xb77c; &#bcf4;&#xac04;&#xd558;&#xae30; &#xb54c;&#xbb38;&#xc785;&#xb2c8;&#xb2e4;. &#xc774; &#xb85c;&#
&#xb610; &#xc8fc;&#xc18c;&#xb4e4;&#xc774; &#xb3c4;&#xb85c;&#xb97c; &#xb530;&#xb77c; &#xade0;&#xb4f1;&#xd55c;
&#xac04;&#xaca9;&#xc73c;&#xb85c; &#xc704;&#xce58;&#xd55c;&#xb2e4;&#xace0; &#xac00;&#xc815;&#xd558;&#xb294;&#xl
&#xbb3c;&#xb860; &#xc9c0;&#xc790;&#xccb4; &#xac74;&#xbb3c;&#xc774; &#xb3c4;&#xb85c; &#bc94;&#xc704;&#xc758;
&#xd070; &#xbd80;&#xbd84;&#xc744; &#xcc28;&#xc9c0;&#xd558;&#xace0; &#xb098;&#xba38;&#xc9c0; &#xac74;&#xbb3c;&#
&#xb3c4;&#xb85c; &#xb05d; &#xbd80;&#xbd84;&#xc5d0; &#xbab0;&#xb824; &#xc788;&#xc744; &#xc218;&#xb3c4; &#xc788;&
&#xb54c;&#xbb38;&#xc5d0; &#xc774; &#xac00;&#xc815;&#xc740; &#xd2c0;&#xb838;&#xc2b5;&#xb2c8;&#xb2e4;.

```

```

&#xc8fc;&#xc758;. &#xc774; &#xd568;&#xc218;&#xb294; TIGER &#xb370;&#xc774;&#xd130;&#xc5d0; &#xc758;&#xc874;&#xc
&#xd574;&#xb2f9; &#xd3ec;&#xc778;&#xd2b8;&#xc758; &#xc9c0;&#xc5ed;&#xc744; &#xcee4;&#bc84;&#xd558;&#xb294;
&#xb370;&#xc774;&#xd130;&#xb97c; &#xb85c;&#xb4dc;&#xd558;&#xc9c0; &#xc54a;&#xc558;&#xc744; &#xacbd;&#xc6b0;,
NULL&#xb85c; &#xcc44;&#xc6cc;&#xc9c4; &#xb808;&#xcf54;&#xb4dc;&#xb97c; &#bc18;&#xd658;&#xd560; &#xac83;&#xc7
&#bc18;&#xd658;&#xb41c; &#xb808;&#xcf54;&#xb4dc;&#xb97c; &#xad6c;&#xc131;&#xd558;&#xb294; &#xc694;&#xc18c;&#
&#xb2e4;&#xc74c;&#xacfc; &#xac19;&#xc2b5;&#xb2c8;&#xb2e4;.

```

- ```

1. intpt &#xc740; &#xd3ec;&#xc778;&#xd2b8; &#bc30;&#xc5f4;&#xc785;&#xb2c8;&#xb2e4;. &#xc785;&#xb825;
&#xd3ec;&#xc778;&#xd2b8;&#xc5d0; &#xac00;&#xc7a5; &#xac00;&#xae4c;&#xc6b4; &#b3c4;&#xb85c; &#xc0c1;&#xc75
&#xc911;&#xc2ec;&#xc120; &#xd3ec;&#xc778;&#xd2b8;&#xb4e4;&#xc785;&#xb2c8;&#xb2e4;. &#xc8fc;&#xc18c;&#xc75
&#xac1c;&#xc218; &#xb9cc;&#xd07c; &#xb9ce;&#xc740; &#xd3ec;&#xc778;&#xd2b8;&#xb4e4;&#xc774; &#xc788;&#xc2

```
- ```

2. addy &#xb294; norm_addy(&#xc815;&#xadcc;&#xd654;&#xb41c; &#xc8fc;&#xc18c;) &#bc30;&#xc5f4;&#xc785;&#xb2c8
&#xc785;&#xb825; &#xd3ec;&#xc778;&#xd2b8;&#xc5d0; &#xc801;&#xd569;&#xd55c;. &#xac00;&#xb2a5;&#xd55c;
&#xc8fc;&#xc18c;&#xb4e4;&#xc758; &#bc30;&#xc5f4;&#xc785;&#xb2c8;&#xb2e4;. &#bc30;&#xc5f4;&#xc758;
&#xccab; &#bc88;&#xc9f8; &#xc8fc;&#xc18c;&#xac00; &#xac00;&#xc7a5; &#xc62c;&#bc14;&#xb97c; &#xac00;&#xb2a
&#xd07d;&#xb2c8;&#xb2e4;. &#xc77c;&#bc18;&#xc801;&#xc73c;&#xb85c;. &#xd3ec;&#xc778;&#xd2b8;&#xac00;
&#b3c4;&#xb85c; 2&#xac1c; &#xb610;&#xb294; 3&#xac1c;&#xc758; &#xad50;&#xcc28;&#xc810;&#xc5d0; &#xc704;&#
&#xacbd;&#xc6b0; &#xb610;&#xb294; &#xd3ec;&#xc778;&#xd2b8;&#xac00; &#b3c4;&#xb85c;&#xc758; &#xc5b4;&#xb2
&#xd55c; &#xd3b8;&#xc774; &#xc544;&#xb2c8;&#xb77c; &#b3c4;&#xb85c; &#xc0c1;&#xc5d0; &#xc704;&#xce58;&#xd5
&#xacbd;&#xc6b0;&#xb97c; &#xc81c;&#xc678;&#xd558;&#xba74;. &#xc8fc;&#xc18c; 1&#xac1c;&#xb9cc; &#xc788;&#xc5
&#xd569;&#xb2c8;&#xb2e4;.

```
- ```

3. street &#xb294; varchar&#xd615; &#bc30;&#xc5f4;&#xc785;&#xb2c8;&#xb2e4;. &#xad50;&#xcc28;&#xb85c;&#xc758;
&#b3c4;&#xb85c;&#xb4e4; (&#b610;&#xb294; &#b3c4;&#xb85c; 1&#xac1c;) &#xc785;&#xb2c8;&#xb2e4; (&#xad50;&#
&#b3c4;&#xb85c;&#xb4e4; &#b610;&#xb294; &#xd3ec;&#xc778;&#xd2b8;&#xac00; &#xadf8; &#xc704;&#xc5d0;
&#xd22c;&#xc601;&#xb41c; &#b3c4;&#xb85c;&#xc785;&#xb2c8;&#xb2e4;).

```

Enhanced: 2.4.1 if optional zcta5 dataset is loaded, the reverse\_geocode function can resolve to state and zip even if the specific state data is not loaded. Refer to [Loader\\_Generate\\_Nation\\_Script](#) for details on loading zcta5 data.

```

2.0.0 &#bc84;&#xc804;&#bd80;&#xd130; &#xc0ac;&#xc6a9;&#xd560; &#xc218; &#xc788;&#xc2b5;&#xb2c8;&#xb2e4;.

```

### **&#xc608;&#xc2dc;**

```

&#xd3ec;&#xc778;&#xd2b8;&#xac00; &#xb450; &#xb3c4;&#xb85c;&#xc758; &#xad50;&#xcc28;&#xc810;&#xc5d0; &#xc788;&#
&#xc5b4;&#xb290; &#xd55c; &#b3c4;&#xb85c;&#xc5d0; &#b354; &#xac00;&#xae4c;&#xc6b4; &#xacbd;&#xc6b0;&#xc758;
&#xc608;&#xc2dc;&#xc785;&#xb2c8;&#xb2e4;. &#xc774; &#xd3ec;&#xc778;&#xd2b8;&#xb294; MIT - 77 Massachusetts
Ave, Cambridge, MA 02139 - &#xc5d0; &#xb9e4;&#xc6b0; &#xac00;&#xae4c;&#xc6b4; &#xc704;&#xce58;&#xc5d0; &#xc788;&#
&#b3c4;&#xb85c; 3&#xac1c;&#xc758; &#xacbd;&#xc6b0;&#xb294; &#xc544;&#xb2c8;&#xc9c0;&#xb9cc;. PostgreSQL&#xc774;
&#xc0c1;&#xd55c; (&#x4e0a;&#x9650; upper bound)&#xc744; &#xb118;&#xc5b4;&#xc11c;&#xb294; &#xd56d;&#xbaa9;&#xc5d0
&#xb300;&#xd574; NULL&#xc744; &#bc18;&#xd658;&#xd560; &#xac83;&#xc774;&#xae30; &#b54c;&#xbb38;&#xc5d0;
&#xc548;&#xc804;&#xd558;&#xac8c; &#xc774;&#xc6a9;&#xd560; &#xc218; &#xc788;&#xb2e4;&#xb294; &#xc810;&#xc5d0;
&#xc8fc;&#xc758;&#xd558;&#xc2ed;&#xc2dc;&#xc624;. &#b3c4;&#xb85c; &#bc94;&#xc704;&#b3c4; &#xd3ec;&#xd568;&#

```

```

SELECT pprint_addy(r.addy[1]) As st1, pprint_addy(r.addy[2]) As st2, pprint_addy(r.addy[3]) ←
As st3,
array_to_string(r.street, ',') As cross_streets

```

```
FROM reverse_geocode(ST_GeomFromText('POINT(-71.093902 42.359446)',4269),true) As r
;

result
-----
st1 | st2 | st3 | cross_streets
-----+-----+-----+-----
67 Massachusetts Ave, Cambridge, MA 02139 | | | 67 - 127 Massachusetts Ave,32 - 88
Vassar St
```

&#xb2e4;&#xc74c; &#xc608;&#xc2dc;&#xb294; &#xad50;&#xcc28;&#xb85c;&#xc5d0; &#xb300;&#xd574; &#xc8fc;&#xc18c; &#xbc94;&#xc704;&#xb97c; &#xd3ec;&#xd568;&#xd558;&#xc9c0; &#xc54a;&#xb3c4;&#xb85d; &#xc120;&#xd0dd;&#xd558;&#xb450; &#xb3c4;&#xb85c;&#xc758; &#xad50;&#xcc28;&#xc810;&#xc5d0; &#xc815;&#xb9d0;&#xb85c; &#xac00;&#xae4c;&#xc704;&#xce58;&#xb97c; &#xace8;&#xb77c;&#xc11c; &#xc11c;&#xb85c; &#xb2e4;&#xb978; &#xb450; &#xc8fc;&#xc18c;&#xbc18;&#xd658;&#xd560; &#xc218; &#xc788;&#xb3c4;&#xb85d; &#xd569;&#xb2c8;&#xb2e4;.

```
SELECT pprint_addy(r.addy[1]) As st1, pprint_addy(r.addy[2]) As st2,
pprint_addy(r.addy[3]) As st3, array_to_string(r.street, ',') As cross_str
FROM reverse_geocode(ST_GeomFromText('POINT(-71.06941 42.34225)',4269)) As r;

result
-----
st1 | st2 | st3 | cross_str
-----+-----+-----+-----
5 Bradford St, Boston, MA 02118 | 49 Waltham St, Boston, MA 02118 | | Waltham St
```

&#xb2e4;&#xc74c; &#xc608;&#xc2dc;&#xb294; **Geocode** &#xc758; &#xc9c0;&#xc624;&#xcf54;&#xb529;&#xb41c; &#xc608;&#xb2e4;&#xc2dc; &#xc774;&#xc6a9;&#xd574;&#xc11c; &#xb3c4;&#xb85c;&#xac00; &#xcd5c;&#xb300;&#xd55c; 2&#xac1c;&#xad50;&#xcc28;&#xd558;&#xb294; &#xad50;&#xcc28;&#xb85c;&#xc5d0; &#xc788;&#xb294; &#xcd5c;&#xc120;&#xc758; &#xc8fc;&#xc18c;&#xb9cc;&#xc744; &#xbc18;&#xd658;&#xd558;&#xb3c4;&#xb85d; &#xd569;&#xb2c8;&#xb2e4;.

```
SELECT actual_addr, lon, lat, pprint_addy((rg).addy[1]) As int_addr1,
(rg).street[1] As cross1, (rg).street[2] As cross2
FROM (SELECT address As actual_addr, lon, lat,
reverse_geocode( ST_SetSRID(ST_Point(lon,lat),4326) ) As rg
FROM addresses_to_geocode WHERE rating
> -1) As foo;

actual_addr | lon | lat | cross1 |
-----+-----+-----+-----+
529 Main Street, Boston MA, 02129 | -71.07181 | 42.38359 | 527 Main St,
Boston, MA 02129 | Medford St |
77 Massachusetts Avenue, Cambridge, MA 02139 | -71.09428 | 42.35988 | 77
Massachusetts Ave, Cambridge, MA 02139 | Vassar St |
26 Capen Street, Medford, MA | -71.12377 | 42.41101 | 9 Edison Ave,
Medford, MA 02155 | Capen St | Tesla Ave
124 Mount Auburn St, Cambridge, Massachusetts 02138 | -71.12304 | 42.37328 | 3 University
Rd, Cambridge, MA 02138 | Mount Auburn St |
950 Main Street, Worcester, MA 01610 | -71.82368 | 42.24956 | 3 Maywood St,
Worcester, MA 01603 | Main St | Maywood Pl
```

&#xcc38;&#xace0;

**PPrint\_Addy, Pprint\_Addy, ST\_AsText**

## 14.2.17 Topology\_Load\_Tiger

Topology\_Load\_Tiger — PostGIS TIGER Topology Loader. Loads TIGER data into a topology. The topology is created from the TIGER data. The topology is created from the TIGER data. The topology is created from the TIGER data.

### Synopsis

text Topology\_Load\_Tiger(varchar topo\_name, varchar region\_type, varchar region\_id);

### Options

PostGIS TIGER Topology Loader. Loads TIGER data into a topology. The topology is created from the TIGER data. The topology is created from the TIGER data. The topology is created from the TIGER data.

PostGIS TIGER Topology Loader. Loads TIGER data into a topology. The topology is created from the TIGER data. The topology is created from the TIGER data. The topology is created from the TIGER data.

#### Note



PostGIS TIGER Topology Loader. Loads TIGER data into a topology. The topology is created from the TIGER data. The topology is created from the TIGER data. The topology is created from the TIGER data.

#### Note



PostGIS TIGER Topology Loader. Loads TIGER data into a topology. The topology is created from the TIGER data. The topology is created from the TIGER data. The topology is created from the TIGER data.

```
&#xd544;&#xc218; &#xc778;&#xc218;:
```

1. `topo_name` - `&#xb370;&#xc774;&#xd130;&#xb97c; &#xb85c;&#xb4dc;&#xd560; &#xae30;&#xc874; PostGIS &#xc9c0;&#xba85;&#xce6d;&#xc785;&#xb2c8;&#xb2e4;`.
2. `region_type` - `&#xacbd;&#xacc4;&#xb97c; &#xc774;&#xb8e8;&#xb294; &#xc9c0;&#xc5ed;&#xc758; &#xc720;&#xd615;&#xd604;&#xc7ac; place &#xc640; county &#xb9cc; &#xc9c0;&#xc6d0;&#xd569;&#xb2c8;&#xb2e4; . &#xba87;&#xba87;&#xc720;&#xd615;&#xc744; &#xb354; &#xc9c0;&#xc6d0;&#xd560; &#xc608;&#xc815;&#xc785;&#xb2c8;&#xb2e4; . &#xc774; &#xc778;&#xc218;&#xb294; tiger.place, tiger.county &#xc98;&#xb7fc; &#xc9c0;&#xc5ed; &#xacbd;&#xc815;&#xc758;&#xd558;&#xb824;&#xba74; &#xc0b4;&#xd3b4;&#xbd10;&#xc57c; &#xd560; &#xd14c;&#xc774;&#xb2e4;`.
3. `region_id` - `TIGER &#xac00; &#xc9c0;&#xb9ac; ID(geoid)&#xb77c;&#xace0; &#xbd80;&#xb974;&#xb294; &#xc2dd;&#xd14c;&#xc774;&#xbel4;&#xc5d0; &#xc788;&#xb294; &#xc9c0;&#xc5ed;&#xc758; &#xc720;&#xc77c;&#xd55c; &#xc2dd;&#xbcc4;&#xc790;&#xc785;&#xb2c8;&#xb2e4; . place &#xc758; &#xacbd;&#xc6b0; tiger.place &#xd14c;&#xc788;&#xb294; plcidfp &#xc5f4;&#xc785;&#xb2c8;&#xb2e4; . county &#xc758; &#xacbd;&#xc6b0; tiger.county &#xd14c;&#xc774;&#xbel4;&#xc5d0; &#xc788;&#xb294; cntyidfp &#xc5f4;&#xc785;&#xb2c8;&#xb2e4;`.

```
2.0.0 &#xbc84;&#xc804;&#xbd80;&#xd130; &#xc0ac;&#xc6a9;&#xd560; &#xc218; &#xc788;&#xc2b5;&#xb2c8;&#xb2e4;
```

```
&#xc608;&#xc2dc;: &#xb9e4;&#xc0ac;&#xcd94;&#xc138;&#xce20; &#xc8fc; &#xbcf4;&#xc2a4;&#xd134; &#xc2dc;&#xc758; &#xc9c0;&#xd615;
```

```
&#xb9e4;&#xc0ac;&#xcd94;&#xc138;&#xce20; &#xc8fc; &#xd53c;&#xd2b8; &#xb2e8;&#xc704; &#xd3c9;&#xba74;(2249)&#xc5c8;&#xb9e4;&#xc0ac;&#xcd94;&#xc138;&#xce20; &#xc8fc; &#xbcf4;&#xc2a4;&#xd134; &#xc2dc;&#xc5d0; &#xb300;&#xd574; &#xd5c8;&#xc6a9; &#xc624;&#xc28; 0.25 &#xd53c;&#xd2b8;&#xb97c; &#xac00;&#xc9c4; &#xc9c0;&#xd615;&#xc744; &#xc0dd;&#xc131;&#xd55c; &#xb2e4;&#xc74c; , &#xbcf4;&#xc2a4;&#xd134; &#xc2dc;&#xc758; TIGER &#xba74; , &#xacbd;&#xb178;&#xb4dc;&#xb97c; &#xb85c;&#xb4dc;&#xd569;&#xb2c8;&#xb2e4;
```

```
SELECT topology.CreateTopology('topo_boston', 2249, 0.25);
createtopology
-----
15
-- &#xc708;&#xb3c4;&#xc6b0;7 &#xb370;&#xc2a4;&#xd06c;&#xd0d1;&#xc5d0;&#xc11c; 9.1 ↔
&#xbc84;&#xc804;&#xc744; (5&#xac1c; &#xc8fc;&#xc758; TIGER ↔
&#xb370;&#xc774;&#xd130;&#xb97c; &#xb85c;&#xb4dc;&#xd55c; &#xc0c1;&#xd0dc;&#xb85c;) ↔
&#xc2e4;&#xd589;&#xd588;&#xc744; &#xb54c; 60,902&#xbc00;&#xb9ac;&#xcd08; ~ 1&#xbd84; ↔
&#xc18c;&#xc694;
SELECT tiger.topology_load_tiger('topo_boston', 'place', '2507000');
-- topology_loader_tiger --
29722 edges holding in temporary. 11108 faces added. 1875 edges of faces added. 20576 ↔
nodes added.
19962 nodes contained in a face. 0 edge start end corrected. 31597 edges added.

-- 41&#xbc00;&#xb9ac;&#xcd08; &#xc18c;&#xc694; --
SELECT topology.TopologySummary('topo_boston');
-- topologysummary--
Topology topo_boston (15), SRID 2249, precision 0.25
20576 nodes, 31597 edges, 11109 faces, 0 topogeoms in 0 layers

-- &#xd655;&#xc778; &#xc791;&#xc5c5;&#xc5d0; 28,797&#xbc00;&#xb9ac;&#xcd08; ↔
&#xc18c;&#xc694; , &#xbclc;&#xc0dd;&#xd55c; &#xc624;&#xb958; &#xc5c6;&#xc74c; --
SELECT * FROM
topology.ValidateTopology('topo_boston');

error | id1 | id2
-----+-----+-----
```



**&#xc124;&#xba85;**

tiger.geocode\_settings &#xd14c;&#xc774;&#xbe14;&#xc5d0; &#xc800;&#xc7a5;&#xb41c; &#xd2b9;&#xc815; &#xc124; &#xac12;&#xc744; &#xc124;&#xc815;&#xd569;&#xb2c8;&#xb2e4;. &#xc0ac;&#xc6a9;&#xc790;&#xac00; &#xd568;&#xc218;&#xb514;&#xbc84;&#xae45;&#xc744; &#xcf1c;&#xace0; &#xb04c; &#xc218; &#xc788;&#xac8c; &#xd574;&#xc8fc;&#xb294; &#xc124;&#xc815;&#xac12;&#xc785;&#xb2c8;&#xb2e4;. &#xd5a5;&#xd6c4; &#xc124;&#xc815;&#xac12;&#xc73c;&#xb85c; &#xc21c;&#xc704;&#xb97c; &#xc81c;&#xc5b4;&#xd560; &#xc218; &#xc788;&#xb3c4;&#xb85d; &#xd560; &#xacc4;&#xd68d;&#xc744; &#xc815;&#xc11c; &#xd604;&#xc7ac; &#xc124;&#xc815;&#xac12; &#xbaa9;&#xb85d;&#xc744; &#xbcf; &#xc218; &#xc788;&#xc2b5;&#xb2c8;&#xb2e4;.

2.1.0 &#xbc84;&#xc804;&#xbd80;&#xd130; &#xc0ac;&#xc6a9;&#xd560; &#xc218; &#xc788;&#xc2b5;&#xb2c8;&#xb2e4;.

**&#xc608;&#xc2dc;: &#xb514;&#xbc84;&#xae45; &#xc124;&#xc815;&#xac12; &#xbc18;&#xd658;**

&#xc774; &#xd568;&#xc218;&#xac00; &#xcc38;&#xc77c; &#xb54c; **Geocode** &#xb97c; &#xc2e4;&#xd589;&#xd560; &#xacbd;&#xb85c;&#xadf8;&#xac00; &#xc18c;&#xc694; &#xc2dc;&#xac04; &#xbc0f; &#xcffc;&#xb9ac;&#xb97c; &#xcd9c;&#xb85c;&#xc785;&#xb2c8;&#xb2e4;.

```
SELECT set_geocode_setting('debug_geocode_address', 'true') As result;
result
-----
true
```

**&#xcc38;&#xace0;**

**Get\_Geocode\_Setting**

## Chapter 15

# PostGIS Special Functions Index

### 15.1 PostGIS Aggregate Functions

The functions given below are spatial aggregate functions provided with PostGIS that can be used just like any other sql aggregate function such as sum, average.

- **ST\_3DExtent** - Aggregate function that returns the 3D bounding box of geometries.
- **ST\_3DUnion** - Perform 3D union.
- **ST\_AsFlatGeobuf** - Return a FlatGeobuf representation of a set of rows.
- **ST\_AsGeobuf** - Return a Geobuf representation of a set of rows.
- **ST\_AsMVT** - Aggregate function returning a MVT representation of a set of rows.
- **ST\_ClusterIntersecting** - Aggregate function that clusters the input geometries into connected sets.
- **ST\_ClusterWithin** - Aggregate function that clusters the input geometries by separation distance.
- **ST\_Extent** - Aggregate function that returns the bounding box of geometries.
- **ST\_GeomCollFromText** - Creates a GeometryCollection or Multi\* geometry from a set of geometries.
- **ST\_MakeLine** -
- **ST\_MemUnion** - Aggregate function which unions geometries in a memory-efficient but slower way
- **ST\_Polygonize** - Computes a collection of polygons formed from the linework of a set of geometries.
- **ST\_SameAlignment** -
- **ST\_Union** - Computes a geometry representing the point-set union of the input geometries.
- **TopoElementArray\_Agg** - Returns a topoelementarray for a set of element\_id, type arrays (topoelements).

## 15.2 PostGIS Window Functions

The functions given below are spatial window functions provided with PostGIS that can be used just like any other sql window function such as `row_number()`, `lead()`, `lag()`. All these require an SQL `OVER()` clause.

- **ST\_ClusterDBSCAN** - Window function that returns a cluster id for each input geometry using the DBSCAN algorithm.
- **ST\_ClusterKMeans** - Window function that returns a cluster id for each input geometry using the K-means algorithm.

## 15.3 PostGIS SQL-MM Compliant Functions

The functions given below are PostGIS functions that conform to the SQL/MM 3 standard

- **ST\_3DArea** - 3D area of a geometry. This method implements the SQL/MM specification. SQL-MM IEC 13249-3: 8.1, 10.5
- **ST\_3DDWithin** - Tests if two 3D geometries are within a given 3D distance. This method implements the SQL/MM specification. SQL-MM ?
- **ST\_3DDifference** - 3D difference of two geometries. This method implements the SQL/MM specification. SQL-MM IEC 13249-3: 5.1
- **ST\_3DDistance** - 3D distance between two geometries. This method implements the SQL/MM specification. SQL-MM ISO/IEC 13249-3
- **ST\_3DIntersection** - 3D intersection of two geometries. This method implements the SQL/MM specification. SQL-MM IEC 13249-3: 5.1
- **ST\_3DIntersects** - Tests if two geometries spatially intersect in 3D - only for points, linestrings, polygons, polyhedral surface (area). This method implements the SQL/MM specification. SQL-MM IEC 13249-3: 5.1
- **ST\_3DLength** - 3D length of a geometry. This method implements the SQL/MM specification. SQL-MM IEC 13249-3: 7.1, 10.3
- **ST\_3DPerimeter** - 3D perimeter of a geometry. This method implements the SQL/MM specification. SQL-MM ISO/IEC 13249-3: 8.1, 10.5
- **ST\_3DUnion** - Perform 3D union. This method implements the SQL/MM specification. SQL-MM IEC 13249-3: 5.1
- **ST\_AddEdgeModFace** - Add a new edge to a face. This method implements the SQL/MM specification. SQL-MM: Topo-Geo and Topo-Net 3: Routine Details: X.3.13
- **ST\_AddEdgeNewFaces** - Add a new edge to a face, creating new faces. This method implements the SQL/MM specification. SQL-MM: Topo-Geo and Topo-Net 3: Routine Details: X.3.12
- **ST\_AddIsoEdge** - Add an isolated edge to a face. This method implements the SQL/MM specification. SQL-MM: Topo-Geo and Topo-Net 3: Routine Details: X.3.4

- **ST\_AddIsoNode** - (isolated) ID; NULL; . This method implements the SQL/MM specification. SQL-MM: Topo-Net Routines: X+1.3.1
- **ST\_Area** - This method implements the SQL/MM specification. SQL-MM 3: 8.1.2, 9.5.3
- **ST\_AsBinary** - Return the OGC/ISO Well-Known Binary (WKB) representation of the geometry/geography without SRID meta data. This method implements the SQL/MM specification. SQL-MM 3: 5.1.37
- **ST\_AsGML** - GML 2; GML 3; . This method implements the SQL/MM specification. SQL-MM IEC 13249-3: 17.2
- **ST\_AsText** - WKT(Well-Known Text) SRID; . This method implements the SQL/MM specification. SQL-MM 3: 5.1.25
- **ST\_Boundary** - This method implements the SQL/MM specification. SQL-MM IEC 13249-3: 5.1.17
- **ST\_Buffer** - Computes a geometry covering all points within a given distance from a geometry. This method implements the SQL/MM specification. SQL-MM IEC 13249-3: 5.1.30
- **ST\_Centroid** - This method implements the SQL/MM specification. SQL-MM 3: 8.1.4, 9.5.5
- **ST\_ChangeEdgeGeom** - This method implements the SQL/MM specification. SQL-MM: Topo-Geo and Topo-Net 3: Routine Details X.3.6
- **ST\_Contains** - Tests if no points of B lie in the exterior of A, and A and B have at least one interior point in common. This method implements the SQL/MM specification. SQL-MM 3: 5.1.31
- **ST\_ConvexHull** - Computes the convex hull of a geometry. This method implements the SQL/MM specification. SQL-MM IEC 13249-3: 5.1.16
- **ST\_CoordDim** - ST\_Geometry; This method implements the SQL/MM specification. SQL-MM 3: 5.1.3
- **ST\_CreateTopoGeo** - This method implements the SQL/MM specification. SQL-MM: Topo-Geo and Topo-Net 3: Routine Details -- X.3.18
- **ST\_Crosses** - Tests if two geometries have some, but not all, interior points in common. This method implements the SQL/MM specification. SQL-MM 3: 5.1.29
- **ST\_CurveToLine** - Converts a geometry containing curves to a linear geometry. This method implements the SQL/MM specification. SQL-MM 3: 7.1.7
- **ST\_Difference** - Computes a geometry representing the part of geometry A that does not intersect geometry B. This method implements the SQL/MM specification. SQL-MM 3: 5.1.20
- **ST\_Dimension** - ST\_Geometry; This method implements the SQL/MM specification. SQL-MM 3: 5.1.2
- **ST\_Disjoint** - Tests if two geometries are disjoint (they have no point in common). This method implements the SQL/MM specification. SQL-MM 3: 5.1.26

- **ST\_Distance** - Returns the distance between two geometries in SRID units. (longest edge of bounding box). This method implements the SQL/MM specification. SQL-MM 3: 5.1.23
- **ST\_EndPoint** - Returns the endpoint of a line string. This method implements the SQL/MM specification. SQL-MM 3: 7.1.4
- **ST\_Envelope** - Returns the envelope (bounding box) of the supplied geometry. This method implements the SQL/MM specification. SQL-MM 3: 5.1.19
- **ST\_Equals** - Tests if two geometries include the same set of points. This method implements the SQL/MM specification. SQL-MM 3: 5.1.24
- **ST\_ExteriorRing** - Returns the exterior ring of a polygon. This method implements the SQL/MM specification. SQL-MM 3: 8.2.3, 8.3.3
- **ST\_GMLToSQL** - Converts a GML geometry to a SQL geometry. This method implements the SQL/MM specification. SQL-MM 3: 5.1.50
- **ST\_GeomCollFromText** - Makes a collection Geometry from collection WKT with the given SRID. If SRID is not given, it defaults to 0. This method implements the SQL/MM specification.
- **ST\_GeomFromText** - WKT to SQL geometry. This method implements the SQL/MM specification. SQL-MM 3: 5.1.40
- **ST\_GeomFromWKB** - WKB (Well-Known Binary) to SQL geometry. This method implements the SQL/MM specification. SQL-MM 3: 5.1.41
- **ST\_GeometryFromText** - WKT (Well-Known Text) to SQL geometry. This method implements the SQL/MM specification. SQL-MM 3: 5.1.40
- **ST\_GeometryN** - Returns the Nth geometry of a multi-geometry. This method implements the SQL/MM specification. SQL-MM 3: 9.1.5
- **ST\_GeometryType** - Returns the geometry type of a geometry. This method implements the SQL/MM specification. SQL-MM 3: 5.1.4
- **ST\_GetFaceEdges** - Returns the edges of a face. This method implements the SQL/MM specification. SQL-MM 3 Topo-Geo and Topo-Net 3: Routine Details: X.3.5
- **ST\_GetFaceGeometry** - Returns the geometry of a face. This method implements the SQL/MM specification. SQL-MM 3 Topo-Geo and Topo-Net 3: Routine Details: X.3.16
- **ST\_InitTopoGeo** - Initializes a TopoGeo object. This method implements the SQL/MM specification. SQL-MM 3 Topo-Geo and Topo-Net 3: Routine Details: X.3.17
- **ST\_InteriorRingN** - Returns the Nth interior ring of a polygon. This method implements the SQL/MM specification. SQL-MM 3: 8.2.6, 8.3.5

- **ST\_Intersection** - Computes a geometry representing the shared portion of geometries A and B. This method implements the SQL/MM specification. SQL-MM 3: 5.1.18
- **ST\_Intersects** - Tests if two geometries intersect (they have at least one point in common). This method implements the SQL/MM specification. SQL-MM 3: 5.1.27
- **ST\_IsClosed** - LINESTRING &#xc758; &#xc2dc;&#xc791;&#xc810;&#xcacf; &#xc885;&#xb2e8;&#xc810;&#xc774; &#xc77c;&#xcabd;&#xc6b0; TRUE &#xb97c; &#xbc18;&#xd658;&#xd569;&#xb2c8;&#xb2e4;. &#xb2e4;&#xba74;&#xc6b4; &#xd45c;&#xcabd;&#xc6b0; (&#bd80;&#xd53c;&#xb97c; &#xac00;&#xc9c0;&#xace0;) &#xc788;&#xb294; &#xcabd;&#xc6b0; TRUE &#xb97c; &#xbc18;&#xd658;&#xd569;&#xb2c8;&#xb2e4;. This method implements the SQL/MM specification. SQL-MM 3: 7.1.5, 9.3.3
- **ST\_IsEmpty** - Tests if a geometry is empty. This method implements the SQL/MM specification. SQL-MM 3: 5.1.7
- **ST\_IsRing** - Tests if a LineString is closed and simple. This method implements the SQL/MM specification. SQL-MM 3: 7.1.6
- **ST\_IsSimple** - &#xd574;&#xb2f9; &#xb3c4;&#xd615;&#xc774; &#xc790;&#xc6b4; &#xad50;&#xcc28;&#xd558;&#xc70;&#xb0 &#xc790;&#xc6b4; &#xc811;&#xcd09;&#xd558;&#xb294; &#xc774;&#xb840;&#xc801;&#xc778; &#xb3c4;&#xd615; &#xd3ec;&#xcabd;&#xc6b0; (&#bd80;&#xd53c;&#xb97c; &#xac00;&#xc9c0;&#xace0; &#xc788;&#xc9c0; &#xc54a;&#xc744; &#xcabd;&#xc6b0; TRUE &#xb97c; &#xbc18;&#xd658;&#xcabd;&#xc6b0;. This method implements the SQL/MM specification. SQL-MM 3: 5.1.8
- **ST\_IsValid** - Tests if a geometry is well-formed in 2D. This method implements the SQL/MM specification. SQL-MM 3: 5.1.9
- **ST\_Length** - &#xb3c4;&#xd615;&#xc758; &#xae30;&#xd558;&#xd559;&#xc801; &#xc911;&#xc2ec;&#xc744; &#xbc18;&#xd658; This method implements the SQL/MM specification. SQL-MM 3: 7.1.2, 9.3.4
- **ST\_LineFromText** - &#xc8fc;&#xc5b4;&#xc9c4; SRID&#xc640; &#xd568;&#xaed8; WKT &#xd45c;&#xd604;&#xc2dd;&#xc73c; &#xb3c4;&#xd615;&#xc744; &#xb9cc;&#xb4ed;&#xb2c8;&#xb2e4;. SRID&#xc640; &#xc8fc;&#xc5b4;&#xc9c0;&#xc9c0; &#xc54a;&#xc740; &#xcabd;&#xc6b0;., &#xae30;&#xbcf8;&#xc12;&#xc778; 0&#xc744; &#xc501;&#xb2c8;&#xb2e4;. This method implements the SQL/MM specification. SQL-MM 3: 7.2.8
- **ST\_LineFromWKB** - &#xc8fc;&#xc5b4;&#xc9c4; SRID&#xc640; &#xd568;&#xaed8; WKB&#xb85c;&#xbd80;&#xd130; LINESTRING &#xc744; &#xb9cc;&#xb4ed;&#xb2c8;&#xb2e4;. This method implements the SQL/MM specification. SQL-MM 3: 7.2.9
- **ST\_LinestringFromWKB** - &#xc8fc;&#xc5b4;&#xc9c4; SRID&#xc640; &#xd568;&#xaed8; WKB&#xb85c;&#xbd80;&#xd130; &#xb3c4;&#xd615;&#xc744; &#xb9cc;&#xb4ed;&#xb2c8;&#xb2e4;. This method implements the SQL/MM specification. SQL-MM 3: 7.2.9
- **ST\_LocateAlong** - Returns the point(s) on a geometry that match a measure value. This method implements the SQL/MM specification. SQL-MM IEC 13249-3: 5.1.13
- **ST\_LocateBetween** - Returns the portions of a geometry that match a measure range. This method implements the SQL/MM specification. SQL-MM IEC 13249-3: 5.1
- **ST\_M** - Returns the M coordinate of a Point. This method implements the SQL/MM specification.
- **ST\_MLineFromText** - WKT &#xd45c;&#xd604;&#xc2dd;&#xc73c;&#xb85c;&#xbd80;&#xd130; &#xc9c0;&#xc815;&#xb41c; ST\_MultiLineString &#xc12;&#xc744; &#xbc18;&#xd658;&#xd569;&#xb2c8;&#xb2e4;. This method implements the SQL/MM specification. SQL-MM 3: 9.4.4
- **ST\_MPointFromText** - Makes a Geometry from WKT with the given SRID. If SRID is not given, it defaults to 0. This method implements the SQL/MM specification. SQL-MM 3: 9.2.4
- **ST\_MPolyFromText** - Makes a MultiPolygon Geometry from WKT with the given SRID. If SRID is not given, it defaults to 0. This method implements the SQL/MM specification. SQL-MM 3: 9.6.4
- **ST\_ModEdgeHeal** - Heals two edges by deleting the node connecting them, modifying the first edge and deleting the second edge. Returns the id of the deleted node. This method implements the SQL/MM specification. SQL-MM: Topo-Geo and Topo-Net 3: Routine Details: X.3.9

- **ST\_ModEdgeSplit** - Moves an edge in a topology from one point to another. If new apoint geometry exists as a node an error is thrown. Returns description of move. This method implements the SQL/MM specification. SQL-MM: Topo-Geo and Topo-Net 3: Routine Details: X.3.9
- **ST\_MoveIsoNode** - Moves an isolated node in a topology from one point to another. If new apoint geometry exists as a node an error is thrown. Returns description of move. This method implements the SQL/MM specification. SQL-MM: Topo-Net Routines: X.3.2
- **ST\_NewEdgeHeal** - Heals two edges by deleting the node connecting them, deleting both edges, and replacing them with an edge whose direction is the same as the first edge provided. This method implements the SQL/MM specification. SQL-MM: Topo-Geo and Topo-Net 3: Routine Details: X.3.9
- **ST\_NewEdgesSplit** - Moves an edge in a topology from one point to another. If new apoint geometry exists as a node an error is thrown. Returns description of move. This method implements the SQL/MM specification. SQL-MM: Topo-Net Routines: X.3.8
- **ST\_NumGeometries** - Returns the number of geometries in a collection. This method implements the SQL/MM specification. SQL-MM 3: 9.1.4
- **ST\_NumInteriorRings** - Returns the number of interior rings in a polygon. This method implements the SQL/MM specification. SQL-MM 3: 8.2.5
- **ST\_NumPatches** - Returns the number of patches in a geometry. This method implements the SQL/MM specification. SQL-MM ISO/IEC 13249-3: 8.5
- **ST\_NumPoints** - Returns the number of points in a geometry. This method implements the SQL/MM specification. SQL-MM 3: 7.2.4
- **ST\_OrderingEquals** - Tests if two geometries represent the same geometry and have points in the same directional order. This method implements the SQL/MM specification. SQL-MM 3: 5.1.43
- **ST\_Overlaps** - Tests if two geometries intersect and have the same dimension, but are not completely contained by each other. This method implements the SQL/MM specification. SQL-MM 3: 5.1.32
- **ST\_PatchN** - Returns the Nth patch in a geometry. This method implements the SQL/MM specification. SQL-MM ISO/IEC 13249-3: 8.5
- **ST\_Perimeter** - Returns the length of the boundary of a polygonal geometry or geography. This method implements the SQL/MM specification. SQL-MM 3: 8.1.3, 9.5.4
- **ST\_Point** - Creates a Point with X, Y and SRID values. This method implements the SQL/MM specification. SQL-MM 3: 6.1.2
- **ST\_PointFromText** - Returns a Point from a text representation. This method implements the SQL/MM specification. SQL-MM 3: 6.1.8
- **ST\_PointFromWKB** - Returns a Point from a WKB representation. This method implements the SQL/MM specification. SQL-MM 3: 6.1.9

- **ST\_PointN** - ST\_LineString &#xb610;&#xb294; ST\_CircularString &#xac12;&#xc5d0; &#xc788;&#xb294; &#xd3ec;&#xc778;&#xcac1c;&#xc218;&#xb97c; &#xbc18;&#xd658;&#xd569;&#xb2c8;&#xb2e4;. This method implements the SQL/MM specification. SQL-MM 3: 7.2.5, 7.3.5
- **ST\_PointOnSurface** - Computes a point guaranteed to lie in a polygon, or on a geometry. This method implements the SQL/MM specification. SQL-MM 3: 8.1.5, 9.5.6. The specifications define ST\_PointOnSurface for surface geometries only. PostGIS extends the function to support all common geometry types. Other databases (Oracle, DB2, ArcSDE) seem to support this function only for surfaces. SQL Server 2008 supports all common geometry types.
- **ST\_Polygon** - Creates a Polygon from a LineString with a specified SRID. This method implements the SQL/MM specification. SQL-MM 3: 8.3.2
- **ST\_PolygonFromText** - Makes a Geometry from WKT with the given SRID. If SRID is not given, it defaults to 0. This method implements the SQL/MM specification. SQL-MM 3: 8.3.6
- **ST\_Relate** - Tests if two geometries have a topological relationship matching an Intersection Matrix pattern, or computes their Intersection Matrix This method implements the SQL/MM specification. SQL-MM 3: 5.1.25
- **ST\_RemEdgeModFace** - &#xacbd;&xacc4;&#xc120;&#xc744; &#xc81c;&xac70;&#xd558;&xace0;, &#xc81c;&xac70;&#xd558;&xace0;&#xacbd;&xacc4;&#xc120;&#xc774; &#xb450; &#xd45c;&#xba74;&#xc744; &#xbd84;&#xd560;&#xd558;&xace0; &#xc788;&#xacbd;&#xc6b0;, &#xd45c;&#xba74; &#xd558;&#xb098;&#xb97c; &#xc0ad;&#xc81c;&#xd558;&xace0; &#xb2e4;&#xb978; &#xd45c;&#xba74;&#xc744; &#xb450; &#xd45c;&#xba74;&#xc758; &#xacf5;&xac04;&#xc744; &#xcc28;&#xc9c0;&#xd558;&#xc218;&#xc815;&#xd569;&#xb2c8;&#xb2e4;. This method implements the SQL/MM specification. SQL-MM: Topo-Geo and Topo-Net 3: Routine Details: X.3.15
- **ST\_RemEdgeNewFace** - &#xacbd;&xacc4;&#xc120;&#xc744; &#xc81c;&xac70;&#xd558;&xace0;, &#xc81c;&xac70;&#xd558;&xace0;&#xacbd;&xacc4;&#xc120;&#xc774; &#xb450; &#xd45c;&#xba74;&#xc744; &#xbd84;&#xd560;&#xd558;&xace0; &#xc788;&#xacbd;&#xc6b0;, &#xc6d0;&#xbcf8; &#xd45c;&#xba74;&#xb4e4;&#xc744; &#xc0ad;&#xc81c;&#xd558;&xace0; &#xc0c8; &#xd45c;&#xba74; &#xd558;&#xb098;&#xb85c; &#xb300;&#xcceb4;&#xd569;&#xb2c8;&#xb2e4;. This method implements the SQL/MM specification. SQL-MM: Topo-Geo and Topo-Net 3: Routine Details: X.3.14
- **ST\_RemoveIsoEdge** - Removes an isolated edge and returns description of action. If the edge is not isolated, then an exception is thrown. This method implements the SQL/MM specification. SQL-MM: Topo-Geo and Topo-Net 3: Routine Details: X+1.3.3
- **ST\_RemoveIsoNode** - &#xace0;&#xb9bd;&#xb41c; &#xb178;&#xb4dc;&#xb97c; &#xc81c;&xac70;&#xd558;&xace0; &#xc791;&#xc5c5; &#xb0b4;&#xc6a9;&#xc744; &#xc124;&#xba85;&#xd558;&#xb294; &#xba54;&#xc2dc;&#xc9c0;&#xb97c; &#xbc18;&#xd658;&#xd569;&#xb2c8;&#xb2e4;. &#xb178;&#xb4dc;&xac00; &xace0;&#xb9bd;&#xb418;&#xc9c0; &#xc54a; (&xacbd;&xacc4;&#xc120;&#xc758; &#xc2dc;&#xc791;&#xc810;&#xc774;&#xb098; &#xc885;&#xb2e8;&#xc810;&#xc778;) &#xacbd;&#xc6b0;, &#xc608;&#xc678;&xac00; &#xbc1c;&#xc0dd;&#xd569;&#xb2c8;&#xb2e4;. This method implements the SQL/MM specification. SQL-MM: Topo-Geo and Topo-Net 3: Routine Details: X+1.3.3
- **ST\_SRID** - Returns the spatial reference identifier for a geometry. This method implements the SQL/MM specification. SQL-MM 3: 5.1.5
- **ST\_StartPoint** - Returns the first point of a LineString. This method implements the SQL/MM specification. SQL-MM 3: 7.1.3
- **ST\_SymDifference** - Computes a geometry representing the portions of geometries A and B that do not intersect. This method implements the SQL/MM specification. SQL-MM 3: 5.1.21
- **ST\_Touches** - Tests if two geometries have at least one point in common, but their interiors do not intersect. This method implements the SQL/MM specification. SQL-MM 3: 5.1.28
- **ST\_Transform** - Return a new geometry with coordinates transformed to a different spatial reference system. This method implements the SQL/MM specification. SQL-MM 3: 5.1.6
- **ST\_Union** - Computes a geometry representing the point-set union of the input geometries. This method implements the SQL/MM specification. SQL-MM 3: 5.1.19 the z-index (elevation) when polygons are involved.
- **ST\_Volume** - 3&#xcc28;&#xc6d0; &#xc785;&xccb4;&#xc758; &#xbd80;&#xd53c;&#xb97c; &xacc4;&#xc0b0;&#xd569;&#xb2 &#xd45c;&#xba74; &#xb3c4;&#xd615;&#xc744; &#xc785;&#xb825;&#xd558;&#xba74; (&#xb2eb;&#xd78c; &#xb3c4;&#xd615 0&#xc744; &#xbc18;&#xd658;&#xd560; &xac83;&#xc785;&#xb2c8;&#xb2e4;. This method implements the SQL/MM specification. SQL-MM IEC 13249-3: 9.1 (same as ST\_3DVolume)

- **ST\_WKBToSQL** - WKB(Well-Known Binary) `ST_Geometry SRID ST_GeomFromWKB`. This method implements the SQL/MM specification. SQL-MM 3: 5.1.36
- **ST\_WKTToSQL** - WKT(Well-Known Text) `ST_Geometry ST_GeomFromText`. This method implements the SQL/MM specification. SQL-MM 3: 5.1.34
- **ST\_Within** - Tests if no points of A lie in the exterior of B, and A and B have at least one interior point in common. This method implements the SQL/MM specification. SQL-MM 3: 5.1.30
- **ST\_X** - Returns the X coordinate of a Point. This method implements the SQL/MM specification. SQL-MM 3: 6.1.3
- **ST\_Y** - Returns the Y coordinate of a Point. This method implements the SQL/MM specification. SQL-MM 3: 6.1.4
- **ST\_Z** - Returns the Z coordinate of a Point. This method implements the SQL/MM specification.
- **TG\_ST\_SRID** - Returns the spatial reference identifier for a topogeometry. This method implements the SQL/MM specification. SQL-MM 3: 14.1.5

## 15.4 PostGIS Geography Support Functions

The functions and operators given below are PostGIS functions/operators that take as input or return as output a **geography** data type object.



### Note

Functions with a (T) are not native geodetic functions, and use a `ST_Transform` call to and from geometry to do the operation. As a result, they may not behave as expected when going over dateline, poles, and for large geometries or geometry pairs that cover more than one UTM zone. Basic transform - (favoring UTM, Lambert Azimuthal (North/South), and falling back on mercator in worst case scenario)

- **ST\_Area** - Returns the area of a geometry.
- **ST\_AsBinary** - Return the OGC/ISO Well-Known Binary (WKB) representation of the geometry/geography without SRID meta data.
- **ST\_AsEWKT** - Returns the Well-Known Text (WKT) representation of the geometry/geography without SRID meta data.
- **ST\_AsGML** - Returns the GML 2 or GML 3 representation of the geometry/geography.
- **ST\_AsGeoJSON** - Return a geometry as a GeoJSON element.
- **ST\_AsKML** - Returns the KML representation of the geometry/geography.
- **ST\_AsSVG** - Returns SVG path data for a geometry.
- **ST\_AsText** - Returns the Well-Known Text (WKT) representation of the geometry/geography.
- **ST\_Azimuth** - Returns the azimuth of a line segment.
- **ST\_Buffer** - Computes a geometry covering all points within a given distance from a geometry.
- **ST\_Centroid** - Returns the centroid of a geometry.

- **ST\_CoveredBy** - Tests if no point in A is outside B
- **ST\_Covers** - Tests if no point in B is outside A
- **ST\_DWithin** - Tests if two geometries are within a given distance
- **ST\_GeogFromText** - WKT (`<math>POINT(100 100)</math>'); EWKB(<math>EWKB(POINT(100 100))</math>'); WKB(<math>WKB(POINT(100 100))</math>')`
- **ST\_GeogFromWKB** - WKB (`<math>EWKB(POINT(100 100))</math>'); WKB(<math>WKB(POINT(100 100))</math>')`
- **ST\_GeographyFromText** - WKT (`<math>POINT(100 100)</math>'); EWKB(<math>EWKB(POINT(100 100))</math>');`
- **=** - Returns TRUE if the coordinates and coordinate order geometry/geography A are the same as the coordinates and coordinate order of geometry/geography B.
- **ST\_Intersection** - Computes a geometry representing the shared portion of geometries A and B.
- **ST\_Intersects** - Tests if two geometries intersect (they have at least one point in common).
- **ST\_Length** - Returns the length of the boundary of a polygonal geometry or geography.
- **ST\_Perimeter** - Returns the length of the boundary of a polygonal geometry or geography.
- **ST\_Project** - Returns a geometry representing the point on the boundary of a polygonal geometry or geography that is closest to the point (x, y).
- **ST\_Segmentize** - Returns a geometry representing the boundary of a polygonal geometry or geography that is segmented into segments of a given maximum length.
- **ST\_Summary** - Returns a text string summarizing the geometry or geography.
- **<->** - Returns a text string representing the geometry or geography.
- **&&** - Returns a text string representing the geometry or geography.

## 15.5 PostGIS Raster Support Functions

The functions and operators given below are PostGIS functions/operators that take as input or return as output a **raster** data type object. Listed in alphabetical order.

- **Box3D** - Returns a 3D bounding box for a raster.
- **@** - Returns a raster with the same geometry as the input raster, but with the value of the input raster at the location of the input raster.
- **~** - Returns a raster with the same geometry as the input raster, but with the value of the input raster at the location of the input raster.
- **=** - Returns a raster with the same geometry as the input raster, but with the value of the input raster at the location of the input raster.



- **ST\_BandIsNoData** - NODATA
- **ST\_BandMetaData** -
- **ST\_BandNoDataValue** - NODATA
- **ST\_BandPath** -
- **ST\_BandPixelType** -
- **ST\_Clip** -
- **ST\_ColorMap** - 8BUI (grayscale, RGB, RGBA)
- **ST\_Contains** -
- **ST\_ContainsProperly** -
- **ST\_Count** - Generates a set of vector contours from the provided raster band, using the GDAL contouring algorithm.
- **ST\_ConvexHull** -
- **ST\_Count** -

- **ST\_CountAgg** - `&#xc885;&#xd569;&#xd568;&#xc218;&#xc785;&#xb2c8;&#xb2e4;. &#xb798;&#xc2a4;&#xd130; &#xc9d1;&#xc785;&#xb825; &#xbc34;&#xb4dc;&#xc5d0; &#xc788;&#xb294; &#xd53d;&#xc140; &#xac1c;&#xc218;&#xb97c; &#xbc18; &#xbc34;&#xb4dc;&#xb97c; &#xb530;&#xb85c; &#xc124;&#xc815;&#xd558;&#xc9c0; &#xc54a;&#xc744; &#xacbd;&#xc6b0; &#xae30;&#xbcf8;&#xac12;&#xc740; &#xbc34;&#xb4dc; 1&#xc785;&#xb2c8;&#xb2e4;. exclude_nodata_value&#xb97c; &#xcc38;&#xc73c;&#xb85c; &#xc124;&#xc815;&#xd560; &#xacbd;&#xc6b0;. NODATA &#xac12;&#xc774; &#xc544;&#xb2cc; &#xd53d;&#xc140;&#xc758; &#xac1c;&#xc218;&#xb9cc; &#xbc18;&#xd658;&#xd560; &#xac83;&#xc785;&#xb2c8;&#xb2e4;.`
- **ST\_CoveredBy** - `&#xb798;&#xc2a4;&#xd130; rastA&#xc758; &#xc5b4;&#xb5a4; &#xd3ec;&#xc778;&#xd2b8;&#xb3c4; &#xb798;&#xc2a4;&#xd130; rastB &#xc678;&#xbd80;&#xc5d0; &#xc5c6;&#xc744; &#xacbd;&#xc6b0; &#xcc38;&#xc744; &#bc18;&#xd658;&#xd569;&#xb2c8;&#xb2e4;.`
- **ST\_Covers** - `&#xb798;&#xc2a4;&#xd130; rastB&#xc758; &#xc5b4;&#xb5a4; &#xd3ec;&#xc778;&#xd2b8;&#xb3c4; &#xb798;&#xc2a4;&#xd130; rastA &#xc678;&#xbd80;&#xc5d0; &#xc5c6;&#xc744; &#xacbd;&#xc6b0; &#xcc38;&#xc744; &#bc18;&#xd658;&#xd569;&#xb2c8;&#xb2e4;.`
- **ST\_DFullyWithin** - `&#xb798;&#xc2a4;&#xd130; rastA&#xc640; &#xb798;&#xc2a4;&#xd130; rastB&#xc640; &#xc644;&#xc804; &#xc11c;&#xb85c; &#xc124;&#xc815;&#xb41c; &#xac70;&#xb9ac; &#xc548;&#xc5d0; &#xc788;&#xc744; &#xacbd;&#xc6b0; &#xcc38;&#xc744; &#bc18;&#xd658;&#xd569;&#xb2c8;&#xb2e4;.`
- **ST\_DWithin** - `&#xb798;&#xc2a4;&#xd130; rastA&#xc640; &#xb798;&#xc2a4;&#xd130; rastB&#xc640; &#xc11c;&#xb85c; &#xc124;&#xc815;&#xb41c; &#xac70;&#xb9ac; &#xc548;&#xc5d0; &#xc788;&#xc744; &#xacbd;&#xc6b0; &#xcc38;&#xc744; &#bc18;&#xd658;&#xd569;&#xb2c8;&#xb2e4;.`
- **ST\_Disjoint** - `&#xb798;&#xc2a4;&#xd130; rastA&#xc640; &#xb798;&#xc2a4;&#xd130; rastB&#xc640; &#xacf5;&#xc04;&#xc804;&#xad50;&#xcc28;&#xd558;&#xc9c0; &#xc54a;&#xc744; &#xacbd;&#xc6b0; &#xcc38;&#xc744; &#bc18;&#xd658;&#xd569;&#xb2c8;&#xb2e4;.`
- **ST\_DumpAsPolygons** - `&#xc785;&#xb825; &#xb798;&#xc2a4;&#xd130; &#xbc34;&#xb4dc;&#xb85c;&#xbd80;&#xd130; geomval(geom, val) &#xd589;&#xb4e4;&#xc758; &#xc9d1;&#xd569;&#xc744; &#bc18;&#xd658;&#xd569;&#xb2c8;&#xb2e4;. &#bc34;&#xb4dc; &#bc88;&#xd638;&#xb97c; &#xc124;&#xc815;&#xd558;&#xc9c0; &#xc54a;&#xc744; &#xacbd;&#xc6b0; &#xae30;&#xbcf8;&#xc801;&#xc73c;&#xb85c; &#bc34;&#xb4dc; 1&#xb85c; &#xc00;&#xc815;&#xd569;&#xb2c8;&#xb2e4;.`
- **ST\_DumpValues** - `&#xc9c0;&#xc815;&#xb41c; &#bc34;&#xb4dc;&#xc758; &#xac12;&#xb4e4;&#xc744; 2&#xcc28;&#xc6d0; &#bc30;&#xc5f4;&#xb85c; &#bc18;&#xd658;&#xd569;&#xb2c8;&#xb2e4;.`
- **ST\_Envelope** - `&#xb798;&#xc2a4;&#xd130; &#bc94;&#xc704;&#xc758; &#xd3f4;&#xb9ac;&#xc04; &#xd45c;&#xd604;&#xc218;&#bc18;&#xd658;&#xd569;&#xb2c8;&#xb2e4;.`
- **ST\_FromGDALRaster** - `&#xc9c0;&#xc6d0; GDAL &#xb798;&#xc2a4;&#xd130; &#xd30c;&#xc77c;&#xb85c;&#xbd80;&#xd130; &#xb798;&#xc2a4;&#xd130;&#xb97c; &#bc18;&#xd658;&#xd569;&#xb2c8;&#xb2e4;.`
- **ST\_GeoReference** - `&#xc6d4;&#xb4dc;(world) &#xd30c;&#xc77c;&#xc5d0;&#xc11c; &#xd754;&#xd788; &#xbcf5; &#xc218; &#xc788;&#xb294; &#xc9c0;&#xb9ac;&#xcc38;&#xc870; &#ba54;&#xd0c0;&#xb370;&#xc774;&#xd130;&#xb97c; GDAL &#xb610;&#xb294; ESRI &#xd615;&#xc2dd;&#xc73c;&#xb85c; &#bc18;&#xd658;&#xd569;&#xb2c8;&#xb2e4;. &#xae30;&#xc785;&#xb2c8;&#xb2e4;.`
- **ST\_Grayscale** - Creates a new one-8BUI band raster from the source raster and specified bands representing Red, Green and Blue
- **ST\_HasNoBand** - `&#xc785;&#xb825;&#xb41c; &#bc34;&#xb4dc; &#bc88;&#xd638;&#xc5d0; &#bc34;&#xb4dc;&#xc00; &#xc5c6;&#xc744; &#xacbd;&#xc6b0; &#xcc38;&#xc744; &#bc18;&#xd658;&#xd569;&#xb2c8;&#xb2e4;. &#bc34;&#xb4dc; &#bc88;&#xd638;&#xb97c; &#xc124;&#xc815;&#xd558;&#xc9c0; &#xc54a;&#xc744; &#xacbd;&#xc6b0;. &#bc34;&#xb4dc; 1&#xb85c; &#xc00;&#xc815;&#xd569;&#xb2c8;&#xb2e4;.`
- **ST\_Height** - `&#xb798;&#xc2a4;&#xd130;&#xc758; &#xb192;&#xc774;&#xb97c; &#xd53d;&#xc140; &#xac1c;&#xc218;&#xb85c; &#bc18;&#xd658;&#xd569;&#xb2c8;&#xb2e4;.`
- **ST\_HillShade** - `&#xc785;&#xb825;&#xd55c; &#bc29;&#xc704;&#xc01;. &#xc00;&#xb3c4;&#xc01;. &#bc1d;&#xae30; &#bc0f; &#xcd95;&#xcc99;&#xc744; &#xc774;&#xc6a9;&#xd574;&#xc11c; &#xd45c;&#xc00; &#xb798;&#xc2a4;&#xd130; &#bc34;&#xb4dc;&#xc758; &#xc00;&#xc0c1;&#xc801;&#xc778; &#xc74c;&#xc601;&#xae30;&#xbcf5;&#xc744; &#bc18;&#xd658;&#xd569;&#xb2c8;&#xb2e4;.`
- **ST\_Histogram** - `&#xb48;(bin; &#xd788;&#xc2a4;&#xd1a0;&#xadf8;&#xb7a8; &#xd45c;&#xc2dc;&#xc5d0;&#xc11c; &#xc218;&#xb9c9;&#xb300;&#xb85c; &#xb098;&#xd0c0;&#xb098;&#xb294; &#xb2e8;&#xc704;) &#bc94;&#xc704;&#xb85c; &#xad6c; &#xb798;&#xc2a4;&#xd130; &#xb610;&#xb294; &#xb798;&#xc2a4;&#xd130; &#xc00;&#xc815;&#xd569;&#xb2c8;&#xb2e4;.`

&#xb370;&#xc774;&#xd130; &#xbd84;&#xd3ec;&#xb97c; &#xc694;&#xc57d;&#xd558;&#xb294; &#xb808;&#xcf54;&#xb4dc; &#xc9d1;&#xd569;&#xc744; &#xbc18;&#xd658;&#xd569;&#xb2c8;&#xb2e4;. &#xb530;&#xb85c; &#xc124;&#xc815;&#xd558; &#xc54a;&#xc744; &#xacbd;&#xc6b0; &#xbe48;&#xc758; &#xac1c;&#xc218;&#xb97c; &#xc790;&#xb3d9;&#xc73c;&#xb85c; &#xacc4;&#xc0b0;&#xd569;&#xb2c8;&#xb2e4;.

- **ST\_MakeEmptyRaster** - Interpolates a gridded surface based on an input set of 3-d points, using the X- and Y-values to position the points on the grid and the Z-value of the points as the surface elevation.
- **ST\_Intersection** - &#xb450; &#xb798;&#xc2a4;&#xd130;&#xc758; &#xaf5;&#xc720; &#xbd80;&#xbd84;&#xc744; &#xd45c;&#xb610;&#xb294; &#xbca1;&#xd130;&#xd654;&#xb41c; &#xb798;&#xc2a4;&#xd130;&#xc640; &#xb3c4;&#xd615;&#xc758; &#xae30;&#xd558;&#xd559;&#xc801; &#xad50;&#xcc28;&#xb97c; &#xd45c;&#xd604;&#xd558;&#xb294; &#xb798;&#xc2a4;&#xb610;&#xb294; &#xb3c4;&#xd615;-&#xd53d;&#xc140;&#xac12; &#xc30d;&#xc758; &#xc9d1;&#xd569;&#xc744; &#xbc18;
- **ST\_Intersects** - &#xb798;&#xc2a4;&#xd130; rastA&#xc640; &#xb798;&#xc2a4;&#xd130; rastB&#xac00; &#xaf5;&#xac04;&#xcad50;&#xcc28;&#xd560; &#xacbd;&#xc6b0; &#xcc38;&#xc744; &#bc18;&#xd658;&#xd569;&#xb2c8;&#xb2e4;.
- **ST\_IsEmpty** - &#xb798;&#xc2a4;&#xd130;&#xac00; &#be44;&#xc5b4; &#xc788;&#xc744; &#xacbd;&#xc6b0; (width = 0, height = 0) &#xcc38;&#xc744; &#bc18;&#xd658;&#xd569;&#xb2c8;&#xb2e4;. &#xadf8;&#xb807;&#xc9c0; &#xc54a;&#xc744; &#xacbd;&#xc6b0; &#xc70;&#xc9d3;&#xc744; &#bc18;&#xd658;&#xd569;&#xb2c8;&#xb2e4;.
- **ST\_MakeEmptyCoverage** - Cover georeferenced area with a grid of empty raster tiles.
- **ST\_MakeEmptyRaster** - &#xc124;&#xc815;&#xb41c; &#xcc28;&#xc6d0; (&#xb108;&#be44; &#xb192;&#xc774;), &#xc88c;&#xc0f; Y, &#xd53d;&#xc140; &#xd06c;&#xae30;, &#xd68c;&#xc804;(scalex, scaley, skewx & skewy) &#xadf8;&#xb9ac;&#xaf5;&#xac04; &#xcc38;&#xc870; &#xc2dc;&#xc2a4;&#xd15c;(SRID)&#xb97c; &#xac00;&#xc9c4; &#xd145; &#be48; (&#bc34;&#xb4dc;&#xac00; &#xc5c6;&#xb294;) &#xb798;&#xc2a4;&#xd130;&#xb97c; &#bc18;&#xd658;&#xd569;&#xb2c8; &#xb798;&#xc2a4;&#xd130;&#xb97c; &#xc785;&#xb825;&#xd560; &#xacbd;&#xc6b0;, &#xb3d9;&#xc77c;&#xd55c; &#xd06c; &#xc815;&#xb82c; &#bc29;&#xd5a5; &#xc0f; SRID&#xb97c; &#xac00;&#xc9c4; &#xc0c8; &#xb798;&#xc2a4;&#xd130;&#bc18;&#xd658;&#xd569;&#xb2c8;&#xb2e4;. SRID&#xb97c; &#xc0dd;&#xb7b5;&#xd560; &#xacbd;&#xc6b0;, &#xaf5;&#xc9c4; &#xcc38;&#xc870; &#xc2dc;&#xc2a4;&#xd15c;&#xc744; 0(unknown)&#xc73c;&#xb85c; &#xc124;&#xc815;&#xd569;&#xb2c8;
- &#xb0b4;&#xc7a5; &#xb9f5; &#xb300;&#xc218; &#xcf5c;&#bc31; &#xd568;&#xc218; - &#xcf5c;&#bc31; &#xd568;&#xc218; &#bc84;&#xc804; - &#b798;&#xc2a4;&#xd130; 1&#xac1c; &#xc774;&#xc0c1;, &#bc34;&#xb4dc; &#xc778;&#xb371;&#xc2a4; &#xadf8;&#xb9ac;&#xace0; &#xc0ac;&#xc6a9;&#xc790; &#xc9c0;&#xc815; &#xcf5c;&#bc31; &#xd568;&#xc218; 1&#xac1c;&#xc785;&#xb825;&#bc1b;&#xc544; &#bc34;&#xb4dc; 1&#xac1c;&#xb97c; &#xac00;&#xc9c4; &#b798;&#xc2a4;&#xd130; &#bc18;&#xd658;&#xd569;&#xb2c8;&#xb2e4;.
- **ST\_MapAlgebraExpr** - &#xb798;&#xc2a4;&#xd130; &#bc34;&#xb4dc; 1&#xac1c; &#bc84;&#xc804;: &#xc785;&#xb825; &#b798;&#xc2a4;&#xd130;&#xc5d0; &#b300;&#xd574; &#xc720;&#xd6a8;&#xd55c; PostgreSQL &#b300;&#xc218; &#xc5f0;&#xc0b0;&#xc744; &#xc801;&#xc6a9;&#xd574;&#xc11c; &#xd615;&#xc131;&#xb418;&#xace0;, &#xc124;&#xc815;&#xd53d;&#xc140; &#xc720;&#xd615;&#xc744; &#xac00;&#xc9c4;, &#bc34;&#xb4dc; 1&#xac1c;&#xb97c; &#xac00;&#xc9c4; &#xc0c8; &#b798;&#xc2a4;&#xd130;&#xb97c; &#xc0dd;&#xc131;&#xd569;&#xb2c8;&#xb2e4;. &#b530;&#xb85c; &#bc34; &#xc124;&#xc815;&#xd558;&#xc9c0; &#xc54a;&#xc744; &#xacbd;&#xc6b0;, &#bc34;&#xb4dc; 1&#xb85c; &#xac00;&#xc815;
- **ST\_MapAlgebraExpr** - &#xb798;&#xc2a4;&#xd130; &#bc34;&#xb4dc; 2&#xac1c; &#bc84;&#xc804;: &#xc785;&#xb825; &#b798;&#xc2a4;&#xd130; 2&#xac1c;&#xc5d0; &#b300;&#xd574; &#xc720;&#xd6a8;&#xd55c; PostgreSQL &#b300;&#xc218; &#xc5f0;&#xc0b0;&#xc744; &#xc801;&#xc6a9;&#xd574;&#xc11c; &#xd615;&#xc131;&#xb418;&#xace0;, &#xc124;&#xc815;&#xd53d;&#xc140; &#xc720;&#xd615;&#xc744; &#xac00;&#xc9c4;, &#bc34;&#xb4dc; 1&#xac1c;&#xb97c; &#xac00;&#xc9c4; &#xc0c8; &#b798;&#xc2a4;&#xd130;&#xb97c; &#xc0dd;&#xc131;&#xd569;&#xb2c8;&#xb2e4;. &#b530;&#xb85c; &#bc34; &#xc124;&#xc815;&#xd558;&#xc9c0; &#xc54a;&#xc744; &#xacbd;&#xc6b0;, &#xac01; &#b798;&#xc2a4;&#xd130;&#xc758; &#bc34;&#xb4dc; 1&#xb85c; &#xac00;&#xc815;&#xd569;&#xb2c8;&#xb2e4;. &#xcd9c;&#xb825; &#b798;&#xc2a4;&#xd130; &#xccab; &#bc88;&#xc9f8; &#b798;&#xc2a4;&#xd130;&#xac00; &#xc815;&#xc758;&#xd558;&#xb294; &#xadf8;&#xb9ac; &#xc0c1;&#xc5d0; (&#xcd95;&#xcc99;, &#xae30;&#xc6b8;&#xae30; &#bc0f; &#xd53d;&#xc140; &#baa8;&#xc11c;&#xb9ac; &#xc815;&#xb82c;&#xb420; &#xac83;&#xc785;&#xb2c8;&#xb2e4;. extenttype &#xd30c;&#xb77c;&#xbbf8;&#xd130;&#xac00; &#xcd9c;&#xb825; &#b798;&#xc2a4;&#xd130;&#xc758; &#bc94;&#xc704;&#xb97c; &#xc815;&#xc758;&#xd560; &#xac83;&#xd130; extenttype &#xc758; &#xac12;&#xc740; INTERSECTION, UNION, FIRST, SECOND&#xac00; &#b420; &#xc218; &#xc788;&#xc744;
- **ST\_MapAlgebraFct** - &#xb798;&#xc2a4;&#xd130; &#bc34;&#xb4dc; 1&#xac1c; &#bc84;&#xc804;: &#xc785;&#xb825; &#b798;&#xc2a4;&#xd130;&#xc5d0; &#b300;&#xd574; &#xc720;&#xd6a8;&#xd55c; PostgreSQL &#b300;&#xc218; &#xc5f0;&#xc0b0;&#xc744; &#xc801;&#xc6a9;&#xd574;&#xc11c; &#xd615;&#xc131;&#xb418;&#xace0;, &#xc124;&#xc815;&#xd558;&#xc9c0; &#xc54a;&#xc744; &#xacbd;&#xc6b0;, &#bc34;&#xb4dc; 1&#xb85c; &#xac00;&#xc815;

- &#xd53d;&#xc140; &#xc720;&#xd615;&#xc744; &#xac00;&#xc9c4;, &#xbc34;&#xb4dc; 1&#xac1c;&#xb97c; &#xac00;&#xc9c4;&#xc0c8; &#xb798;&#xc2a4;&#xd130;&#xb97c; &#xc0dd;&#xc131;&#xd569;&#xb2c8;&#xb2e4;. &#xb530;&#xb85c; &#xbc34;&#xc124;&#xc815;&#xd558;&#xc9c0; &#xc54a;&#xc744; &#xacbd;&#xc6b0;, &#xbc34;&#xb4dc; 1&#xb85c; &#xac00;&#xc815;
- **ST\_MapAlgebraFct** - &#xb798;&#xc2a4;&#xd130; &#xbc34;&#xb4dc; 2&#xac1c; &#xbc84;&#xc804;: &#xc785;&#xb825; &#xb798;&#xc2a4;&#xd130; 2&#xac1c;&#xc5d0; &#xb300;&#xd574; &#xc720;&#xd6a8;&#xd55c; PostgreSQL &#xd568;&#xc2 &#xc801;&#xc6a9;&#xd574;&#xc11c; &#xd615;&#xc131;&#xb418;&#xace0;, &#xc124;&#xc815;&#xd55c; &#xd53d;&#xc140; &#xc720;&#xd615;&#xc744; &#xac00;&#xc9c4;, &#xbc34;&#xb4dc; 1&#xac1c;&#xb97c; &#xac00;&#xc9c4; &#xc0c8; &#xb798;&#xc2a4;&#xd130;&#xb97c; &#xc0dd;&#xc131;&#xd569;&#xb2c8;&#xb2e4;. &#xb530;&#xb85c; &#xbc34;&#xb4dc; &#xc124;&#xc815;&#xd558;&#xc9c0; &#xc54a;&#xc744; &#xacbd;&#xc6b0;, &#xbc34;&#xb4dc; 1&#xb85c; &#xac00;&#xc815 &#xbc94;&#xc704; &#xc720;&#xd615;&#xc744; &#xb530;&#xb85c; &#xc124;&#xc815;&#xd558;&#xc9c0; &#xc54a;&#xc744; &#xacbd;&#xc6b0; &#xae30;&#xbcf8;&#xac12;&#xc740; INTERSECTION&#xc785;&#xb2c8;&#xb2e4;.
  - **ST\_MapAlgebraFctNgb** - &#xb798;&#xc2a4;&#xd130; &#xbc34;&#xb4dc; 1&#xac1c; &#xbc84;&#xc804;: &#xc0ac;&#xc6a9;&#&#xc9c0;&#xc815; PostgreSQL &#xd568;&#xc218;&#xb97c; &#xc774;&#xc6a9;&#xd558;&#xb294; &#xb9f5; &#xb300;&#xc21 &#xcd5c;&#xadfc;&#xc811; &#xc774;&#xc6c3;(Map Algebra Nearest Neighbor)&#xc785;&#xb2c8;&#xb2e4;. &#xc785;&#xb825 &#xb798;&#xc2a4;&#xd130; &#xbc34;&#xb4dc;&#xc758; &#xac12;&#xc758; &#xc774;&#xc6c3;(neighborhood)&#xc774; &#xad00;&#xb828;&#xb41c; PostgreSQL &#xc0ac;&#xc6a9;&#xc790; &#xd568;&#xc218;&#xac00; &#xcd9c;&#xb825;&#xd558 &#xac12;&#xc744; &#xac00;&#xc9c4; &#xb798;&#xc2a4;&#xd130;&#xb97c; &#xbc18;&#xd658;&#xd569;&#xb2c8;&#xb2e4;.
  - **ST\_MapAlgebraExpr** - &#xd45c;&#xd604;&#xc2dd; &#xbc84;&#xc804; - &#xc785;&#xb825; &#xb798;&#xc2a4;&#xd130; 1&#xac1c; &#xb610;&#xb294; 2&#xac1c;, &#bc34;&#xb4dc; &#xc778;&#xb371;&#xc2a4;, &#xadf8;&#xb9ac;&#xace0; &#xc0ac;&#xc6a9;&#xc790; &#xc9c0;&#xc815; SQL &#xd45c;&#xd604;&#xc2dd; 1&#xac1c; &#xc774;&#xc0c1;&#xc744; &#xc785;&#xb825;&#xbcb1;&#xc544; &#bc34;&#xb4dc; 1&#xac1c;&#xb97c; &#xac00;&#xc9c4; &#xb798;&#xc2a4;&#xd130; &#bc18;&#xd658;&#xd569;&#xb2c8;&#xb2e4;.
  - **ST\_MemSize** - &#xb798;&#xc2a4;&#xd130;&#xac00; &#xcc28;&#xc9c0;&#xd558;&#xb294; &#xacf5;&#xac04;&#xc758; &#xc6a9;&#xb7c9;&#xc744; (&#bc14;&#xc774;&#xd2b8; &#xb2e8;&#xc704;&#xb85c;) &#bc18;&#xd658;&#xd569;&#xb2c8;
  - **ST\_MetaData** - &#xb798;&#xc2a4;&#xd130; &#xac1d;&#xccb4;&#xc758; &#xd53d;&#xc140; &#xd06c;&#xae30;, &#xd68c;&#&#xc88c;&#xc0c1;&#xb2e8;, &#xc88c;&#xd558;&#xb2e8; &#xb4f1;&#xacfc; &#xac19;&#xc740; &#xae30;&#xbcf8; &#xba54;& &#bc18;&#xd658;&#xd569;&#xb2c8;&#xb2e4;.
  - **ST\_MinConvexHull** - &#xb798;&#xc2a4;&#xd130;&#xc758; NODATA &#xd53d;&#xc140;&#xc744; &#xc81c;&#xc678;&#xd55c &#xbcf8;&#xb85d; &#xaecd;&#xc9c8; &#xb3c4;&#xd615;&#xc744; &#bc18;&#xd658;&#xd569;&#xb2c8;&#xb2e4;.
  - **ST\_NearestValue** - columnx &#bc0f; rowy, &#xb610;&#xb294; &#xb798;&#xc2a4;&#xd130;&#xc640; &#xb3d9;&#xc77c;&#xd5 &#xacf5;&#xac04; &#xcc38;&#xc870; &#xc88c;&#xd45c; &#xc2dc;&#xc2a4;&#xd15c; &#xb2e8;&#xc704;&#xb85c; &#xd45c;& &#xae30;&#xd558;&#xd559;&#xc801; &#xd3ec;&#xc778;&#xd2b8;&#xb85c; &#xc9c0;&#xc815;&#xb41c; &#xc785;&#xb825; &#bc34;&#xb4dc;&#xc758; &#xd53d;&#xc140;&#xc5d0; &#xac00;&#xc7a5; &#xac00;&#xae4c;&#xc6b4; NODATA &#xac00; &#xc544;&#xb2cc; &#xac12;&#xc744; &#bc18;&#xd658;&#xd569;&#xb2c8;&#xb2e4;.
  - **ST\_Neighborhood** - columnx &#bc0f; rowy, &#xb610;&#xb294; &#xb798;&#xc2a4;&#xd130;&#xc640; &#xb3d9;&#xc77c;&#xd &#xacf5;&#xac04; &#xcc38;&#xc870; &#xc88c;&#xd45c; &#xc2dc;&#xc2a4;&#xd15c; &#xb2e8;&#xc704;&#xb85c; &#xd45c;& &#xae30;&#xd558;&#xd559;&#xc801; &#xd3ec;&#xc778;&#xd2b8;&#xb85c; &#xc9c0;&#xc815;&#xb41c; &#xc785;&#xb825; &#bc34;&#xb4dc;&#xc758; &#xd53d;&#xc140; &#xc8fc;&#xc704;&#xc758; NODATA &#xac00; &#xc544;&#xb2cc; &#xac12;& &#xc774;&#xc911; &#xc815;&#xbcb0;&#xb3c4; &#xb370;&#xc774;&#xd130;&#xd615; 2&#xcc28;&#xc6d0; &#bc30;&#xc5f4; &#bc18;&#xd658;&#xd569;&#xb2c8;&#xb2e4;.
  - **ST\_NotSameAlignmentReason** - &#xb798;&#xc2a4;&#xd130;&#xb4e4;&#xc774; &#xc815;&#xb82c;&#xb3fc; &#xc788;&#xb294 &#xc544;&#xb2cc;&#xc9c0;, &#xadf8;&#xb9ac;&#xace0; &#xc815;&#xb82c;&#xb418;&#xc9c0; &#xc54a;&#xc558;&#xb2e4;& &#xadf8; &#xc774;&#xc720;&#xb97c; &#xc124;&#xba85;&#xd558;&#xb294; &#xd14d;&#xc2a4;&#xd2b8;&#xb97c; &#bc18;&#&
  - **ST\_NumBands** - &#xb798;&#xc2a4;&#xd130; &#xac1d;&#xccb4; &#xb0b4;&#xbd80;&#xc5d0; &#xc788;&#xb294; &#bc34;&#&#xac1c;&#xc218;&#xb97c; &#bc18;&#xd658;&#xd569;&#xb2c8;&#xb2e4;.
  - **ST\_Overlaps** - &#xb798;&#xc2a4;&#xd130; rastA&#xc640; &#xb798;&#xc2a4;&#xd130; rastB&#xac00; &#xad50;&#xcc28;&#xd &#xc5b4;&#xb290; &#xd55c; &#xacbd;&#xc774; &#xb2e4;&#xb978; &#xd55c; &#xacbd;&#xc744; &#xc644;&#xc804;&#xd788; &#xb2f4;&#xace0; &#xc788;&#xc9c0;&#xb294; &#xc54a;&#xc744; &#xacbd;&#xc6b0; &#xcc38;&#xc744; &#bc18;&#xd658;&#&
  - **ST\_PixelAsCentroid** - &#xd53d;&#xc140; &#xd558;&#xb098;&#xac00; &#xcc28;&#xc9c0;&#xd558;&#xb294; &#xba74;&#xc758 &#xc911;&#xc2ec;&#xc810; (&#xd3ec;&#xc778;&#xd2b8; &#xb3c4;&#xd615;) &#xc744; &#bc18;&#xd658;&#xd569;&#xb2c8;

- **ST\_PixelAsCentroids** - Returns a set of points representing the centroids of the pixels in the raster. The output is a table with columns: X, Y, and a column for the pixel value. The output is a table with columns: X, Y, and a column for the pixel value.
- **ST\_PixelAsPoint** - Returns a single point representing the centroid of the raster.
- **ST\_PixelAsPoints** - Returns a set of points representing the centroids of the pixels in the raster. The output is a table with columns: X, Y, and a column for the pixel value.
- **ST\_PixelAsPolygon** - Returns a set of polygons representing the pixels in the raster. The output is a table with columns: X, Y, and a column for the pixel value.
- **ST\_PixelAsPolygons** - Returns a set of polygons representing the pixels in the raster. The output is a table with columns: X, Y, and a column for the pixel value.
- **ST\_PixelHeight** - Returns the height of the raster in meters.
- **ST\_PixelOfValue** - Returns the pixel value of the raster at the specified location.
- **ST\_PixelWidth** - Returns the width of the raster in meters.
- **ST\_Polygon** - Returns a set of polygons representing the pixels in the raster. The output is a table with columns: X, Y, and a column for the pixel value.
- **ST\_Quantile** - Returns the quantile of the raster values.
- **ST\_RastFromHexWKB** - Return a raster value from a Hex representation of Well-Known Binary (WKB) raster.
- **ST\_RastFromWKB** - Return a raster value from a Well-Known Binary (WKB) raster.
- **ST\_RasterToWorldCoord** - Returns the world coordinates of the pixel at the specified location.
- **ST\_RasterToWorldCoordX** - Returns the X world coordinate of the pixel at the specified location.
- **ST\_RasterToWorldCoordY** - Returns the Y world coordinate of the pixel at the specified location.
- **ST\_Reclass** - Reclassifies the raster values based on the specified reclassification table.

- &#xac00;&#xb9ac;&#xd0b5;&#xb2c8;&#xb2e4;. nband &#xb97c; &#xb530;&#xb85c; &#xc124;&#xc815;&#xd558;&#xc9c0;&#xc54a;&#xc744; &#xacbd;&#xc6b0; &#xbc34;&#xb4dc; 1&#xb85c; &#xac00;&#xc815;&#xd569;&#xb2c8;&#xb2e4;.&#xb2e4;&#xb978; &#xaa8;&#xb4e0; &#xc34;&#xb4dc;&#xb4e4;&#xc740; &#xbcc0;&#xacbd; &#xc5c6;&#xc774; &#xbc18;&#xc2e4;&#xc81c; &#xc0ac;&#xb840;: &#xbcf4;&#xae30; &#xc88b;&#xc740; &#xd615;&#xc2dd;&#xc73c;&#xb85c; &#xb354; &#xac04;&#xb2e8;&#xd558;&#xac8c; &#xb80c;&#xb354;&#xb9c1;&#xd558;&#xae30; &#xc704;&#xd574; 16BUI &#xc34;&#xb8BUI &#xb4f1;&#xb4f1;&#xc73c;&#xb85c; &#xbcc0;&#xd658;&#xd558;&#xc2ed;&#xc2dc;&#xc624;.
- **ST\_Resample** - &#xd2b9;&#xc815; &#xb9ac;&#xc0d8;&#xd50c;&#xb9c1; &#xc54c;&#xace0;&#xb9ac;&#xc998;,&#xc0c8;&#xb&#xc28;&#xc6d0;,&#xc784;&#xc758;&#xc758; &#xadf8;&#xb9ac;&#xb4dc; &#xaa8;&#xc11c;&#xb9ac;,&#xadf8;&#xb9ac;&#xb610; &#xb2e4;&#xb978; &#xb798;&#xc2a4;&#xd130;&#xc5d0;&#xc11c; &#xc815;&#xc758;&#xb418;&#xac70;&#xb098; &#bbe4c;&#xb824;&#xc628; &#xb798;&#xc2a4;&#xd130; &#xc9c0;&#xb9ac;&#xc38;&#xc870; &#xc18d;&#xc131;&#xb4e4;&#xc9d1;&#xd569;&#xc744; &#xc774;&#xc6a9;&#xd574;&#xc11c; &#xb798;&#xc2a4;&#xd130;&#xb97c; &#xb9ac;&#xc0d8;&#xc870;&#xc815;&#xd558;&#xb798;&#xc2a4;&#xd130;&#xb97c; &#xb9ac;&#xc0d8;&#xd50c;&#xb9c1;&#xd569;&#xb2c8;&#xb2e4;. NearestNeighbor(&#xc601;&#xad6d; &#xb610;&#xb294; &#bbf8;&#xad6d; &#cca0;&#xc790;), Bilinear, Cubic, CubicSpline &#xb610;&#xb294; Lanczos &#xb9ac;&#xc0d8;&#xd50c;&#xb9c1; &#xc54c;&#xace0;&#xb9ac;&#xc998;&#xc744; &#xc774;&#xc6a9;&#xd574;&#xc0c8; &#xd53d;&#xc140; &#xac12;&#xc744; &#xacc4;&#xc0b0;&#xd569;&#xb2c8;&#xb2e4;. &#xae30;&#xbcf8;&#xac12;&#xc744; NearestNeighbor&#xc785;&#xb2c8;&#xb2e4;.
  - **ST\_Resize** - &#xb798;&#xc2a4;&#xd130;&#xc758; &#xd06c;&#xae30;&#xb97c; &#xc0c8; &#xb108;&#xbe44;/&#xb192;&#xc774;&#xc870;&#xc815;&#xd569;&#xb2c8;&#xb2e4;.
  - **ST\_Reskew** - &#xae30;&#xc6b8;&#xae30;(&#xb610;&#xb294; &#xd68c;&#xc804;&#xac01; &#xd30c;&#xb77c;&#bbf8;&#xd130;&#xc870;&#xc815;&#xd574;&#xc11c; &#xb798;&#xc2a4;&#xd130;&#xb97c; &#xb9ac;&#xc0d8;&#xd50c;&#xb9c1;&#xd569;&#xc0c8; &#xb108;&#xbe44;/&#xb192;&#xc774;&#xc870;&#xc815;&#xd574;&#xc11c; &#xb798;&#xc2a4;&#xd130;&#xb97c; &#xb9ac;&#xc0d8;&#xd50c;&#xb9c1; &#xc54c;&#xace0;&#xb9ac;&#xc998;&#xc744; &#xc774;&#xc6a9;&#xd574;&#xc11c; &#xc0c8; &#xd53d;&#xc140; &#xac12;&#xc744; &#xacc4;&#xc0b0;&#xd569;&#xb2c8;&#xb2e4;. &#xae30;&#xbcf8;&#xac12;&#xc744; NearestNeighbor&#xc785;&#xb2c8;&#xb2e4;.
  - **ST\_Rotation** - &#xb798;&#xc2a4;&#xd130;&#xc758; &#xd68c;&#xc804;&#xac01;&#xc744; &#xb77c;&#xb514;&#xc548;&#xc774;&#xc18;&#xd658;&#xd569;&#xb2c8;&#xb2e4;.
  - **ST\_Roughness** - DEM&#xc758; &#xacc4;&#xc0b0;&#xb41c; " &#xac70;&#xc60;&#xae30;(roughness)" &#xc640; &#xd568;&#xb798;&#xc2a4;&#xd130;&#xb97c; &#xc18;&#xd658;&#xd569;&#xb2c8;&#xb2e4;.
  - **ST\_SRID** - spatial\_ref\_sys &#xd14c;&#xc774;&#xbe14;&#xc5d0; &#xc815;&#xc758;&#xb418;&#xc5b4; &#xc788;&#xb294;,&#xb798;&#xc2a4;&#xd130;&#xc758; &#xacf5;&#xac04; &#xc38;&#xc870; &#xc2dd;&#xbcc4;&#xc790;&#xb97c; &#xc18;&#xc774;&#xc624;.
  - **ST\_SameAlignment** - &#xb798;&#xc2a4;&#xd130;&#xb4e4;&#xc774; &#xb3d9;&#xc77c;&#xd55c; &#xae30;&#xc6b8;&#xae30; &#xcd95;&#xc99;,&#xacf5;&#xac04; &#xc38;&#xc870; &#xc2dc;&#xc2a4;&#xd15c; &#xadf8;&#xb9ac;&#xace0; &#xc624;&#xac00;&#xc9c0;&#xace0; &#xc788;&#xc744; &#xacbd;&#xc6b0; (&#xd53d;&#xc140;&#xc774; &#xb2e4;&#xb978; &#xd53d;&#xc68;&#xc94;&#xd558;&#xc9c0; &#xc54a;&#xace0; &#xb3d9;&#xc77c;&#xd55c; &#xadf8;&#xb9ac;&#xb4dc;&#xc5d0; &#xb193;&#xc77c; &#xc218; &#xc788;&#xc744; &#xacbd;&#xc6b0;) &#xc38;&#xc744; &#xc18;&#xd658;&#xd558;&#xace0; &#xadf8;&#xb807;&#xc9c0; &#xc54a;&#xc744; &#xacbd;&#xc6b0; &#xbb38;&#xc81c;&#xc810;&#xc744; &#xc124;&#xba85;&#xc548;&#xb0b4;&#xbb38;&#xacfc; &#xd568;&#xaed8; &#xac70;&#xc9d3;&#xc744; &#xc18;&#xd658;&#xd569;&#xb2c8;&#xb2e4;.
  - **ST\_ScaleX** - &#xd53d;&#xc140; &#xb108;&#xbe44;&#xc758; X &#xc694;&#xc18c;&#xb97c; &#xc88c;&#xd45c; &#xc38;&#xc2dc;&#xc2a4;&#xd15c;&#xc758; &#xb2e8;&#xc704;&#xb85c; &#xc18;&#xd658;&#xd569;&#xb2c8;&#xb2e4;.
  - **ST\_ScaleY** - &#xd53d;&#xc140; &#xb192;&#xc774;&#xc758; Y &#xc694;&#xc18c;&#xb97c; &#xc88c;&#xd45c; &#xc38;&#xc2dc;&#xc2a4;&#xd15c;&#xc758; &#xb2e8;&#xc704;&#xb85c; &#xc18;&#xd658;&#xd569;&#xb2c8;&#xb2e4;.
  - **ST\_SetBandIndex** - Update the external band number of an out-db band
  - **ST\_SetBandIsNoData** - &#xc34;&#xb4dc;&#xc758; isnodata &#xd50c;&#xb798;&#xadf8;&#xb97c; &#xc38;&#xc73c;&#xb85c; &#xc124;&#xc815;&#xd569;&#xb2c8;&#xb2e4;.
  - **ST\_SetBandNoDataValue** - NODATA&#xb97c; &#xb098;&#xd0c0;&#xb0b4;&#xb294; &#xc785;&#xb825; &#xc34;&#xb4dc;&#xac12;&#xc744; &#xc124;&#xc815;&#xd569;&#xb2c8;&#xb2e4;. &#xc34;&#xb4dc;&#xb97c; &#xb530;&#xb85c; &#xc124;&#xc54a;&#xc744; &#xacbd;&#xc6b0; &#xc34;&#xb4dc; 1&#xb85c; &#xac00;&#xc815;&#xd569;&#xb2c8;&#xb2e4;. &#xc34;&#xb4dc;&#xc5d0; NODATA&#xac00; &#xc5c6;&#xb2e4;&#xace0; &#xd45c;&#xc2dc;&#xd558;&#xb824;&#xba74;,&#xc774;&#xb77c;&#xace0; &#xc124;&#xc815;&#xd558;&#xc2ed;&#xc2dc;&#xc624;.





- `ST_Width` - Returns the width of a raster in pixels.
- `ST_Within` - Returns true if a raster is within the bounding box of another raster.
- `ST_WorldToRasterCoord` - Returns the raster coordinates for a given world coordinate.
- `ST_WorldToRasterCoordX` - Returns the X raster coordinate for a given world coordinate.
- `ST_WorldToRasterCoordY` - Returns the Y raster coordinate for a given world coordinate.
- `UpdateRasterSRID` - Updates the SRID of a raster.
- `ST_PixelOfValue` - Returns the value of a pixel in a raster.

## 15.6 PostGIS Geometry / Geography / Raster Dump Functions

The functions given below are PostGIS functions that take as input or return as output a set of or single `geometry_dump` or `geomval` data type object.

- `ST_DumpAsPolygons` - Returns a set of polygons from a raster.
- `ST_Intersection` - Returns the intersection of two rasters.

## 15.7 PostGIS Box Functions

The functions given below are PostGIS functions that take as input or return as output the box\* family of PostGIS spatial types. The box family of types consists of `box2d`, and `box3d`

- `Box2D` - Returns a BOX2D representing the 2D extent of a geometry.
- `Box3D` - Returns a BOX3D representing the 3D extent of a geometry.

- **Box3D** - Returns the 3D bounding box of a geometry.
- **ST\_3DExtent** - Aggregate function that returns the 3D bounding box of geometries.
- **ST\_3DMakeBox** - Creates a BOX3D defined by two 3D point geometries.
- **ST\_AsMVTGeom** - Transforms a geometry into the coordinate space of a MVT tile.
- **ST\_AsTWKB** - Returns the TWKB (Tiny Well-Known Binary) representation of a geometry.
- **ST\_Box2dFromGeoHash** - GeoHash to 2D bounding box.
- **ST\_ClipByBox2D** - Computes the portion of a geometry falling within a rectangle.
- **ST\_EstimatedExtent** - Returns the estimated extent of a spatial table.
- **ST\_Expand** - Returns a bounding box expanded from another bounding box or a geometry.
- **ST\_Extent** - Aggregate function that returns the bounding box of geometries.
- **ST\_MakeBox2D** - Creates a BOX2D defined by two 2D point geometries.
- **ST\_XMax** - Returns the X maxima of a 2D or 3D bounding box or a geometry.
- **ST\_XMin** - Returns the X minima of a 2D or 3D bounding box or a geometry.
- **ST\_YMax** - Returns the Y maxima of a 2D or 3D bounding box or a geometry.
- **ST\_YMin** - Returns the Y minima of a 2D or 3D bounding box or a geometry.
- **ST\_ZMax** - Returns the Z maxima of a 2D or 3D bounding box or a geometry.
- **ST\_ZMin** - Returns the Z minima of a 2D or 3D bounding box or a geometry.
- **RemoveUnusedPrimitives** - Removes topology primitives which not needed to define existing TopoGeometry objects.
- **ValidateTopology** - Returns a set of validate\_topology\_return\_type objects detailing issues with topology.
- **~(box2df,box2df)** - Returns TRUE if a 2D float precision bounding box (BOX2DF) contains another 2D float precision bounding box (BOX2DF).
- **~(box2df,geometry)** - Returns TRUE if a 2D float precision bounding box (BOX2DF) contains a geometry's 2D bounding box.
- **~(geometry,box2df)** - Returns TRUE if a geometry's 2D bounding box contains a 2D float precision bounding box (BOX2DF).
- **@(box2df,box2df)** - Returns TRUE if a 2D float precision bounding box (BOX2DF) is contained into another 2D float precision bounding box.
- **@(box2df,geometry)** - Returns TRUE if a 2D float precision bounding box (BOX2DF) is contained into a geometry's 2D bounding box.
- **@(geometry,box2df)** - Returns TRUE if a geometry's 2D bounding box is contained into a 2D float precision bounding box (BOX2DF).
- **&&(box2df,box2df)** - Returns TRUE if two 2D float precision bounding boxes (BOX2DF) intersect each other.
- **&&(box2df,geometry)** - Returns TRUE if a 2D float precision bounding box (BOX2DF) intersects a geometry's (cached) 2D bounding box.
- **&&(geometry,box2df)** - Returns TRUE if a geometry's (cached) 2D bounding box intersects a 2D float precision bounding box (BOX2DF).

## 15.8 PostGIS Functions that support 3D

The functions given below are PostGIS functions that do not throw away the Z-Index.

- **AddGeometryColumn** - Adds a geometry column to a table.
- **Box3D** - Returns a BOX3D representing the 3D extent of a geometry.
- **DropGeometryColumn** - Drops a geometry column from a table.
- **GeometryType** - Returns the geometry type of a geometry.
- **ST\_3DArea** - Returns the 3D area of a geometry.
- **ST\_3DClosestPoint** - Returns the 3D closest point to a geometry.
- **ST\_3DConvexHull** - Returns the 3D convex hull of a geometry.
- **ST\_3DDFullyWithin** - Tests if two 3D geometries are entirely within a given 3D distance.
- **ST\_3DDWithin** - Tests if two 3D geometries are within a given 3D distance.
- **ST\_3DDifference** - Returns the 3D difference of two geometries.
- **ST\_3DDistance** - Returns the 3D distance between two geometries.
- **ST\_3DExtent** - Aggregate function that returns the 3D bounding box of geometries.
- **ST\_3DIntersection** - Returns the 3D intersection of two geometries.
- **ST\_3DIntersects** - Tests if two geometries spatially intersect in 3D - only for points, linestrings, polygons, polyhedral surface (area).
- **ST\_3DLength** - Returns the 3D length of a geometry.
- **ST\_3DLineInterpolatePoint** - Returns a point interpolated along a 3D line at a fractional location.
- **ST\_3DLongestLine** - Returns the 3D longest line of a geometry.
- **ST\_3DMaxDistance** - Returns the 3D maximum distance between two geometries.
- **ST\_3DPerimeter** - Returns the 3D perimeter of a geometry.
- **ST\_3DShortestLine** - Returns the 3D shortest line of a geometry.
- **ST\_3DUnion** - Perform 3D union.
- **ST\_AddMeasure** - Interpolates measures along a linear geometry.
- **ST\_AddPoint** - Adds a point to a geometry.
- **ST\_Affine** - Apply a 3D affine transformation to a geometry.

- **ST\_ApproximateMedialAxis** - Returns the approximate medial axis of a geometry.
- **ST\_AsBinary** - Return the OGC/ISO Well-Known Binary (WKB) representation of the geometry/geography without SRID meta data.
- **ST\_AsEWKB** - Return the Extended Well-Known Binary (EWKB) representation of the geometry with SRID meta data.
- **ST\_AsEWKT** - Returns the Well-Known Text (WKT) representation of the geometry/geography with SRID meta data.
- **ST\_AsGML** - Returns the GML 2 or GML 3 representation of the geometry/geography.
- **ST\_AsGeoJSON** - Return a geometry as a GeoJSON element.
- **ST\_AsHEXEWKB** - Returns the HEXEWKB representation of the geometry/geography.
- **ST\_AsKML** - Returns the KML representation of the geometry/geography.
- **ST\_AsX3D** - Returns the X3D XML representation of the geometry/geography.
- **ST\_Boundary** - Returns the boundary of a geometry.
- **ST\_BoundingDiagonal** - Returns the bounding diagonal of a geometry.
- **ST\_CPAWithin** - Tests if the closest point of approach of two trajectories is within the specified distance.
- **ST\_ClosestPointOfApproach** - Returns a measure at the closest point of approach of two trajectories.
- **ST\_Collect** - Creates a GeometryCollection or Multi\* geometry from a set of geometries.
- **ST\_ConstrainedDelaunayTriangles** - Return a constrained Delaunay triangulation around the given input geometry.
- **ST\_ConvexHull** - Computes the convex hull of a geometry.
- **ST\_CoordDim** - Returns the coordinate dimension of a geometry.
- **ST\_CurveToLine** - Converts a geometry containing curves to a linear geometry.
- **ST\_DelaunayTriangles** - Returns the Delaunay triangulation of the vertices of a geometry.
- **ST\_Difference** - Computes a geometry representing the part of geometry A that does not intersect geometry B.
- **ST\_DistanceCPA** - Returns the distance between the closest point of approach of two trajectories.
- **ST\_Dump** - Returns a set of geometry\_dump rows for the components of a geometry.
- **ST\_DumpPoints** - Returns a set of geometry\_dump rows for the vertices of a geometry.
- **ST\_DumpRings** - Returns a set of geometry\_dump rows for the exterior and interior rings of a Polygon.
- **ST\_DumpSegments** - Returns a set of geometry\_dump rows for the segments of a geometry.
- **ST\_EndPoint** - Returns the end point of a geometry.

- **ST\_ExteriorRing** - Returns the exterior ring of a geometry.
- **ST\_Extrude** - Extrudes a 2D geometry into 3D.
- **ST\_FlipCoordinates** - Returns a version of a geometry with X and Y axis flipped.
- **ST\_Force2D** - Forces a 3D geometry to be 2D.
- **ST\_ForceCurve** - Forces a geometry to be a curve.
- **ST\_ForceLHR** - LHR (Left Hand Reverse) orientation.
- **ST\_ForcePolygonCCW** - Orients all exterior rings counter-clockwise and all interior rings clockwise.
- **ST\_ForcePolygonCW** - Orients all exterior rings clockwise and all interior rings counter-clockwise.
- **ST\_ForceRHR** - RHR (Right Hand Rule) orientation.
- **ST\_ForceSFS** - SFS 1.1 force function.
- **ST\_Force\_3D** - Forces a 2D geometry to be 3D.
- **ST\_Force\_3DZ** - Forces a 2D geometry to be 3D with Z.
- **ST\_Force\_4D** - Forces a 2D geometry to be 4D.
- **ST\_Force\_Collection** - Forces a collection of geometries to be a specific type.
- **ST\_GeomFromEWKB** - EWKB (Extended Well-Known Binary) format.
- **ST\_GeomFromEWKT** - EWKT (Extended Well-Known Text) format.
- **ST\_GeomFromGML** - GML (Geography Markup Language) format.
- **ST\_GeomFromGeoJSON** - GeoJSON format.
- **ST\_GeomFromKML** - KML (Keyhole Markup Language) format.
- **ST\_GeometricMedian** - Finds the geometric median of a set of points.
- **ST\_GeometryN** - Returns the Nth geometry in a collection.
- **ST\_GeometryType** - Returns the type of a geometry.
- **ST\_HasArc** - Tests if a geometry contains a circular arc.
- **ST\_InteriorRingN** - Returns the Nth interior ring of a geometry.
- **ST\_InterpolatePoint** - Interpolates a point along a line.

- **ST\_Intersection** - Computes a geometry representing the shared portion of geometries A and B.
- **ST\_IsClosed** - LINESTRING &#xc758; &#xc2dc; &#xc791; &#xc810; &#xacfc; &#xc885; &#xb2e8; &#xc810; &#xc774; &#xc77c; &#xacbd; &#xc6b0; TRUE &#xb97c; &#xbc18; &#xd658; &#xd569; &#xb2c8; &#xb2e4;. &#xb2e4; &#xba74; &#xccb4; &#xd45c; &#xb2eb; &#xd600; (&#xbd80; &#xd53c; &#xb97c; &#xac00; &#xc9c0; &#xace0;) &#xc788; &#xb294; &#xacbd; &#xc6b0; TRUE &#xb97c; &#xbc18; &#xd658; &#xd569; &#xb2c8; &#xb2e4;.
- **ST\_IsCollection** - &#xd574; &#xb2f9; &#xb3c4; &#xd615; &#xc774; &#xd145; &#xbe48; &#xb3c4; &#xd615; &#xc9d1; &#xd569; &#xd3f4; &#xb9ac; &#xace4; &#xd3ec; &#xc778; &#xd2b8; &#xb4f1; &#xc778; &#xacbd; &#xc6b0; TRUE &#xb97c; &#xbc18; &#xd658; &#xd569; &#xb2c8; &#xb2e4;.
- **ST\_IsPlanar** - &#xd45c; &#xba74; &#xc774; &#xd3c9; &#xba74; &#xc778; &#xc9c0; &#xc544; &#xb2cc; &#xc9c0; &#xd655; &#xc774; &#xd615; &#xc774; &#xc790; &#xccb4; &#xad50; &#xcc28; &#xd558; &#xac70; &#xb000; &#xc790; &#xccb4; &#xc811; &#xcd09; &#xd558; &#xb294; &#xc774; &#xb840; &#xc801; &#xc778; &#xb3c4; &#xd615; &#xd3ec; &#xcac0; &#xc9c0; &#xace0; &#xc788; &#xc9c0; &#xc54a; &#xc744; &#xacbd; &#xc6b0; TRUE &#xb97c; &#xbc18; &#xd658; &#xd569; &#xb2c8; &#xb2e4;.
- **ST\_IsSolid** - &#xb3c4; &#xd615; &#xc774; &#xc785; &#xccb4; &#xc778; &#xc9c0; &#xd14c; &#xc2a4; &#xd2b8; &#xd569; &#xb2c8; &#xc5b4; &#xb5a4; &#xc720; &#xd6a8; &#xc131; &#xac80; &#xc0ac; &#xb3c4; &#xc218; &#xd589; &#xd558; &#xc9c0; &#xc54a; &#xc774; &#xc790; &#xccb4; &#xad50; &#xcc28; &#xd558; &#xac70; &#xb000; &#xc790; &#xccb4; &#xc811; &#xcd09; &#xd558; &#xb294; &#xc774; &#xb840; &#xc801; &#xc778; &#xb3c4; &#xd615; &#xd3ec; &#xcac0; &#xc9c0; &#xace0; &#xc788; &#xc9c0; &#xc54a; &#xc744; &#xacbd; &#xc6b0; TRUE &#xb97c; &#xbc18; &#xd658; &#xd569; &#xb2c8; &#xb2e4;.
- **ST\_IsValidTrajectory** - Tests if the geometry is a valid trajectory.
- **ST\_Length\_Spheroid** - &#xb3c4; &#xd615; &#xc758; &#xae30; &#xd558; &#xd559; &#xc801; &#xc911; &#xc2ec; &#xc744; &#xbc18; &#xd658; &#xd569; &#xb2c8; &#xb2e4;.
- **ST\_LineFromMultiPoint** - &#xba40; &#xd2f0; &#xd3ec; &#xc778; &#xd2b8; &#xb3c4; &#xd615; &#xc73c; &#xb85c; &#xbd80; &#xd3ec; &#xc778; &#xc2a4; &#xd2b8; &#xb9c1; &#xc744; &#xc0dd; &#xc131; &#xd569; &#xb2c8; &#xb2e4;.
- **ST\_LineInterpolatePoint** - Returns a point interpolated along a line at a fractional location.
- **ST\_LineInterpolatePoints** - Returns points interpolated along a line at a fractional interval.
- **ST\_LineSubstring** - Returns the part of a line between two fractional locations.
- **ST\_LineToCurve** - Converts a linear geometry to a curved geometry.
- **ST\_LocateBetweenElevations** - Returns the portions of a geometry that lie in an elevation (Z) range.
- **ST\_M** - Returns the M coordinate of a Point.
- **ST\_MakeLine** - &#xd3ec; &#xc778; &#xd2b8; &#xba40; &#xd2f0; &#xd3ec; &#xc778; &#xd2b8; &#xb610; &#xb294; &#xb77c; &#xb3c4; &#xd615; &#xc73c; &#xb85c; &#xbd80; &#xd130; &#xb77c; &#xc778; &#xc2a4; &#xd2b8; &#xb9c1; &#xc744; &#xc0dd; &#xc131; &#xd569; &#xb2c8; &#xb2e4;.
- **ST\_MakePoint** - Creates a 2D, 3DZ or 4D Point.
- **ST\_MakePolygon** - Creates a Polygon from a shell and optional list of holes.
- **ST\_MakeSolid** - &#xb3c4; &#xd615; &#xc744; &#xc785; &#xccb4; &#xb85c; &#xc9c0; &#xc815; &#xd569; &#xb2c8; &#xb2e4;. &#xc5b4; &#xb5a4; &#xd655; &#xc778; &#xc791; &#xc5c5; &#xb3c4; &#xc218; &#xd589; &#xd558; &#xc9c0; &#xc54a; &#xc2b5; &#xc720; &#xd6a8; &#xd55c; &#xc785; &#xccb4; &#xb97c; &#xc5bb; &#xc73c; &#xb824; &#xba74; &#xc785; &#xb825; &#xb3c4; &#xb2eb; &#xd78c; &#xb2e4; &#xba74; &#xccb4; &#xd45c; &#ba74; &#xb610; &#xb294; &#xb2eb; &#xd78c; TIN &#xc774; &#xc790; &#xccb4; &#xad50; &#xcc28; &#xd558; &#xac70; &#xb000; &#xc790; &#xccb4; &#xc811; &#xcd09; &#xd558; &#xb294; &#xc774; &#xb840; &#xc801; &#xc778; &#xb3c4; &#xd615; &#xd3ec; &#xcac0; &#xc9c0; &#xace0; &#xc788; &#xc9c0; &#xc54a; &#xc744; &#xacbd; &#xc6b0; TRUE &#xb97c; &#xbc18; &#xd658; &#xd569; &#xb2c8; &#xb2e4;.
- **ST\_MakeValid** - Attempts to make an invalid geometry valid without losing vertices.
- **ST\_MemSize** - ST\_Geometry &#xac12; &#xc758; &#xb3c4; &#xd615; &#xc720; &#xd615; &#xc744; &#xbc18; &#xd658; &#xd569; &#xc774; &#xc790; &#xccb4; &#xad50; &#xcc28; &#xd558; &#xac70; &#xb000; &#xc790; &#xccb4; &#xc811; &#xcd09; &#xd558; &#xb294; &#xc774; &#xb840; &#xc801; &#xc778; &#xb3c4; &#xd615; &#xd3ec; &#xcac0; &#xc9c0; &#xace0; &#xc788; &#xc9c0; &#xc54a; &#xc744; &#xacbd; &#xc6b0; TRUE &#xb97c; &#xbc18; &#xd658; &#xd569; &#xb2c8; &#xb2e4;.
- **ST\_MemUnion** - Aggregate function which unions geometries in a memory-efficient but slower way
- **ST\_NDims** - ST\_Geometry &#xac12; &#xc758; &#xc88c; &#xd45c; &#xcc28; &#xc6d0; &#xc744; &#xbc18; &#xd658; &#xd569; &#xc774; &#xc790; &#xccb4; &#xad50; &#xcc28; &#xd558; &#xac70; &#xb000; &#xc790; &#xccb4; &#xc811; &#xcd09; &#xd558; &#xb294; &#xc774; &#xb840; &#xc801; &#xc778; &#xb3c4; &#xd615; &#xd3ec; &#xcac0; &#xc9c0; &#xace0; &#xc788; &#xc9c0; &#xc54a; &#xc744; &#xacbd; &#xc6b0; TRUE &#xb97c; &#xbc18; &#xd658; &#xd569; &#xb2c8; &#xb2e4;.
- **ST\_NPoints** - &#xb3c4; &#xd615; &#xc774; &#xac00; &#xc9c0; &#xace0; &#xc788; &#xb294; &#xd3ec; &#xc778; &#xd2b8; (&#xaf20; &#xc1c; &#xc218; &#xb97c; &#xbc18; &#xd658; &#xd569; &#xb2c8; &#xb2e4;.

- **ST\_NRings** - Returns the number of rings in a geometry.
- **ST\_Node** - Nodes a collection of lines.
- **ST\_NumGeometries** - Returns the number of geometries in a collection.
- **ST\_NumPatches** - Returns the number of patches in a geometry.
- **ST\_Orientation** - Returns the orientation of a geometry.
- **ST\_PatchN** - Returns the Nth patch of a geometry.
- **ST\_PointFromWKB** - Returns a point from a WKB.
- **ST\_PointN** - Returns the Nth point of a geometry.
- **ST\_PointOnSurface** - Computes a point guaranteed to lie in a polygon, or on a geometry.
- **ST\_Points** - Returns the points of a geometry.
- **ST\_Polygon** - Creates a Polygon from a LineString with a specified SRID.
- **ST\_RemovePoint** - Remove a point from a linestring.
- **ST\_RemoveRepeatedPoints** - Returns a version of a geometry with duplicate points removed.
- **ST\_Reverse** - Returns the reverse of a geometry.
- **ST\_Rotate** - Rotates a geometry about an origin point.
- **ST\_RotateX** - Rotates a geometry about the X axis.
- **ST\_RotateY** - Rotates a geometry about the Y axis.
- **ST\_RotateZ** - Rotates a geometry about the Z axis.
- **ST\_Scale** - Scales a geometry by given factors.
- **ST\_Scroll** - Change start point of a closed LineString.
- **ST\_SetPoint** - Sets a point in a geometry.
- **ST\_Shift\_Longitude** - Shifts the longitude coordinates of a geometry between -180..180 and 0..360.
- **ST\_SnapToGrid** - Snaps a geometry to a grid.
- **ST\_StartPoint** - Returns the first point of a LineString.
- **ST\_StraightSkeleton** - Returns the straight skeleton of a geometry.
- **ST\_SwapOrdinates** - Swaps the X and Y ordinates of a geometry.
- **ST\_SymDifference** - Computes a geometry representing the portions of geometries A and B that do not intersect.

- **ST\_Tessellate** - Returns a TIN from a geometry. The TIN is a set of triangles that approximate the geometry. The TIN is a set of triangles that approximate the geometry. The TIN is a set of triangles that approximate the geometry.
- **ST\_TransScale** - Translates and scales a geometry by given offsets and factors.
- **ST\_Translate** - Translates a geometry by given offsets.
- **ST\_UnaryUnion** - Computes the union of the components of a single geometry.
- **ST\_Union** - Computes a geometry representing the point-set union of the input geometries.
- **ST\_Volume** - Returns the volume of a 3D geometry. The volume is a scalar value representing the volume of the geometry.
- **ST\_WrapX** - Wraps the X coordinate of a geometry. The wrapped X coordinate is a scalar value representing the X coordinate of the geometry.
- **ST\_X** - Returns the X coordinate of a Point.
- **ST\_XMax** - Returns the X maxima of a 2D or 3D bounding box or a geometry.
- **ST\_XMin** - Returns the X minima of a 2D or 3D bounding box or a geometry.
- **ST\_Y** - Returns the Y coordinate of a Point.
- **ST\_YMax** - Returns the Y maxima of a 2D or 3D bounding box or a geometry.
- **ST\_YMin** - Returns the Y minima of a 2D or 3D bounding box or a geometry.
- **ST\_Z** - Returns the Z coordinate of a Point.
- **ST\_ZMax** - Returns the Z maxima of a 2D or 3D bounding box or a geometry.
- **ST\_ZMin** - Returns the Z minima of a 2D or 3D bounding box or a geometry.
- **ST\_Zmflag** - Returns the Z maxima of a 2D or 3D bounding box or a geometry.
- **TG\_Equals** - Returns TRUE if two TopoGeometry objects are equal. The TopoGeometry objects are TopoGeometry objects.
- **TG\_Intersects** - Returns TRUE if two TopoGeometry objects intersect. The TopoGeometry objects are TopoGeometry objects.
- **UpdateGeometrySRID** - Updates the SRID of all features in a geometry column, and the table metadata.
- **geometry\_overlaps\_nd** - Returns TRUE if a geometry's (cached) n-D bounding box intersects a n-D float precision bounding box (GIDX).
- **overlaps\_nd\_geometry\_gidx** - Returns TRUE if a geometry's (cached) n-D bounding box intersects a n-D float precision bounding box (GIDX).
- **overlaps\_nd\_gidx\_geometry** - Returns TRUE if a n-D float precision bounding box (GIDX) intersects a geometry's (cached) n-D bounding box.
- **overlaps\_nd\_gidx\_gidx** - Returns TRUE if two n-D float precision bounding boxes (GIDX) intersect each other.
- **postgis\_sfcgal\_full\_version** - Returns the full version of SFCGAL in use including CGAL and Boost versions.
- **postgis\_sfcgal\_version** - Returns the version of SFCGAL in use.

## 15.9 PostGIS Curved Geometry Support Functions

The functions given below are PostGIS functions that can use CIRCULARSTRING, CURVEPOLYGON, and other curved geometry types

- **AddGeometryColumn** - Adds a geometry column to a table.
- **Box2D** - Returns a BOX2D representing the 2D extent of a geometry.
- **Box3D** - Returns a BOX3D representing the 3D extent of a geometry.
- **DropGeometryColumn** - Drops a geometry column from a table.
- **ST\_3DExtent** - Aggregate function that returns the 3D bounding box of geometries.
- **ST\_Affine** - Apply a 3D affine transformation to a geometry.
- **ST\_AsBinary** - Return the OGC/ISO Well-Known Binary (WKB) representation of the geometry/geography without SRID meta data.
- **ST\_AsEWKB** - Return the Extended Well-Known Binary (EWKB) representation of the geometry with SRID meta data.
- **ST\_AsEWKT** - Returns the Well-Known Text (WKT) representation of the geometry with SRID meta data.
- **ST\_AsHEXEWKB** - Returns the Extended Well-Known Binary (EWKB) representation of the geometry with SRID meta data in hexadecimal format.
- **ST\_AsText** - Returns the Well-Known Text (WKT) representation of the geometry without SRID meta data.
- **ST\_GeomCollFromText** - Creates a GeometryCollection or Multi\* geometry from a set of geometries.
- **ST\_CoordDim** - Returns the coordinate dimension of a geometry.
- **ST\_CurveToLine** - Converts a geometry containing curves to a linear geometry.
- **ST\_Distance** - Returns the distance between two geometries.
- **ST\_Dump** - Returns a set of geometry\_dump rows for the components of a geometry.
- **ST\_NumPoints** - Returns the number of points in a geometry.
- **ST\_EndPoint** - Returns the endpoint of a line or line string.
- **ST\_EstimatedExtent** - Returns the estimated extent of a spatial table.
- **ST\_FlipCoordinates** - Returns a version of a geometry with X and Y axis flipped.
- **ST\_Force2D** - Returns a 2D geometry from a 3D geometry.

- **ST\_ForceCurve** - Converts a geometry to a curve.
- **ST\_ForceSFS** - Converts a geometry to a surface.
- **ST\_Force3D** - Converts a 2D geometry to a 3D geometry.
- **ST\_Force3DM** - Converts a 2D geometry to a 3D geometry with M values.
- **ST\_Force3DZ** - Converts a 2D geometry to a 3D geometry with Z values.
- **ST\_Force4D** - Converts a 2D geometry to a 4D geometry with M and Z values.
- **ST\_ForceCollection** - Converts a geometry to a collection.
- **ST\_GeoHash** - Returns a GeoHash for a geometry.
- **ST\_GeogFromWKB** - Converts a Well-Known Binary (WKB) to a geography.
- **ST\_GeomFromEWKB** - Converts an Extended Well-Known Binary (EWKB) to a geometry.
- **ST\_GeomFromEWKT** - Converts an Extended Well-Known Text (EWKT) to a geometry.
- **ST\_GeomFromText** - Converts a Well-Known Text (WKT) to a geometry.
- **ST\_GeomFromWKB** - Converts a Well-Known Binary (WKB) to a geometry.
- **ST\_GeometryN** - Returns the Nth geometry in a collection.
- **=** - Returns TRUE if the coordinates and coordinate order geometry/geography A are the same as the coordinates and coordinate order of geometry/geography B.
- **<** - Returns TRUE if the geometry/geography A is smaller than the geometry/geography B.
- **ST\_HasArc** - Tests if a geometry contains a circular arc.
- **ST\_Intersects** - Tests if two geometries intersect (they have at least one point in common).
- **ST\_IsClosed** - Tests if a LINESTRING is closed.
- **ST\_IsCollection** - Tests if a geometry is a collection.
- **ST\_IsEmpty** - Tests if a geometry is empty.
- **ST\_LineToCurve** - Converts a linear geometry to a curved geometry.
- **ST\_MemSize** - Returns the memory size of a geometry.
- **ST\_NPoints** - Returns the number of points in a geometry.

- **ST\_NRings** - Returns the number of rings in a geometry.
- **ST\_PointFromWKB** - Returns a point from a Well-Known Binary (WKB) representation.
- **ST\_PointN** - Returns the Nth point in a geometry.
- **ST\_Points** - Returns an array of points from a geometry.
- **ST\_Rotate** - Rotates a geometry about an origin point.
- **ST\_RotateZ** - Rotates a geometry about the Z axis.
- **ST\_SRID** - Returns the spatial reference identifier for a geometry.
- **ST\_Scale** - Scales a geometry by given factors.
- **ST\_SetSRID** - Set the SRID on a geometry.
- **ST\_StartPoint** - Returns the first point of a LineString.
- **ST\_Summary** - Returns a summary of a geometry.
- **ST\_SwapOrdinates** - Swaps the X and Y coordinates of a geometry.
- **ST\_TransScale** - Translates and scales a geometry by given offsets and factors.
- **ST\_Transform** - Return a new geometry with coordinates transformed to a different spatial reference system.
- **ST\_Translate** - Translates a geometry by given offsets.
- **ST\_XMax** - Returns the X maxima of a 2D or 3D bounding box or a geometry.
- **ST\_XMin** - Returns the X minima of a 2D or 3D bounding box or a geometry.
- **ST\_YMax** - Returns the Y maxima of a 2D or 3D bounding box or a geometry.
- **ST\_YMin** - Returns the Y minima of a 2D or 3D bounding box or a geometry.
- **ST\_ZMax** - Returns the Z maxima of a 2D or 3D bounding box or a geometry.
- **ST\_ZMin** - Returns the Z minima of a 2D or 3D bounding box or a geometry.
- **ST\_Zmflag** - Returns the Z maxima flag of a geometry.
- **UpdateGeometrySRID** - Updates the SRID of all features in a geometry column, and the table metadata.
- **~(box2df,box2df)** - Returns TRUE if a 2D float precision bounding box (BOX2DF) contains another 2D float precision bounding box (BOX2DF).
- **~(box2df,geometry)** - Returns TRUE if a 2D float precision bounding box (BOX2DF) contains a geometry's 2D bounding box.
- **~(geometry,box2df)** - Returns TRUE if a geometry's 2D bounding box contains a 2D float precision bounding box (BOX2DF).
- **&&** - Returns TRUE if a 2D float precision bounding box (BOX2DF) contains another 2D float precision bounding box (BOX2DF).
- **&&&** - Returns TRUE if a 2D float precision bounding box (BOX2DF) contains a geometry's 2D bounding box.

- `@(box2df,box2df)` - Returns TRUE if a 2D float precision bounding box (BOX2DF) is contained into another 2D float precision bounding box.
- `@(box2df,geometry)` - Returns TRUE if a 2D float precision bounding box (BOX2DF) is contained into a geometry's 2D bounding box.
- `@(geometry,box2df)` - Returns TRUE if a geometry's 2D bounding box is contained into a 2D float precision bounding box (BOX2DF).
- `&&(box2df,box2df)` - Returns TRUE if two 2D float precision bounding boxes (BOX2DF) intersect each other.
- `&&(box2df,geometry)` - Returns TRUE if a 2D float precision bounding box (BOX2DF) intersects a geometry's (cached) 2D bounding box.
- `&&(geometry,box2df)` - Returns TRUE if a geometry's (cached) 2D bounding box intersects a 2D float precision bounding box (BOX2DF).
- `&&&(geometry,gidx)` - Returns TRUE if a geometry's (cached) n-D bounding box intersects a n-D float precision bounding box (GIDX).
- `&&&(gidx,geometry)` - Returns TRUE if a n-D float precision bounding box (GIDX) intersects a geometry's (cached) n-D bounding box.
- `&&&(gidx,gidx)` - Returns TRUE if two n-D float precision bounding boxes (GIDX) intersect each other.

## 15.10 PostGIS Polyhedral Surface Support Functions

The functions given below are PostGIS functions that can use POLYHEDRALSURFACE, POLYHEDRALSURFACEM geometries

- `Box2D` - Returns a BOX2D representing the 2D extent of a geometry.
- `Box3D` - Returns a BOX3D representing the 3D extent of a geometry.
- `&#xb3c4;&#xd615; &#xc720;&#xd615;` - ST\_Geometry &#xac12;&#xc758; &#xb3c4;&#xd615; &#xc720;&#xd615;&#xc744; &#xbc18;&#xd658;&#xd569;&#xb2c8;&#xb2e4;.
- `ST_3DArea` - 3&#xcc28;&#xc6d0; &#xd45c;&#xba74; &#xb3c4;&#xd615;&#xc758; &#xba74;&#xc801;&#xc744; &#xacc4;&#xc785;&#xccb4;&#xc77c; &#xacbd;&#xc6b0; 0&#xc744; &#bc18;&#xd658;&#xd560; &#xac83;&#xc785;&#xb2c8;&#xb2e4;
- `ST_3DClosestPoint` - g2&#xc5d0; &#xac00;&#xc7a5; &#xac00;&#xae4c;&#xc6b4; g1 &#xc0c1;&#xc5d0; &#xc788;&#xb294; 3&#xcc28;&#xc6d0; &#xd3ec;&#xc778;&#xd2b8;&#xb97c; &#bc18;&#xd658;&#xd569;&#xb2c8;&#xb2e4;. &#xd574;&#xb2f9 &#xd3ec;&#xc778;&#xd2b8;&#xb294; 3D &#xcd5c;&#xb2e8; &#xb77c;&#xc778;&#xc758; &#xccab; &#bc88;&#xc9f8; &#xd3ec;&#xc778;&#xd2b8;&#xc785;&#xb2c8;&#xb2e4;.
- `ST_3DConvexHull` - &#xba74; &#xb3c4;&#xd615;&#xc758; &#xadfc;&#xc0ac; &#xc911;&#xc2ec;&#xcd95;&#xc744; &#xacc4;&#xc785;&#xccb4;&#xc77c; &#xacbd;&#xc6b0; 0&#xc744; &#bc18;&#xd658;&#xd560; &#xac83;&#xc785;&#xb2c8;&#xb2e4;
- `ST_3DDFullyWithin` - Tests if two 3D geometries are entirely within a given 3D distance
- `ST_3DDWithin` - Tests if two 3D geometries are within a given 3D distance
- `ST_3DDifference` - 3&#xcc28;&#xc6d0; &#xcc28;&#xc774;&#xb97c; &#xc218;&#xd589;&#xd569;&#xb2c8;&#xb2e4;.
- `ST_3DDistance` - &#xb3c4;&#xd615; &#xc720;&#xd615;&#xc5d0; &#xb300;&#xd574;. &#xb450; &#xb3c4;&#xd615; &#xc0ac;&#xc785;&#xccb4;&#xc77c; &#xacbd;&#xc6b0; &#xb370;&#xce74;&#xb974;&#xd2b8; &#xcd5c;&#xb2e8; &#xac70;&#xb9ac;&#xb97c; &#xd22c;&#xc601; &#xb2e8;&#xc704;&#xb85c; &#bc18;&#xd658;&#xd569;&#xb2c8;&#xb2e4;.
- `ST_3DExtent` - Aggregate function that returns the 3D bounding box of geometries.
- `ST_3DIntersection` - 3&#xcc28;&#xc6d0; &#xad50;&#xcc28;&#xb97c; &#xc218;&#xd589;&#xd569;&#xb2c8;&#xb2e4;.

- **ST\_3DIntersects** - Tests if two geometries spatially intersect in 3D - only for points, linestrings, polygons, polyhedral surface (area).
- **ST\_3DLongestLine** - Returns the longest line segment within a 3D geometry.
- **ST\_3DMaxDistance** - Returns the maximum distance between any two points in a 3D geometry.
- **ST\_3DShortestLine** - Returns the shortest line segment within a 3D geometry.
- **ST\_3DUnion** - Perform 3D union.
- **ST\_Affine** - Apply a 3D affine transformation to a geometry.
- **ST\_ApproximateMedialAxis** - Returns the approximate medial axis of a 3D geometry.
- **ST\_Area** - Returns the area of a 3D geometry.
- **ST\_AsBinary** - Return the OGC/ISO Well-Known Binary (WKB) representation of the geometry/geography without SRID meta data.
- **ST\_AsEWKB** - Return the Extended Well-Known Binary (EWKB) representation of the geometry with SRID meta data.
- **ST\_AsEWKT** - Returns the Well-Known Text (WKT) representation of the geometry with SRID meta data.
- **ST\_AsGML** - Returns the GML 2 or GML 3 representation of the geometry with SRID meta data.
- **ST\_AsX3D** - Returns the X3D XML or ISO-IEC-19776-1.2-X3DEncodings-XML representation of the geometry with SRID meta data.
- **ST\_CoordDim** - Returns the coordinate dimension of a 3D geometry.
- **ST\_Dimension** - Returns the dimension of a 3D geometry.
- **ST\_Dump** - Returns a set of geometry\_dump rows for the components of a geometry.
- **ST\_NumPoints** - Returns the number of points in a 3D geometry.
- **ST\_Expand** - Returns a bounding box expanded from another bounding box or a geometry.
- **ST\_Extent** - Aggregate function that returns the bounding box of geometries.
- **ST\_Extrude** - Returns a 3D geometry extruded from a 2D geometry.
- **ST\_FlipCoordinates** - Returns a version of a geometry with X and Y axis flipped.
- **ST\_Force2D** - Returns a 2D geometry from a 3D geometry.
- **ST\_ForceLHR** - Returns a 3D geometry with Left-Hand Rule orientation.
- **ST\_ForceRHR** - Returns a 3D geometry with Right-Hand Rule orientation.
- **ST\_ForceSFS** - Returns a 3D geometry with SFS 1.1 orientation.
- **ST\_Force3D** - Returns a 3D geometry from a 2D geometry.

- **ST\_Force3DZ** - XYZ
- **ST\_ForceCollection** -
- **ST\_GeomFromEWKB** - EWKB(Extended Well-Known Binary)
- **ST\_GeomFromEWKT** - EWKT(Extended Well-Known Text)
- **ST\_GeomFromGML** - GML
- **ST\_GeometryN** - ST\_Geometry
- **ST\_GeometryType** - ST\_Geometry
- **=** - Returns TRUE if the coordinates and coordinate order geometry/geography A are the same as the coordinates and coordinate order of geometry/geography B.
- **<** - A
- **~** - A
- **ST\_IsClosed** - LINESTRING
- **ST\_IsPlanar** -
- **ST\_IsSolid** -
- **ST\_MakeSolid** -
- **ST\_MemSize** - ST\_Geometry
- **ST\_NPoints** -
- **ST\_NumGeometries** -
- **ST\_NumPatches** -
- **ST\_PatchN** - ST\_Geometry
- **ST\_RemoveRepeatedPoints** - Returns a version of a geometry with duplicate points removed.
- **ST\_Reverse** -
- **ST\_Rotate** - Rotates a geometry about an origin point.

- **ST\_RotateX** - Rotates a geometry about the X axis.
- **ST\_RotateY** - Rotates a geometry about the Y axis.
- **ST\_RotateZ** - Rotates a geometry about the Z axis.
- **ST\_Scale** - Scales a geometry by given factors.
- **ST\_ShiftLongitude** - Shifts the longitude coordinates of a geometry between -180..180 and 0..360.
- **ST\_StraightSkeleton** - (straight skeleton)
- **ST\_Summary** -
- **ST\_SwapOrdinates** -
- **ST\_Tessellate** - (tessellation) TIN
- **ST\_Transform** - Return a new geometry with coordinates transformed to a different spatial reference system.
- **ST\_Volume** -
- **~(box2df,box2df)** - Returns TRUE if a 2D float precision bounding box (BOX2DF) contains another 2D float precision bounding box (BOX2DF).
- **~(box2df,geometry)** - Returns TRUE if a 2D float precision bounding box (BOX2DF) contains a geometry's 2D bounding box.
- **~(geometry,box2df)** - Returns TRUE if a geometry's 2D bounding box contains a 2D float precision bounding box (BOX2DF).
- **&&** - A 2D bounding box contains another 2D bounding box; TRUE
- **&&&** - A n-D bounding box contains another n-D bounding box; TRUE
- **@(box2df,box2df)** - Returns TRUE if a 2D float precision bounding box (BOX2DF) is contained into another 2D float precision bounding box.
- **@(box2df,geometry)** - Returns TRUE if a 2D float precision bounding box (BOX2DF) is contained into a geometry's 2D bounding box.
- **@(geometry,box2df)** - Returns TRUE if a geometry's 2D bounding box is contained into a 2D float precision bounding box (BOX2DF).
- **&&(box2df,box2df)** - Returns TRUE if two 2D float precision bounding boxes (BOX2DF) intersect each other.
- **&&(box2df,geometry)** - Returns TRUE if a 2D float precision bounding box (BOX2DF) intersects a geometry's (cached) 2D bounding box.
- **&&(geometry,box2df)** - Returns TRUE if a geometry's (cached) 2D bounding box intersects a 2D float precision bounding box (BOX2DF).
- **&&&(geometry,gidx)** - Returns TRUE if a geometry's (cached) n-D bounding box intersects a n-D float precision bounding box (GIDX).
- **&&&(gidx,geometry)** - Returns TRUE if a n-D float precision bounding box (GIDX) intersects a geometry's (cached) n-D bounding box.

- `&&(gidx,gidx)` - Returns TRUE if two n-D float precision bounding boxes (GIDX) intersect each other.
- `postgis_sfcgal_version` - Returns the full version of SFCGAL in use including CGAL and Boost versions
- `postgis_sfcgal_version` - `&#xc2e4;&#xd589;&#xc911;&#xc778; SFCGAL&#xc758; &#xbc84;&#xc804;&#xc744; &#xbc18;&#xd6`

## 15.11 PostGIS Function Support Matrix

Below is an alphabetical listing of spatial specific functions in PostGIS and the kinds of spatial types they work with or OGC/SQL compliance they try to conform to.

- A  means the function works with the type or subtype natively.
- A  means it works but with a transform cast built-in using cast to geometry, transform to a "best srid" spatial ref and then cast back. Results may not be as expected for large areas or areas at poles and may accumulate floating point junk.
- A  means the function works with the type because of a auto-cast to another such as to box3d rather than direct type support.
- A  means the function only available if PostGIS compiled with SFCGAL support.
- A  means the function support is provided by SFCGAL if PostGIS compiled with SFCGAL support, otherwise GEOS/built-in support.
- geom - Basic 2D geometry support (x,y).
- geog - Basic 2D geography support (x,y).
- 2.5D - basic 2D geometries in 3 D/4D space (has Z or M coord).
- PS - Polyhedral surfaces
- T - Triangles and Triangulated Irregular Network surfaces (TIN)

| Function                                                      | geom | geog | 2.5D | Curves | SQL MM | PS | T |
|---------------------------------------------------------------|------|------|------|--------|--------|----|---|
| <code>Box2D</code>                                            | ✓    |      |      | ✓      |        | ✓  | ✓ |
| <code>Box3D</code>                                            | ✓    |      | ✓    | ✓      |        | ✓  | ✓ |
| <code>&amp;#xb3c4;&amp;#xd615;&amp;#xc720;&amp;#xd615;</code> | ✓    |      | ✓    | ✓      |        | ✓  | ✓ |
| <code>PostGIS_AddBBox</code>                                  | ✓    |      |      | ✓      |        |    |   |
| <code>PostGIS_DropBBox</code>                                 | ✓    |      |      | ✓      |        |    |   |
| <code>PostGIS_Extensions_Upgrade</code>                       |      |      |      |        |        |    |   |
| <code>PostGIS_Full_Version</code>                             |      |      |      |        |        |    |   |
| <code>PostGIS_GEOS_Version</code>                             |      |      |      |        |        |    |   |
| <code>PostGIS_HasBBox</code>                                  | ✓    |      |      | ✓      |        |    |   |
| <code>PostGIS_LibXML_Version</code>                           |      |      |      |        |        |    |   |
| <code>PostGIS_Lib_Build_Date</code>                           |      |      |      |        |        |    |   |
| <code>PostGIS_Lib_Version</code>                              |      |      |      |        |        |    |   |
| <code>PostGIS_Libwgeom_Version</code>                         |      |      |      |        |        |    |   |
| <code>PostGIS_PROJ_Version</code>                             |      |      |      |        |        |    |   |

| Function                   | geom | geog | 2.5D | Curves | SQL MM | PS | T |
|----------------------------|------|------|------|--------|--------|----|---|
| PostGIS_Scripts_Build_Date |      |      |      |        |        |    |   |
| PostGIS_Scripts_Installed  |      |      |      |        |        |    |   |
| PostGIS_Scripts_Released   |      |      |      |        |        |    |   |
| PostGIS_Version            |      |      |      |        |        |    |   |
| PostGIS_Wagyu_Version      |      |      |      |        |        |    |   |
| ST_3DArea                  |      |      |      |        |        |    |   |
| ST_3DClosestPoint          | ✓    |      | ✓    |        |        | ✓  |   |
| ST_3DConvexHull            |      |      |      |        |        |    |   |
| ST_3DDifference            |      |      |      |        |        |    |   |
| ST_3DDistance              | ✓    |      | ✓    |        | ✓      | ✓  |   |
| ST_3DExtent                | ✓    |      | ✓    | ✓      |        | ✓  | ✓ |
| ST_3DIntersection          |      |      |      |        |        |    |   |
| ST_3DLength                | ✓    |      | ✓    |        | ✓      |    |   |
| ST_LineInterpolate         | ✓    |      | ✓    |        |        |    |   |
| ST_3DLongestLine           | ✓    |      | ✓    |        |        | ✓  |   |
| ST_3DMakeBox               | ✓    |      |      |        |        |    |   |
| ST_3DMaxDistance           | ✓    |      | ✓    |        |        | ✓  |   |
| ST_3DPerimeter             | ✓    |      | ✓    |        | ✓      |    |   |
| ST_3DShortestLine          | ✓    |      | ✓    |        |        | ✓  |   |
| ST_3DUnion                 |      |      |      |        |        |    |   |
| ST_AddMeasure              | ✓    |      | ✓    |        |        |    |   |
| ST_AddPoint                | ✓    |      | ✓    |        |        |    |   |
| ST_Affine                  | ✓    |      | ✓    | ✓      |        | ✓  | ✓ |
| ST_AlphaShape              |      |      |      |        |        |    |   |
| ST_Angle                   | ✓    |      |      |        |        |    |   |
| ST_ApproximateM            | Axis |      |      |        |        |    |   |
| ST_Area                    | ✓    | ✓    |      |        | ✓      | ✓  |   |
| ST_Azimuth                 | ✓    | ✓    |      |        |        |    |   |
| ST_Boundary                | ✓    |      | ✓    |        | ✓      |    |   |
| ST_BoundingDiagonal        | ✓    |      | ✓    |        |        |    |   |
| ST_Buffer                  | ✓    | ✓    |      |        | ✓      |    |   |
| ST_BuildArea               | ✓    |      |      |        |        |    |   |
| ST_CPAWithin               | ✓    |      | ✓    |        |        |    |   |

| Function                        | geom                                                                                | geog  | 2.5D                                                                              | Curves | SQL MM | PS | T                                                                                   |
|---------------------------------|-------------------------------------------------------------------------------------|-------|-----------------------------------------------------------------------------------|--------|--------|----|-------------------------------------------------------------------------------------|
| ST_Centroid                     | ✓                                                                                   | ✓     |                                                                                   |        | ✓      |    |                                                                                     |
| ST_ChaikinSmooth                | ✓                                                                                   |       |                                                                                   |        |        |    |                                                                                     |
| ST_ClipByBox2D                  | ✓                                                                                   |       |                                                                                   |        |        |    |                                                                                     |
| ST_ClosestPoint                 | ✓                                                                                   |       |                                                                                   |        |        |    |                                                                                     |
| ST_ClosestPointOf               | ✓                                                                                   | reach | ✓                                                                                 |        |        |    |                                                                                     |
| ST_ClusterDBSCAN                | ✓                                                                                   |       |                                                                                   |        |        |    |                                                                                     |
| ST_ClusterIntersect             | ✓                                                                                   |       |                                                                                   |        |        |    |                                                                                     |
| ST_ClusterKMeans                | ✓                                                                                   |       |                                                                                   |        |        |    |                                                                                     |
| ST_ClusterWithin                | ✓                                                                                   |       |                                                                                   |        |        |    |                                                                                     |
| ST_GeomCollFrom                 | ✓                                                                                   |       | ✓                                                                                 | ✓      |        |    |                                                                                     |
| ST_CollectionExtract            | ✓                                                                                   |       |                                                                                   |        |        |    |                                                                                     |
| ST_CollectionHomogenize         | ✓                                                                                   | size  |                                                                                   |        |        |    |                                                                                     |
| ST_ConcaveHull                  | ✓                                                                                   |       |                                                                                   |        |        |    | ✓                                                                                   |
| ST_ConstrainedDelaunayTriangles |    |       |  |        |        |    |  |
| ST_ConvexHull                   | ✓                                                                                   |       | ✓                                                                                 |        | ✓      |    |                                                                                     |
| ST_CoordDim                     | ✓                                                                                   |       | ✓                                                                                 | ✓      | ✓      | ✓  | ✓                                                                                   |
| ST_CurveToLine                  | ✓                                                                                   |       | ✓                                                                                 | ✓      | ✓      |    |                                                                                     |
| ST_DelaunayTriang               | ✓                                                                                   |       | ✓                                                                                 |        |        |    | ✓                                                                                   |
| ST_Difference                   | ✓                                                                                   |       | ✓                                                                                 |        | ✓      |    |                                                                                     |
| ST_Dimension                    | ✓                                                                                   |       |                                                                                   |        | ✓      | ✓  | ✓                                                                                   |
| ST_Distance                     | ✓                                                                                   |       |                                                                                   | ✓      | ✓      |    |                                                                                     |
| ST_DistanceCPA                  | ✓                                                                                   |       | ✓                                                                                 |        |        |    |                                                                                     |
| ST_DistanceSphere               | ✓                                                                                   |       |                                                                                   |        |        |    |                                                                                     |
| ST_DistanceSpheroid             | ✓                                                                                   |       |                                                                                   |        |        |    |                                                                                     |
| ST_Dump                         | ✓                                                                                   |       | ✓                                                                                 | ✓      |        | ✓  | ✓                                                                                   |
| ST_NumPoints                    | ✓                                                                                   |       | ✓                                                                                 | ✓      |        | ✓  | ✓                                                                                   |
| ST_NRings                       | ✓                                                                                   |       | ✓                                                                                 |        |        |    |                                                                                     |
| ST_NumPoints                    | ✓                                                                                   |       | ✓                                                                                 |        |        |    | ✓                                                                                   |
| ST_EndPoint                     | ✓                                                                                   |       | ✓                                                                                 | ✓      | ✓      |    |                                                                                     |
| ST_Envelope                     | ✓                                                                                   |       |                                                                                   |        | ✓      |    |                                                                                     |
| ST_EstimatedExtent              |  |       |                                                                                   | ✓      |        |    |                                                                                     |
| ST_Expand                       | ✓                                                                                   |       |                                                                                   |        |        | ✓  | ✓                                                                                   |
| ST_Extent                       | ✓                                                                                   |       |                                                                                   |        |        | ✓  | ✓                                                                                   |

| Function            | geom | geog | 2.5D | Curves | SQL MM | PS | T |
|---------------------|------|------|------|--------|--------|----|---|
| ST_ExteriorRing     | ✓    |      | ✓    |        | ✓      |    |   |
| ST_Extrude          |      |      |      |        |        |    |   |
| ST_FilterByM        | ✓    |      |      |        |        |    |   |
| ST_FlipCoordinates  | ✓    |      | ✓    | ✓      |        | ✓  | ✓ |
| ST_Force2D          | ✓    |      | ✓    | ✓      |        | ✓  |   |
| ST_ForceCurve       | ✓    |      | ✓    | ✓      |        |    |   |
| ST_ForceLHR         |      |      |      |        |        |    |   |
| ST_ForcePolygonC    | ✓    |      | ✓    |        |        |    |   |
| ST_ForcePolygonC    | ✓    |      | ✓    |        |        |    |   |
| ST_ForceRHR         | ✓    |      | ✓    |        |        | ✓  |   |
| ST_ForceSFS         | ✓    |      | ✓    | ✓      |        | ✓  | ✓ |
| ST_Force3D          | ✓    |      | ✓    | ✓      |        | ✓  |   |
| ST_Force3DM         | ✓    |      |      | ✓      |        |    |   |
| ST_Force3DZ         | ✓    |      | ✓    | ✓      |        | ✓  |   |
| ST_Force4D          | ✓    |      | ✓    | ✓      |        |    |   |
| ST_ForceCollection  | ✓    |      | ✓    | ✓      |        | ✓  |   |
| ST_FrechetDistance  | ✓    |      |      |        |        |    |   |
| ST_GeneratePoints   | ✓    |      |      |        |        |    |   |
| ST_GeometricMedi    | ✓    |      | ✓    |        |        |    |   |
| ST_GeometryN        | ✓    |      | ✓    | ✓      | ✓      | ✓  | ✓ |
| ST_GeometryType     | ✓    |      | ✓    |        | ✓      | ✓  |   |
| ST_HasArc           | ✓    |      | ✓    | ✓      |        |    |   |
| ST_HausdorffDista   | ✓    |      |      |        |        |    |   |
| ST_Hexagon          | ✓    |      |      |        |        |    |   |
| ST_HexagonGrid      | ✓    |      |      |        |        |    |   |
| ST_InteriorRingN    | ✓    |      | ✓    |        | ✓      |    |   |
| ST_InterpolatePoint | ✓    |      | ✓    |        |        |    |   |
| ST_Intersection     | ✓    | ✓    | ✓    |        | ✓      |    |   |
| ST_IsClosed         | ✓    |      | ✓    | ✓      | ✓      | ✓  |   |
| ST_IsCollection     | ✓    |      | ✓    | ✓      |        |    |   |
| ST_IsEmpty          | ✓    |      |      | ✓      | ✓      |    |   |
| ST_IsPlanar         |      |      |      |        |        |    |   |
| ST_IsPolygonCCW     | ✓    |      | ✓    |        |        |    |   |

| Function                 | geom                                                                                | geog | 2.5D                                                                                | Curves | SQL MM                                                                              | PS                                                                                    | T                                                                                     |
|--------------------------|-------------------------------------------------------------------------------------|------|-------------------------------------------------------------------------------------|--------|-------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------|
| ST_IsPolygonCW           | ✓                                                                                   |      | ✓                                                                                   |        |                                                                                     |                                                                                       |                                                                                       |
| ST_IsRing                | ✓                                                                                   |      |                                                                                     |        | ✓                                                                                   |                                                                                       |                                                                                       |
| ST_IsSimple              | ✓                                                                                   |      | ✓                                                                                   |        | ✓                                                                                   |                                                                                       |                                                                                       |
| ST_IsSolid               |    |      |    |        |                                                                                     |    |    |
| ST_IsValid               | ✓                                                                                   |      |                                                                                     |        | ✓                                                                                   |                                                                                       |                                                                                       |
| ST_IsValidDetail         | ✓                                                                                   |      |                                                                                     |        |                                                                                     |                                                                                       |                                                                                       |
| ST_IsValidReason         | ✓                                                                                   |      |                                                                                     |        |                                                                                     |                                                                                       |                                                                                       |
| ST_IsValidTrajectory     | ✓                                                                                   |      | ✓                                                                                   |        |                                                                                     |                                                                                       |                                                                                       |
| ST_Length                | ✓                                                                                   | ✓    |                                                                                     |        |  |                                                                                       |                                                                                       |
| ST_Length2D              | ✓                                                                                   |      |                                                                                     |        |                                                                                     |                                                                                       |                                                                                       |
| ST_LengthSpheroid        | ✓                                                                                   |      | ✓                                                                                   |        |                                                                                     |                                                                                       |                                                                                       |
| ST_Letters               | ✓                                                                                   |      |                                                                                     |        |                                                                                     |                                                                                       |                                                                                       |
| ST_LineFromMultiPoint    | ✓ t                                                                                 |      | ✓                                                                                   |        |                                                                                     |                                                                                       |                                                                                       |
| ST_LineInterpolatePoint  | ✓ t                                                                                 |      | ✓                                                                                   |        |                                                                                     |                                                                                       |                                                                                       |
| ST_LineInterpolatePoints | ✓ ts                                                                                |      | ✓                                                                                   |        |                                                                                     |                                                                                       |                                                                                       |
| ST_LineLocatePoint       | ✓                                                                                   |      |                                                                                     |        |                                                                                     |                                                                                       |                                                                                       |
| ST_LineMerge             | ✓                                                                                   |      |                                                                                     |        |                                                                                     |                                                                                       |                                                                                       |
| ST_LineSubstring         | ✓                                                                                   |      | ✓                                                                                   |        |                                                                                     |                                                                                       |                                                                                       |
| ST_LineToCurve           | ✓                                                                                   |      | ✓                                                                                   | ✓      |                                                                                     |                                                                                       |                                                                                       |
| ST_LocateAlong           | ✓                                                                                   |      |                                                                                     |        | ✓                                                                                   |                                                                                       |                                                                                       |
| ST_LocateBetween         | ✓                                                                                   |      |                                                                                     |        | ✓                                                                                   |                                                                                       |                                                                                       |
| ST_LocateBetweenPoints   | ✓ ations                                                                            |      | ✓                                                                                   |        |                                                                                     |                                                                                       |                                                                                       |
| ST_LongestLine           | ✓                                                                                   |      |                                                                                     |        |                                                                                     |                                                                                       |                                                                                       |
| ST_M                     | ✓                                                                                   |      | ✓                                                                                   |        | ✓                                                                                   |                                                                                       |                                                                                       |
| ST_MakeBox2D             | ✓                                                                                   |      |                                                                                     |        |                                                                                     |                                                                                       |                                                                                       |
| ST_MakeEnvelope          | ✓                                                                                   |      |                                                                                     |        |                                                                                     |                                                                                       |                                                                                       |
| ST_MakeLine              | ✓                                                                                   |      | ✓                                                                                   |        |                                                                                     |                                                                                       |                                                                                       |
| ST_MakePoint             | ✓                                                                                   |      | ✓                                                                                   |        |                                                                                     |                                                                                       |                                                                                       |
| ST_MakePointM            | ✓                                                                                   |      |                                                                                     |        |                                                                                     |                                                                                       |                                                                                       |
| ST_MakePolygon           | ✓                                                                                   |      | ✓                                                                                   |        |                                                                                     |                                                                                       |                                                                                       |
| ST_MakeSolid             |  |      |  |        |                                                                                     |  |  |
| ST_MakeValid             | ✓                                                                                   |      | ✓                                                                                   |        |                                                                                     |                                                                                       |                                                                                       |
| ST_MaxDistance           | ✓                                                                                   |      |                                                                                     |        |                                                                                     |                                                                                       |                                                                                       |

| Function          | geom                                                                                | geog | 2.5D                                                                                | Curves | SQL MM | PS | T |
|-------------------|-------------------------------------------------------------------------------------|------|-------------------------------------------------------------------------------------|--------|--------|----|---|
| ST_MaximumInscri  | ✓ Circle                                                                            |      |                                                                                     |        |        |    |   |
| ST_MemSize        | ✓                                                                                   |      | ✓                                                                                   | ✓      |        | ✓  | ✓ |
| ST_MemUnion       | ✓                                                                                   |      | ✓                                                                                   |        |        |    |   |
| ST_MinimumBound   | ✓ Circle                                                                            |      |                                                                                     |        |        |    |   |
| ST_MinimumBound   | ✓ Radius                                                                            |      |                                                                                     |        |        |    |   |
| ST_MinimumClear   | ✓                                                                                   |      |                                                                                     |        |        |    |   |
| ST_MinimumClear   | ✓ Line                                                                              |      |                                                                                     |        |        |    |   |
| ST_MinkowskiSum   |    |      |                                                                                     |        |        |    |   |
| ST_Multi          | ✓                                                                                   |      |                                                                                     |        |        |    |   |
| ST_NDims          | ✓                                                                                   |      | ✓                                                                                   |        |        |    |   |
| ST_NPoints        | ✓                                                                                   |      | ✓                                                                                   | ✓      |        | ✓  |   |
| ST_NRings         | ✓                                                                                   |      | ✓                                                                                   | ✓      |        |    |   |
| ST_Node           | ✓                                                                                   |      | ✓                                                                                   |        |        |    |   |
| ST_Normalize      | ✓                                                                                   |      |                                                                                     |        |        |    |   |
| ST_NumGeometrie   | ✓                                                                                   |      | ✓                                                                                   |        | ✓      | ✓  | ✓ |
| ST_NumInteriorRin | ✓                                                                                   |      |                                                                                     |        |        |    |   |
| ST_NumInteriorRin | ✓                                                                                   |      |                                                                                     |        | ✓      |    |   |
| ST_NumPatches     | ✓                                                                                   |      | ✓                                                                                   |        | ✓      | ✓  |   |
| ST_NumPoints      | ✓                                                                                   |      |                                                                                     |        | ✓      |    |   |
| ST_OffsetCurve    | ✓                                                                                   |      |                                                                                     |        |        |    |   |
| ST_OptimalAlphaS  |  |      |                                                                                     |        |        |    |   |
| ST_Orientation    |  |      |  |        |        |    |   |
| ST_OrientedEnvelo | ✓                                                                                   |      |                                                                                     |        |        |    |   |
| ST_PatchN         | ✓                                                                                   |      | ✓                                                                                   |        | ✓      | ✓  |   |
| ST_Perimeter      | ✓                                                                                   | ✓    |                                                                                     |        | ✓      |    |   |
| ST_Perimeter2D    | ✓                                                                                   |      |                                                                                     |        |        |    |   |
| ST_Point          | ✓                                                                                   |      |                                                                                     |        | ✓      |    |   |
| ST_Point          | ✓                                                                                   |      |                                                                                     |        |        |    |   |
| ST_PointN         | ✓                                                                                   |      | ✓                                                                                   | ✓      | ✓      |    |   |
| ST_PointOnSurface | ✓                                                                                   |      | ✓                                                                                   |        | ✓      |    |   |
| ST_Point          | ✓                                                                                   |      |                                                                                     |        |        |    |   |
| ST_Point          | ✓                                                                                   |      |                                                                                     |        |        |    |   |
| ST_Points         | ✓                                                                                   |      | ✓                                                                                   | ✓      |        |    |   |

| Function                    | geom                                                                                | geog | 2.5D                                                                                | Curves | SQL MM | PS                                                                                    | T                                                                                     |
|-----------------------------|-------------------------------------------------------------------------------------|------|-------------------------------------------------------------------------------------|--------|--------|---------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------|
| ST_Polygon                  | ✓                                                                                   |      | ✓                                                                                   |        | ✓      |                                                                                       |                                                                                       |
| ST_Polygonize               | ✓                                                                                   |      |                                                                                     |        |        |                                                                                       |                                                                                       |
| ST_Project                  |                                                                                     | ✓    |                                                                                     |        |        |                                                                                       |                                                                                       |
| ST_QuantizeCoordinates      | ✓                                                                                   |      |                                                                                     |        |        |                                                                                       |                                                                                       |
| ST_ReducePrecision          | ✓                                                                                   |      |                                                                                     |        |        |                                                                                       |                                                                                       |
| ST_RemovePoint              | ✓                                                                                   |      | ✓                                                                                   |        |        |                                                                                       |                                                                                       |
| ST_RemoveRepeatedPoints     | ✓                                                                                   |      | ✓                                                                                   |        |        | ✓                                                                                     |                                                                                       |
| ST_Reverse                  | ✓                                                                                   |      | ✓                                                                                   |        |        | ✓                                                                                     |                                                                                       |
| ST_Rotate                   | ✓                                                                                   |      | ✓                                                                                   | ✓      |        | ✓                                                                                     | ✓                                                                                     |
| ST_RotateX                  | ✓                                                                                   |      | ✓                                                                                   |        |        | ✓                                                                                     | ✓                                                                                     |
| ST_RotateY                  | ✓                                                                                   |      | ✓                                                                                   |        |        | ✓                                                                                     | ✓                                                                                     |
| ST_RotateZ                  | ✓                                                                                   |      | ✓                                                                                   | ✓      |        | ✓                                                                                     | ✓                                                                                     |
| ST_SRID                     | ✓                                                                                   |      |                                                                                     | ✓      | ✓      |                                                                                       |                                                                                       |
| ST_Scale                    | ✓                                                                                   |      | ✓                                                                                   | ✓      |        | ✓                                                                                     | ✓                                                                                     |
| ST_Scroll                   | ✓                                                                                   |      | ✓                                                                                   |        |        |                                                                                       |                                                                                       |
| ST_Segmentize               | ✓                                                                                   | ✓    |                                                                                     |        |        |                                                                                       |                                                                                       |
| ST_SetEffectiveArea         | ✓                                                                                   |      |                                                                                     |        |        |                                                                                       |                                                                                       |
| ST_SetPoint                 | ✓                                                                                   |      | ✓                                                                                   |        |        |                                                                                       |                                                                                       |
| ST_SetSRID                  | ✓                                                                                   |      |                                                                                     | ✓      |        |                                                                                       |                                                                                       |
| ST_SharedPaths              | ✓                                                                                   |      |                                                                                     |        |        |                                                                                       |                                                                                       |
| ST_ShiftLongitude           | ✓                                                                                   |      | ✓                                                                                   |        |        | ✓                                                                                     | ✓                                                                                     |
| ST_ShortestLine             | ✓                                                                                   |      |                                                                                     |        |        |                                                                                       |                                                                                       |
| ST_Simplify                 | ✓                                                                                   |      |                                                                                     |        |        |                                                                                       |                                                                                       |
| ST_SimplifyPolygon          | ✓                                                                                   |      |                                                                                     |        |        |                                                                                       |                                                                                       |
| ST_SimplifyPreserveTopology | ✓                                                                                   |      |                                                                                     |        |        |                                                                                       |                                                                                       |
| ST_SimplifyVW               | ✓                                                                                   |      |                                                                                     |        |        |                                                                                       |                                                                                       |
| ST_Snap                     | ✓                                                                                   |      |                                                                                     |        |        |                                                                                       |                                                                                       |
| ST_SnapToGrid               | ✓                                                                                   |      | ✓                                                                                   |        |        |                                                                                       |                                                                                       |
| ST_Split                    | ✓                                                                                   |      |                                                                                     |        |        |                                                                                       |                                                                                       |
| ST_Square                   | ✓                                                                                   |      |                                                                                     |        |        |                                                                                       |                                                                                       |
| ST_SquareGrid               | ✓                                                                                   |      |                                                                                     |        |        |                                                                                       |                                                                                       |
| ST_StartPoint               | ✓                                                                                   |      | ✓                                                                                   | ✓      | ✓      |                                                                                       |                                                                                       |
| ST_StraightSkeleton         |  |      |  |        |        |  |  |

| Function                     | geom                                                                                | geog | 2.5D                                                                                | Curves | SQL MM                                                                              | PS                                                                                    | T                                                                                     |
|------------------------------|-------------------------------------------------------------------------------------|------|-------------------------------------------------------------------------------------|--------|-------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------|
| ST_Subdivide                 | ✓                                                                                   |      |                                                                                     |        |                                                                                     |                                                                                       |                                                                                       |
| ST_Summary                   | ✓                                                                                   | ✓    |                                                                                     | ✓      |                                                                                     | ✓                                                                                     | ✓                                                                                     |
| ST_SwapOrdinates             | ✓                                                                                   |      | ✓                                                                                   | ✓      |                                                                                     | ✓                                                                                     | ✓                                                                                     |
| ST_SymDifference             | ✓                                                                                   |      | ✓                                                                                   |        | ✓                                                                                   |                                                                                       |                                                                                       |
| ST_Tessellate                |    |      |    |        |                                                                                     |    |    |
| ST_MakeEnvelope              | ✓                                                                                   |      |                                                                                     |        |                                                                                     |                                                                                       |                                                                                       |
| ST_TransScale                | ✓                                                                                   |      | ✓                                                                                   | ✓      |                                                                                     |                                                                                       |                                                                                       |
| ST_Transform                 | ✓                                                                                   |      |                                                                                     | ✓      | ✓                                                                                   | ✓                                                                                     |                                                                                       |
| ST_Translate                 | ✓                                                                                   |      | ✓                                                                                   | ✓      |                                                                                     |                                                                                       |                                                                                       |
| ST_TriangulatePoly           | ✓                                                                                   |      |                                                                                     |        |                                                                                     |                                                                                       | ✓                                                                                     |
| ST_UnaryUnion                | ✓                                                                                   |      | ✓                                                                                   |        |                                                                                     |                                                                                       |                                                                                       |
| ST_Union                     | ✓                                                                                   |      | ✓                                                                                   |        | ✓                                                                                   |                                                                                       |                                                                                       |
| ST_Volume                    |    |      |    |        |  |    |    |
| ST_VoronoiLines              | ✓                                                                                   |      |                                                                                     |        |                                                                                     |                                                                                       |                                                                                       |
| ST_VoronoiPolygor            | ✓                                                                                   |      |                                                                                     |        |                                                                                     |                                                                                       |                                                                                       |
| ST_WrapX                     | ✓                                                                                   |      | ✓                                                                                   |        |                                                                                     |                                                                                       |                                                                                       |
| ST_X                         | ✓                                                                                   |      | ✓                                                                                   |        | ✓                                                                                   |                                                                                       |                                                                                       |
| ST_XMax                      |  |      | ✓                                                                                   | ✓      |                                                                                     |                                                                                       |                                                                                       |
| ST_XMin                      |  |      | ✓                                                                                   | ✓      |                                                                                     |                                                                                       |                                                                                       |
| ST_Y                         | ✓                                                                                   |      | ✓                                                                                   |        | ✓                                                                                   |                                                                                       |                                                                                       |
| ST_YMax                      |  |      | ✓                                                                                   | ✓      |                                                                                     |                                                                                       |                                                                                       |
| ST_YMin                      |  |      | ✓                                                                                   | ✓      |                                                                                     |                                                                                       |                                                                                       |
| ST_Z                         | ✓                                                                                   |      | ✓                                                                                   |        | ✓                                                                                   |                                                                                       |                                                                                       |
| ST_ZMax                      |  |      | ✓                                                                                   | ✓      |                                                                                     |                                                                                       |                                                                                       |
| ST_ZMin                      |  |      | ✓                                                                                   | ✓      |                                                                                     |                                                                                       |                                                                                       |
| ST_Zmflag                    | ✓                                                                                   |      | ✓                                                                                   | ✓      |                                                                                     |                                                                                       |                                                                                       |
| postgis.backend              |                                                                                     |      |                                                                                     |        |                                                                                     |                                                                                       |                                                                                       |
| postgis.enable_outdb_rasters |                                                                                     |      |                                                                                     |        |                                                                                     |                                                                                       |                                                                                       |
| postgis.gdal_datapath        |                                                                                     |      |                                                                                     |        |                                                                                     |                                                                                       |                                                                                       |
| postgis.gdal_enabled_drivers |                                                                                     |      |                                                                                     |        |                                                                                     |                                                                                       |                                                                                       |
| postgis.gdal_vsi_options     |                                                                                     |      |                                                                                     |        |                                                                                     |                                                                                       |                                                                                       |
| postgis_sfcgal_version       |                                                                                     |      |  |        |                                                                                     |  |  |
| postgis_sfcgal_version       |                                                                                     |      |  |        |                                                                                     |  |  |

## 15.12 New, Enhanced or changed PostGIS Functions

### 15.12.1 PostGIS Functions new or enhanced in 3.3

The functions given below are PostGIS functions that were added or enhanced.

Functions new in PostGIS 3.3

- **RemoveUnusedPrimitives** - Availability: 3.3.0 Removes topology primitives which not needed to define existing TopoGeometry objects.
- **ST\_3DUnion** - Availability: 3.3.0 aggregate variant was added Perform 3D union.
- **ST\_AlphaShape** - Availability: 3.3.0 - requires SFCGAL >= 1.4.1. Computes a possible concave geometry using the CGAL Alpha Shapes algorithm.
- **ST\_AsMARC21** - Availability: 3.3.0 Returns geometry as a MARC21/XML record with a geographic datafield (034).
- **ST\_GeomFromMARC21** - Availability: 3.3.0, requires libxml2 2.6+ Takes MARC21/XML geographic data as input and returns a PostGIS geometry object.
- **ST\_OptimalAlphaShape** - Availability: 3.3.0 - requires SFCGAL >= 1.4.1. Computes a possible concave geometry using the CGAL Alpha Shapes algorithm after have computed the "optimal" alpha value.
- **ST\_SimplifyPolygonHull** - Availability: 3.3.0 - requires GEOS >= 3.11.0 Computes a simplified topology-preserving outer or inner hull of a polygonal geometry.
- **ST\_TriangulatePolygon** - Availability: 3.3.0 - requires GEOS >= 3.11.0 Computes the constrained Delaunay triangulation of polygons

Functions enhanced in PostGIS 3.3

- **ST\_ConcaveHull** - Enhanced: 3.3.0, GEOS native implementation enabled for GEOS 3.11+ Computes a possibly concave geometry that encloses all input geometry vertices
- **ST\_LineMerge** - Enhanced: 3.3.0 accept a directed parameter - requires GEOS >= 3.11.0 Return the lines formed by sewing together a MultiLineString.

Functions changed in PostGIS 3.3

- **PostGIS\_Extensions\_Upgrade** - Changed: 3.3.0 support for upgrades from any PostGIS version. Does not work on all systems. Packages and upgrades PostGIS extensions (e.g. postgis\_raster, postgis\_topology, postgis\_sfcgal) to latest available version.

### 15.12.2 PostGIS Functions new or enhanced in 3.2

The functions given below are PostGIS functions that were added or enhanced.

Functions new in PostGIS 3.2

- **FindLayer** - Availability: 3.2.0 Returns a topology.layer record by different means.
  - **FindTopology** - Availability: 3.2.0 Returns a topology record by different means.
  - **GetFaceContainingPoint** - Availability: 3.2.0 Finds the face containing a point.
  - **ST\_AsFlatGeobuf** - Availability: 3.2.0 Return a FlatGeobuf representation of a set of rows.
  - **ST\_FromFlatGeobuf** - Availability: 3.2.0 Reads FlatGeobuf data.
  - **ST\_FromFlatGeobufToTable** - Availability: 3.2.0 Creates a table based on the structure of FlatGeobuf data.
-

- **ST\_SRID** - Availability: 3.2.0 Returns the spatial reference identifier for a topogeometry.
- **ST\_Scroll** - Availability: 3.2.0 Change start point of a closed LineString.
- **TopoGeom\_addTopoGeom** - Availability: 3.2 Adds element of a TopoGeometry to the definition of another TopoGeometry.
- **ValidateTopologyRelation** - Availability: 3.2.0 Returns info about invalid topology relation records

#### Functions enhanced in PostGIS 3.2

- **GetFaceByPoint** - Enhanced: 3.2.0 more efficient implementation and clearer contract, stops working with invalid topologies. Finds face intersecting a given point.
- **ST\_ClusterKMeans** - Enhanced: 3.2.0 Support for max\_radius Window function that returns a cluster id for each input geometry using the K-means algorithm.
- **ST\_MakeValid** - Enhanced: 3.2.0, added algorithm options, 'linework' and 'structure' which requires GEOS >= 3.10.0. Attempts to make an invalid geometry valid without losing vertices.
- **ST\_MoveIsoNode** - Enhanced: 3.2.0 ensures the nod cannot be moved in a different face Moves an isolated node in a topology from one point to another. If new apoint geometry exists as a node an error is thrown. Returns description of move.
- **ST\_Point** - Enhanced: 3.2.0 srid as an extra optional argument was added. Older installs require combining with ST\_SetSRID to mark the srid on the geometry. Creates a Point with X, Y and SRID values.
- **ST\_Point** - Enhanced: 3.2.0 srid as an extra optional argument was added. Older installs require combining with ST\_SetSRID to mark the srid on the geometry. Creates a Point with X, Y, Z and SRID values.
- **ST\_Point** - Enhanced: 3.2.0 srid as an extra optional argument was added. Older installs require combining with ST\_SetSRID to mark the srid on the geometry. Creates a Point with X, Y, M and SRID values.
- **ST\_Point** - Enhanced: 3.2.0 srid as an extra optional argument was added. Older installs require combining with ST\_SetSRID to mark the srid on the geometry. Creates a Point with X, Y, Z, M and SRID values.
- **ST\_RemovePoint** - Enhanced: 3.2.0 Remove a point from a linestring.
- **ST\_RemoveRepeatedPoints** - Enhanced: 3.2.0 Returns a version of a geometry with duplicate points removed.
- **ST\_StartPoint** - Enhanced: 3.2.0 returns a point for all geometries. Prior behavior returns NULLs if input was not a LineString. Returns the first point of a LineString.

#### Functions changed in PostGIS 3.2

- **ST\_Boundary** - Changed: 3.2.0 support for TIN, does not use geos, does not linearize curves
- **ValidateTopology** - Changed: 3.2.0 added optional bbox parameter, perform face labeling and edge linking checks. Returns a set of validate\_topology\_returntype objects detailing issues with topology.

### 15.12.3 PostGIS Functions new or enhanced in 3.1

The functions given below are PostGIS functions that were added or enhanced.

#### Functions new in PostGIS 3.1

- **ST\_MaximumInscribedCircle** - Availability: 3.1.0 - requires GEOS >= 3.9.0.
- **ST\_ReducePrecision** - Availability: 3.1.0 - requires GEOS >= 3.9.0. Returns a valid geometry with points rounded to a grid tolerance.

### Functions enhanced in PostGIS 3.1

- **ST\_AsEWKT** - Enhanced: 3.1.0 support for optional precision parameter. WKT(Well-Known Text) SRID
- **ST\_ClusterKMeans** - Enhanced: 3.1.0 Support for 3D geometries and weights Window function that returns a cluster id for each input geometry using the K-means algorithm.
- **ST\_Difference** - Enhanced: 3.1.0 accept a gridSize parameter - requires GEOS >= 3.9.0 Computes a geometry representing the part of geometry A that does not intersect geometry B.
- **ST\_Intersection** - Enhanced: 3.1.0 accept a gridSize parameter - requires GEOS >= 3.9.0 Computes a geometry representing the shared portion of geometries A and B.
- **ST\_MakeValid** - Enhanced: 3.1.0, added removal of Coordinates with NaN values. Attempts to make an invalid geometry valid without losing vertices.
- **ST\_Subdivide** - Enhanced: 3.1.0 accept a gridSize parameter, requires GEOS >= 3.9.0 to use this new feature. Computes a rectilinear subdivision of a geometry.
- **ST\_SymDifference** - Enhanced: 3.1.0 accept a gridSize parameter - requires GEOS >= 3.9.0 Computes a geometry representing the portions of geometries A and B that do not intersect.
- **ST\_UnaryUnion** - Enhanced: 3.1.0 accept a gridSize parameter - requires GEOS >= 3.9.0 Computes the union of the components of a single geometry.
- **ST\_Union** - Enhanced: 3.1.0 accept a gridSize parameter - requires GEOS >= 3.9.0 Computes a geometry representing the point-set union of the input geometries.

### Functions changed in PostGIS 3.1

- **ST\_Force3D** - Changed: 3.1.0. Added support for supplying a non-zero Z value. XYZ
- **ST\_Force3DM** - Changed: 3.1.0. Added support for supplying a non-zero M value. XYM
- **ST\_Force3DZ** - Changed: 3.1.0. Added support for supplying a non-zero Z value. XYZ
- **ST\_Force4D** - Changed: 3.1.0. Added support for supplying non-zero Z and M values. XYZM
- **ST\_Histogram** - Changed: 3.1.0 Removed ST\_Histogram(table\_name, column\_name) variant. (bin; (population)
- **ST\_Quantile** - Changed: 3.1.0 Removed ST\_Quantile(table\_name, column\_name) variant. (population)

### 15.12.4 PostGIS Functions new or enhanced in 3.0

The functions given below are PostGIS functions that were added or enhanced.

Functions enhanced in PostGIS 3.0

- **ST\_AsMVT** - Enhanced: 3.0 - added support for Feature ID. Aggregate function returning a MVT representation of a set of rows.
- **ST\_Contains** - Enhanced: 3.0.0 enabled support for GEOMETRYCOLLECTION Tests if no points of B lie in the exterior of A, and A and B have at least one interior point in common.
- **ST\_ContainsProperly** - Enhanced: 3.0.0 enabled support for GEOMETRYCOLLECTION Tests if B intersects the interior of A but not the boundary or exterior.
- **ST\_CoveredBy** - Enhanced: 3.0.0 enabled support for GEOMETRYCOLLECTION Tests if no point in A is outside B
- **ST\_Covers** - Enhanced: 3.0.0 enabled support for GEOMETRYCOLLECTION Tests if no point in B is outside A
- **ST\_Crosses** - Enhanced: 3.0.0 enabled support for GEOMETRYCOLLECTION Tests if two geometries have some, but not all, interior points in common.
- **ST\_CurveToLine** - Enhanced: 3.0.0 implemented a minimum number of segments per linearized arc to prevent topological collapse. Converts a geometry containing curves to a linear geometry.
- **ST\_Disjoint** - Enhanced: 3.0.0 enabled support for GEOMETRYCOLLECTION Tests if two geometries are disjoint (they have no point in common).
- **ST\_Equals** - Enhanced: 3.0.0 enabled support for GEOMETRYCOLLECTION Tests if two geometries include the same set of points.
- **ST\_GeneratePoints** - Enhanced: 3.0.0, added seed parameter Generates random points contained in a Polygon or MultiPolygon.
- **ST\_GeomFromGeoJSON** - Enhanced: 3.0.0 parsed geometry defaults to SRID=4326 if not specified otherwise. GeoJSON
- **ST\_LocateBetween** - Enhanced: 3.0.0 - added support for POLYGON, TIN, TRIANGLE. Returns the portions of a geometry that match a measure range.
- **ST\_LocateBetweenElevations** - Enhanced: 3.0.0 - added support for POLYGON, TIN, TRIANGLE. Returns the portions of a geometry that lie in an elevation (Z) range.
- **ST\_Overlaps** - Enhanced: 3.0.0 enabled support for GEOMETRYCOLLECTION Tests if two geometries intersect and have the same dimension, but are not completely contained by each other.
- **ST\_Relate** - Enhanced: 3.0.0 enabled support for GEOMETRYCOLLECTION Tests if two geometries have a topological relationship matching an Intersection Matrix pattern, or computes their Intersection Matrix
- **ST\_Segmentize** - Enhanced: 3.0.0 Segmentize geometry now uses equal length segments
- **ST\_Touches** - Enhanced: 3.0.0 enabled support for GEOMETRYCOLLECTION Tests if two geometries have at least one point in common, but their interiors do not intersect.
- **ST\_Within** - Enhanced: 3.0.0 enabled support for GEOMETRYCOLLECTION Tests if no points of A lie in the exterior of B, and A and B have at least one interior point in common.

Functions changed in PostGIS 3.0

- **PostGIS\_Extensions\_Upgrade** - Changed: 3.0.0 to repackage loose extensions and support postgis\_raster. Packages and upgrades PostGIS extensions (e.g. postgis\_raster, postgis\_topology, postgis\_sfcgal) to latest available version.

- **ST\_3DDistance** - Changed: 3.0.0 - SFCGAL version removed. Tests if two geometries spatially intersect in 3D - only for points, linestrings, polygons, polyhedral surface (area).
- **ST\_3DIntersects** - Changed: 3.0.0 SFCGAL backend removed, GEOS backend supports TINs. Tests if two geometries spatially intersect in 3D - only for points, linestrings, polygons, polyhedral surface (area).
- **ST\_Area** - Changed: 3.0.0 - does not depend on SFCGAL anymore.
- **ST\_AsGeoJSON** - Changed: 3.0.0 support records as input Return a geometry as a GeoJSON element.
- **ST\_AsGeoJSON** - Changed: 3.0.0 output SRID if not EPSG:4326. Return a geometry as a GeoJSON element.
- **ST\_AsKML** - Changed: 3.0.0 - Removed the "versioned" variant signature. GML 2, GML 3.
- **ST\_Distance** - Changed: 3.0.0 - does not depend on SFCGAL anymore.
- **ST\_Intersection** - Changed: 3.0.0 does not depend on SFCGAL. Computes a geometry representing the shared portion of geometries A and B.
- **ST\_Intersects** - Changed: 3.0.0 SFCGAL version removed and native support for 2D TINs added. Tests if two geometries intersect (they have at least one point in common).
- **ST\_Union** - Changed: 3.0.0 does not depend on SFCGAL. Computes a geometry representing the point-set union of the input geometries.

## 15.12.5 PostGIS Functions new or enhanced in 2.5

The functions given below are PostGIS functions that were added or enhanced.

Functions new in PostGIS 2.5

- **PostGIS\_Extensions\_Upgrade** - Availability: 2.5.0 Packages and upgrades PostGIS extensions (e.g. postgis\_raster, postgis\_topology, postgis\_sfcgal) to latest available version.
- **ST\_Angle** - Availability: 2.5.0 Returns the angle between two line segments.
- **ST\_AsHexWKB** - Availability: 2.5.0 Return the Well-Known Binary (WKB) in Hex representation of the raster.
- **ST\_BandFileSize** - Availability: 2.5.0 Returns the file size of a band stored in file system. If no bandnum specified, 1 is assumed.
- **ST\_BandFileTimestamp** - Availability: 2.5.0 Returns the file timestamp of a band stored in file system. If no bandnum specified, 1 is assumed.
- **ST\_ChaikinSmoothing** - Availability: 2.5.0 Returns a smoothed version of a geometry, using the Chaikin algorithm.
- **ST\_FilterByM** - Availability: 2.5.0 Removes vertices based on their M value.
- **ST\_Grayscale** - Availability: 2.5.0 Creates a new one-8BUI band raster from the source raster and specified bands representing Red, Green and Blue.
- **ST\_LineInterpolatePoints** - Availability: 2.5.0 Returns points interpolated along a line at a fractional interval.
- **ST\_OrientedEnvelope** - Availability: 2.5.0 Returns a minimum-area rectangle containing a geometry.
- **ST\_QuantizeCoordinates** - Availability: 2.5.0 Sets least significant bits of coordinates to zero.

- **ST\_RastFromHexWKB** - Availability: 2.5.0 Return a raster value from a Hex representation of Well-Known Binary (WKB) raster.
- **ST\_RastFromWKB** - Availability: 2.5.0 Return a raster value from a Well-Known Binary (WKB) raster.
- **ST\_SetBandIndex** - Availability: 2.5.0 Update the external band number of an out-db band
- **ST\_SetBandPath** - Availability: 2.5.0 Update the external path and band number of an out-db band

#### Functions enhanced in PostGIS 2.5

- **ST\_AsBinary/ST\_AsWKB** - Enhanced: 2.5.0 Addition of ST\_AsWKB Return the Well-Known Binary (WKB) representation of the raster.
- **ST\_AsMVT** - Enhanced: 2.5.0 - added support parallel query. Aggregate function returning a MVT representation of a set of rows.
- **ST\_AsText** - Enhanced: 2.5 - optional parameter precision introduced. WKT(Well-Known Text) SRID
- **ST\_BandMetaData** - Enhanced: 2.5.0 to include outdbbandnum, filesize and filetimestamp for outdb rasters.
- **ST\_Buffer** - Enhanced: 2.5.0 - ST\_Buffer geometry support was enhanced to allow for side buffering specification side=both|left|right. Computes a geometry covering all points within a given distance from a geometry.
- **ST\_GeomFromGeoJSON** - Enhanced: 2.5.0 can now accept json and jsonb as inputs. GeoJSON
- **ST\_GeometricMedian** - Enhanced: 2.5.0 Added support for M as weight of points. (median)
- **ST\_Intersects** - Enhanced: 2.5.0 Supports GEOMETRYCOLLECTION. Tests if two geometries intersect (they have at least one point in common).
- **ST\_OffsetCurve** - Enhanced: 2.5 - added support for GEOMETRYCOLLECTION and MULTILINESTRING Returns an offset line at a given distance and side from an input line.
- **ST\_Scale** - Enhanced: 2.5.0 support for scaling relative to a local origin (origin parameter) was introduced. Scales a geometry by given factors.
- **ST\_Split** - Enhanced: 2.5.0 support for splitting a polygon by a multiline was introduced. Returns a collection of geometries created by splitting a geometry by another geometry.
- **ST\_Subdivide** - Enhanced: 2.5.0 reuses existing points on polygon split, vertex count is lowered from 8 to 5. Computes a rectilinear subdivision of a geometry.

#### Functions changed in PostGIS 2.5

- **ST\_GDALDrivers** - Changed: 2.5.0 - add can\_read and can\_write columns. Returns a list of raster formats supported by PostGIS through GDAL. Only those formats with can\_write=True can be used by ST\_AsGDALRaster

## 15.12.6 PostGIS Functions new or enhanced in 2.4

The functions given below are PostGIS functions that were added or enhanced.

Functions new in PostGIS 2.4

- **ST\_AsGeobuf** - Availability: 2.4.0 Return a Geobuf representation of a set of rows.
- **ST\_AsMVT** - Availability: 2.4.0 Aggregate function returning a MVT representation of a set of rows.
- **ST\_AsMVTGeom** - Availability: 2.4.0 Transforms a geometry into the coordinate space of a MVT tile.
- **ST\_Centroid** - Availability: 2.4.0 support for geography was introduced.
- **ST\_ForcePolygonCCW** - Availability: 2.4.0 Orients all exterior rings counter-clockwise and all interior rings clockwise.
- **ST\_ForcePolygonCW** - Availability: 2.4.0 Orients all exterior rings clockwise and all interior rings counter-clockwise.
- **ST\_FrechetDistance** - Availability: 2.4.0 - requires GEOS >= 3.7.0
- **ST\_MakeEmptyCoverage** - Availability: 2.4.0 Cover georeferenced area with a grid of empty raster tiles.

Functions enhanced in PostGIS 2.4

All aggregates now marked as parallel safe which should allow them to be used in plans that can employ parallelism.

PostGIS 2.4.1 `postgis_tiger_geocoder` set to load Tiger 2017 data. Can optionally load zip code 5-digit tabulation (zcta) as part of the `Loader_Generate_Nation_Script`.

- **Loader\_Generate\_Nation\_Script** - Enhanced: 2.4.1 zip code 5 tabulation area (zcta5) load step was fixed and when enabled, zcta5 data is loaded as a single table called `zcta5_all` as part of the nation script load.
- **Normalize\_Address** - Enhanced: 2.4.0 `norm_addy` object includes additional fields `zip4` and `address_alphanumeric`.
- **Page\_Normalize\_Address** - Enhanced: 2.4.0 `norm_addy` object includes additional fields `zip4` and `address_alphanumeric`.
- **Reverse\_Geocode** - Enhanced: 2.4.1 if optional `zcta5` dataset is loaded, the `reverse_geocode` function can resolve to state and zip even if the specific state data is not loaded. Refer to for details on loading `zcta5` data.

- **ST\_AsTWKB** - Enhanced: 2.4.0 memory and speed improvements. TWKB(Tiny Well-Known Binary)
- **ST\_Covers** - Enhanced: 2.4.0 Support for polygon in polygon and line in polygon added for geography type Tests if no point in B is outside A
- **ST\_CurveToLine** - Enhanced: 2.4.0 added support for max-deviation and max-angle tolerance, and for symmetric output. Converts a geometry containing curves to a linear geometry.
- **ST\_Project** - Enhanced: 2.4.0 Allow negative distance and non-normalized azimuth.
- **ST\_Reverse** - Enhanced: 2.4.0 support for curves was introduced.

#### Functions changed in PostGIS 2.4

All PostGIS aggregates now marked as parallel safe. This will force a drop and recreate of aggregates during upgrade which may fail if any user views or sql functions rely on PostGIS aggregates.

- **=** - Changed: 2.4.0, in prior versions this was bounding box equality not a geometric equality. If you need bounding box equality, use `ST_Equals` instead. Returns TRUE if the coordinates and coordinate order geometry/geography A are the same as the coordinates and coordinate order of geometry/geography B.
- **ST\_Node** - Changed: 2.4.0 this function uses GEOSNode internally instead of GEOSUnaryUnion. This may cause the resulting linestrings to have a different order and direction compared to PostGIS < 2.4. Nodes a collection of lines.

### 15.12.7 PostGIS Functions new or enhanced in 2.3

The functions given below are PostGIS functions that were added or enhanced.



#### Note

PostGIS 2.3.0: PostgreSQL 9.6+ support for parallel queries.



#### Note

PostGIS 2.3.0: PostGIS extension, all functions schema qualified to reduce issues in database restore.



#### Note

PostGIS 2.3.0: PostgreSQL 9.4+ support for BRIN indexes. Refer to Section [4.9.2](#).



#### Note

PostGIS 2.3.0: Tiger Geocoder upgraded to work with TIGER 2016 data.

#### Functions new in PostGIS 2.3

- `&&(geometry,gidx)` - Availability: 2.3.0 support for Block Range INdexes (BRIN) was introduced. Requires PostgreSQL 9.5+. Returns TRUE if a geometry's (cached) n-D bounding box intersects a n-D float precision bounding box (GIDX).
- `&&(gidx,geometry)` - Availability: 2.3.0 support for Block Range INdexes (BRIN) was introduced. Requires PostgreSQL 9.5+. Returns TRUE if a n-D float precision bounding box (GIDX) intersects a geometry's (cached) n-D bounding box.
- `&&(gidx,gidx)` - Availability: 2.3.0 support for Block Range INdexes (BRIN) was introduced. Requires PostgreSQL 9.5+. Returns TRUE if two n-D float precision bounding boxes (GIDX) intersect each other.
- `&&(box2df,box2df)` - Availability: 2.3.0 support for Block Range INdexes (BRIN) was introduced. Requires PostgreSQL 9.5+. Returns TRUE if two 2D float precision bounding boxes (BOX2DF) intersect each other.
- `&&(box2df,geometry)` - Availability: 2.3.0 support for Block Range INdexes (BRIN) was introduced. Requires PostgreSQL 9.5+. Returns TRUE if a 2D float precision bounding box (BOX2DF) intersects a geometry's (cached) 2D bounding box.
- `&&(geometry,box2df)` - Availability: 2.3.0 support for Block Range INdexes (BRIN) was introduced. Requires PostgreSQL 9.5+. Returns TRUE if a geometry's (cached) 2D bounding box intersects a 2D float precision bounding box (BOX2DF).
- `@(box2df,box2df)` - Availability: 2.3.0 support for Block Range INdexes (BRIN) was introduced. Requires PostgreSQL 9.5+. Returns TRUE if a 2D float precision bounding box (BOX2DF) is contained into another 2D float precision bounding box.
- `@(box2df,geometry)` - Availability: 2.3.0 support for Block Range INdexes (BRIN) was introduced. Requires PostgreSQL 9.5+. Returns TRUE if a 2D float precision bounding box (BOX2DF) is contained into a geometry's 2D bounding box.
- `@(geometry,box2df)` - Availability: 2.3.0 support for Block Range INdexes (BRIN) was introduced. Requires PostgreSQL 9.5+. Returns TRUE if a geometry's 2D bounding box is contained into a 2D float precision bounding box (BOX2DF).
- `ST_ClusterDBSCAN` - Availability: 2.3.0 Window function that returns a cluster id for each input geometry using the DBSCAN algorithm.
- `ST_ClusterKMeans` - Availability: 2.3.0 Window function that returns a cluster id for each input geometry using the K-means algorithm.
- `ST_WrapX` - Availability: 2.3.0 requires GEOS X; `&#xc12; &#xadfc; &#xcc98; &#xc5d0; &#xc11c; &#xb3c4; &#xd615; &#xc744; &#xb798; &#xd551; &#xd569; &#xb2c8; &#xb2e4;`
- `~(box2df,box2df)` - Availability: 2.3.0 support for Block Range INdexes (BRIN) was introduced. Requires PostgreSQL 9.5+. Returns TRUE if a 2D float precision bounding box (BOX2DF) contains another 2D float precision bounding box (BOX2DF).
- `~(box2df,geometry)` - Availability: 2.3.0 support for Block Range INdexes (BRIN) was introduced. Requires PostgreSQL 9.5+. Returns TRUE if a 2D float precision bounding box (BOX2DF) contains a geometry's 2D bonding box.
- `~(geometry,box2df)` - Availability: 2.3.0 support for Block Range INdexes (BRIN) was introduced. Requires PostgreSQL 9.5+. Returns TRUE if a geometry's 2D bonding box contains a 2D float precision bounding box (GIDX).

The functions given below are PostGIS functions that are enhanced in PostGIS 2.3.

- `ST_Contains` - Enhanced: 2.3.0 Enhancement to PIP short-circuit extended to support MultiPoints with few points. Prior versions only supported point in polygon.
- `ST_Covers` - Enhanced: 2.3.0 Enhancement to PIP short-circuit for geometry extended to support MultiPoints with few points. Prior versions only supported point in polygon.
- `ST_Expand` - Enhanced: 2.3.0 support was added to expand a box by different amounts in different dimensions.
- `ST_Intersects` - Enhanced: 2.3.0 Enhancement to PIP short-circuit extended to support MultiPoints with few points. Prior versions only supported point in polygon.
- `ST_Segmentize` - Enhanced: 2.3.0 Segmentize geography now uses equal length segments
- `ST_Transform` - Enhanced: 2.3.0 support for direct PROJ.4 text was introduced.
- `ST_Within` - Enhanced: 2.3.0 Enhancement to PIP short-circuit for geometry extended to support MultiPoints with few points. Prior versions only supported point in polygon.

## 15.12.8 PostGIS Functions new or enhanced in 2.2

The functions given below are PostGIS functions that were added or enhanced.

**Note**

postgis\_sfcgal now can be installed as an extension using `CREATE EXTENSION postgis_sfcgal;`

**Note**

PostGIS 2.2.0: Tiger Geocoder upgraded to work with TIGER 2015 data.

**Note**

`address_standardizer`, `address_standardizer_data_us` extensions for standardizing address data refer to Section [14.1](#) for details.

**Note**

Many functions in topology rewritten as C functions for increased performance.

Functions new in PostGIS 2.2

- **ST\_CPAWithin** - Availability: 2.2.0 Tests if the closest point of approach of two trajectories is within the specified distance.
- **ST\_ClipByBox2D** - Availability: 2.2.0 Computes the portion of a geometry falling within a rectangle.
- **ST\_ClosestPointOfApproach** - Availability: 2.2.0 Returns a measure at the closest point of approach of two trajectories.
- **ST\_ClusterIntersecting** - Availability: 2.2.0 Aggregate function that clusters the input geometries into connected sets.
- **ST\_ClusterWithin** - Availability: 2.2.0 Aggregate function that clusters the input geometries by separation distance.
- **ST\_DistanceCPA** - Availability: 2.2.0 Returns the distance between the closest point of approach of two trajectories.
- **ST\_IsValidTrajectory** - Availability: 2.2.0 Tests if the geometry is a valid trajectory.
- **ST\_Subdivide** - Availability: 2.2.0 Computes a rectilinear subdivision of a geometry.

The functions given below are PostGIS functions that are enhanced in PostGIS 2.2.

- **ST\_Scale** - Enhanced: 2.2.0 support for scaling all dimension (factor parameter) was introduced.
- **ST\_Split** - Enhanced: 2.2.0 support for splitting a line by a multiline, a multipoint or (multi)polygon boundary was introduced.

## 15.12.9 PostGIS functions breaking changes in 2.2

The functions given below are PostGIS functions that have possibly breaking changes in PostGIS 2.2. If you use any of these, you may need to check your existing code.

- **ST\_Equals** - Changed: 2.2.0 Returns true even for invalid geometries if they are binary equal
- **ST\_MemSize** - Changed: 2.2.0 name changed to `ST_MemSize` to follow naming convention.
- **ST\_PointInsideCircle** - Changed: 2.2.0 In prior versions this was called `ST_Point_Inside_Circle`

### 15.12.10 PostGIS Functions new or enhanced in 2.1

The functions given below are PostGIS functions that were added or enhanced.

**Note**

More Topology performance Improvements. Please refer to Chapter 10 for more details.

**Note**

Bug fixes (particularly with handling of out-of-band rasters), many new functions (often shortening code you have to write to accomplish a common task) and massive speed improvements to raster functionality. Refer to Chapter 12 for more details.

**Note**

PostGIS 2.1.0: Tiger Geocoder upgraded to work with TIGER 2012 census data. `geocode_settings` added for debugging and tweaking rating preferences, loader made less greedy, now only downloads tables to be loaded. PostGIS 2.1.1: Tiger Geocoder upgraded to work with TIGER 2013 data. Please refer to Section 14.2 for more details.

The functions given below are PostGIS functions that are enhanced in PostGIS 2.1.

- **ST\_DWithin** - Enhanced: 2.1.0 improved speed for geography. See Making Geography faster for details.
- **ST\_DWithin** - Enhanced: 2.1.0 support for curved geometries was introduced.
- **ST\_NumPoints** - Enhanced: 2.1.0 Faster speed. Reimplemented as native-C.
- **ST\_MakeValid** - Enhanced: 2.1.0, added support for GEOMETRYCOLLECTION and MULTIPOINT.

### 15.12.11 PostGIS functions breaking changes in 2.1

The functions given below are PostGIS functions that have possibly breaking changes in PostGIS 2.1. If you use any of these, you may need to check your existing code.

- **ST\_EstimatedExtent** - Changed: 2.1.0. Up to 2.0.x this was called `ST_Estimated_Extent`.

### 15.12.12 PostGIS Functions new, behavior changed, or enhanced in 2.0

The functions given below are PostGIS functions that were added, enhanced, or have Section 15.12.13 breaking changes in 2.0 releases.

New geometry types: TIN and Polyhedral surfaces was introduced in 2.0

**Note**

Greatly improved support for Topology. Please refer to Chapter 10 for more details.

**Note**

In PostGIS 2.0, raster type and raster functionality has been integrated. There are way too many new raster functions to list here and all are new so please refer to Chapter 12 for more details of the raster functions available. Earlier pre-2.0 versions had `raster_columns/raster_overviews` as real tables. These were changed to views before release. Functions such as `ST_AddRasterColumn` were removed and replaced with `AddRasterConstraints`, `DropRasterConstraints` as a result some apps that created raster tables may need changing.

**Note**

Tiger Geocoder upgraded to work with TIGER 2010 census data and now included in the core PostGIS documentation. A reverse geocoder function was also added. Please refer to Section [14.2](#) for more details.

- **ST\_3DDFullyWithin** - Availability: 2.0.0 Tests if two 3D geometries are entirely within a given 3D distance
- **ST\_3DDWithin** - Availability: 2.0.0 Tests if two 3D geometries are within a given 3D distance
- **ST\_3DIntersects** - Availability: 2.0.0 Tests if two geometries spatially intersect in 3D - only for points, linestrings, polygons, polyhedral surface (area).
- **ST\_IsValidDetail** - Availability: 2.0.0 Returns a valid\_detail row stating if a geometry is valid or if not a reason and a location.
- **ST\_IsValidReason** - Availability: 2.0 version taking flags. Returns text stating if a geometry is valid, or a reason for invalidity.
- **ST\_MakeValid** - Availability: 2.0.0 Attempts to make an invalid geometry valid without losing vertices.
- **ST\_Node** - Availability: 2.0.0 Nodes a collection of lines.
- **ST\_RelateMatch** - Availability: 2.0.0 Tests if a DE-9IM Intersection Matrix matches an Intersection Matrix pattern
- **ST\_Split** - Availability: 2.0.0 requires GEOS Returns a collection of geometries created by splitting a geometry by another geometry.
- **ST\_UnaryUnion** - Availability: 2.0.0 Computes the union of the components of a single geometry.

The functions given below are PostGIS functions that are enhanced in PostGIS 2.0.

- **Box2D** - Enhanced: 2.0.0 support for Polyhedral surfaces, Triangles and TIN was introduced.
- **Box3D** - Enhanced: 2.0.0 support for Polyhedral surfaces, Triangles and TIN was introduced.
- **ST\_3DExtent** - Enhanced: 2.0.0 support for Polyhedral surfaces, Triangles and TIN was introduced.
- **ST\_Affine** - Enhanced: 2.0.0 support for Polyhedral surfaces, Triangles and TIN was introduced.
- **ST\_Expand** - Enhanced: 2.0.0 support for Polyhedral surfaces, Triangles and TIN was introduced.
- **ST\_Extent** - Enhanced: 2.0.0 support for Polyhedral surfaces, Triangles and TIN was introduced.
- **ST\_MakeValid** - Enhanced: 2.0.1, speed improvements
- **ST\_Relate** - Enhanced: 2.0.0 - added support for specifying boundary node rule.
- **ST\_Rotate** - Enhanced: 2.0.0 support for Polyhedral surfaces, Triangles and TIN was introduced.
- **ST\_Rotate** - Enhanced: 2.0.0 additional parameters for specifying the origin of rotation were added.
- **ST\_RotateX** - Enhanced: 2.0.0 support for Polyhedral surfaces, Triangles and TIN was introduced.
- **ST\_RotateY** - Enhanced: 2.0.0 support for Polyhedral surfaces, Triangles and TIN was introduced.
- **ST\_RotateZ** - Enhanced: 2.0.0 support for Polyhedral surfaces, Triangles and TIN was introduced.
- **ST\_Scale** - Enhanced: 2.0.0 support for Polyhedral surfaces, Triangles and TIN was introduced.
- **ST\_Transform** - Enhanced: 2.0.0 support for Polyhedral surfaces was introduced.

### 15.12.13 PostGIS Functions changed behavior in 2.0

The functions given below are PostGIS functions that have changed behavior in PostGIS 2.0 and may require application changes.



#### Note

Most deprecated functions have been removed. These are functions that haven't been documented since 1.2 or some internal functions that were never documented. If you are using a function that you don't see documented, it's probably deprecated, about to be deprecated, or internal and should be avoided. If you have applications or tools that rely on deprecated functions, please refer to [?qandaentry] for more details.



#### Note

Bounding boxes of geometries have been changed from float4 to double precision (float8). This has an impact on answers you get using bounding box operators and casting of bounding boxes to geometries. E.g ST\_SetSRID(abbox) will often return a different more accurate answer in PostGIS 2.0+ than it did in prior versions which may very well slightly change answers to view port queries.



#### Note

The arguments hasnodata was replaced with exclude\_nodata\_value which has the same meaning as the older hasnodata but clearer in purpose.

- **ST\_3DExtent** - Changed: 2.0.0 In prior versions this used to be called ST\_Extent3D
- **ST\_3DMakeBox** - Changed: 2.0.0 In prior versions this used to be called ST\_MakeBox3D

### 15.12.14 PostGIS Functions new, behavior changed, or enhanced in 1.5

The functions given below are PostGIS functions that were introduced or enhanced in this minor release.

- **PostGIS\_LibXML\_Version** - Availability: 1.5 Returns the version number of the libxml2 library.
- **ST\_Buffer** - Availability: 1.5 - ST\_Buffer was enhanced to support different endcaps and join types. These are useful for example to convert road linestrings into polygon roads with flat or square edges instead of rounded edges. Thin wrapper for geography was added. Computes a geometry covering all points within a given distance from a geometry.
- **ST\_Covers** - Availability: 1.5 - support for geography was introduced. Tests if no point in B is outside A
- **ST\_DFullyWithin** - Availability: 1.5.0 Tests if two geometries are entirely within a given distance
- **ST\_DWithin** - Availability: 1.5.0 support for geography was introduced Tests if two geometries are within a given distance
- **ST\_Expand** - Availability: 1.5.0 behavior changed to output double precision instead of float4 coordinates. Returns a bounding box expanded from another bounding box or a geometry.
- **ST\_GeomFromKML** - Availability: 1.5, requires libxml2 2.6+ &#xb3c4;&#xd615;&#xc758; KML &#xd45c;&#xd604;&#xc2dd;&#xc785;&#xb825;&#xbc1b;&#xc544; PostGIS &#xb3c4;&#xd615;&#xac1d;&#xccb4;&#xb97c;&#xcd9c;&#xb825;&#xd569;&#xc785;&#xb825;&#xbc1b;&#xc544;
- **ST\_Intersection** - Availability: 1.5 support for geography data type was introduced. Computes a geometry representing the shared portion of geometries A and B.
- **ST\_Intersects** - Availability: 1.5 support for geography was introduced. Tests if two geometries intersect (they have at least one point in common).

### 15.12.15 PostGIS Functions new, behavior changed, or enhanced in 1.4

The functions given below are PostGIS functions that were introduced or enhanced in the 1.4 release.

- **ST\_ContainsProperly** - Tests if B intersects the interior of A but not the boundary or exterior. Availability: 1.4.0
- **ST\_IsValidReason** - Returns text stating if a geometry is valid, or a reason for invalidity. Availability: 1.4
- **ST\_LineCrossingDirection** - Returns a number indicating the crossing behavior of two LineStrings. Availability: 1.4
- **ST\_Union** - Computes a geometry representing the point-set union of the input geometries. Availability: 1.4.0 - ST\_Union was enhanced. ST\_Union(geomarray) was introduced and also faster aggregate collection in PostgreSQL.

### 15.12.16 PostGIS Functions new in 1.3

The functions given below are PostGIS functions that were introduced in the 1.3 release.

- **ST\_CurveToLine** - Converts a geometry containing curves to a linear geometry. Availability: 1.3.0
  - **ST\_LineToCurve** - Converts a linear geometry to a curved geometry. Availability: 1.3.0
-

## Chapter 16

# Reporting Problems

### 16.1 Reporting Software Bugs

Reporting bugs effectively is a fundamental way to help PostGIS development. The most effective bug report is that enabling PostGIS developers to reproduce it, so it would ideally contain a script triggering it and every information regarding the environment in which it was detected. Good enough info can be extracted running `SELECT postgis_full_version()` [for PostGIS] and `SELECT version()` [for postgresql].

If you aren't using the latest release, it's worth taking a look at its [release changelog](#) first, to find out if your bug has already been fixed.

Using the [PostGIS bug tracker](#) will ensure your reports are not discarded, and will keep you informed on its handling process. Before reporting a new bug please query the database to see if it is a known one, and if it is please add any new information you have about it.

You might want to read Simon Tatham's paper about [How to Report Bugs Effectively](#) before filing a new report.

### 16.2 Reporting Documentation Issues

The documentation should accurately reflect the features and behavior of the software. If it doesn't, it could be because of a software bug or because the documentation is in error or deficient.

Documentation issues can also be reported to the [PostGIS bug tracker](#).

If your revision is trivial, just describe it in a new bug tracker issue, being specific about its location in the documentation.

If your changes are more extensive, a patch is definitely preferred. This is a four step process on Unix (assuming you already have `git` installed):

1. Clone the PostGIS' git repository. On Unix, type:  
**`git clone https://git.osgeo.org/gitea/postgis/postgis.git`**  
This will be stored in the directory `postgis`
  2. Make your changes to the documentation with your favorite text editor. On Unix, type (for example):  
**`vim doc/postgis.xml`**  
Note that the documentation is written in DocBook XML rather than HTML, so if you are not familiar with it please follow the example of the rest of the documentation.
  3. Make a patch file containing the differences from the master copy of the documentation. On Unix, type:  
**`git diff doc/postgis.xml > doc.patch`**
  4. Attach the patch to a new issue in bug tracker.
-

## Appendix A

# Appendix

### A.1 PostGIS 3.3.5

2023/11/19

This version requires PostgreSQL 11-16, GEOS 3.6 or higher, and Proj 4.9+ Additional features are enabled if you are running GEOS 3.9+ (and ST\_MakeValid enhancements with 3.10+), Proj 6.1+, and PostgreSQL 15+.

#### A.1.1 Bug Fixes and Enhancements

[5598](#), Fix restore of 3.3 dumps (Sandro Santilli)

[5568](#), Improve robustness of topology face split handling (Sandro Santilli)

[5548](#), Fix box-filtered validity check of topologies with edge-less faces (Sandro Santilli)

[5442](#), [5525](#), [postgis\_tiger\_geocoder,postgis\_topology] Database search\_path does not do what it intends to do (Jelte Fennema, Sandro Santilli, Regina Obe)

[5494](#), Fix double-upgrade with view using st\_dwithin(text, ...) (Sandro Santilli)

Fix JsonB casting issue (Paul Ramsey)

[5616](#), Fix to find docbook on newer docbook installs (Regina Obe)

### A.2 PostGIS 3.3.4

2023/07/28

This version requires PostgreSQL 11-16, GEOS 3.6 or higher, and Proj 4.9+ Additional features are enabled if you are running GEOS 3.9+ (and ST\_MakeValid enhancements with 3.10+), Proj 6.1+, and PostgreSQL 15+.

#### A.2.1 Bug Fixes and Enhancements

[5450](#), Fix macro expansion recursion on powerpc architectures. (Bas Couwenberg)

[5395](#), [postgis\_topology] Allow unprivileged user dump of database with topology enabled (Sandro Santilli)

[5394](#), [postgis\_topology] Improve robustness of finding distinct vertices in topology edges (Sandro Santilli)

[5403](#), [postgis\_topology] Fix ValidateTopology(bbox) without topology in search\_path (Sandro Santilli)

[5410](#), [postgis\_raster] ST\_Value bilinear resample, don't throw an error if Band has no NODATA value (Regina Obe)

5385, Postgres malloc assertion fail when using pg\_cancel\_backend with ST\_AsMVT (Regina Obe, Paul Ramsey)

5452, Updated README.postgis for apt-based systems (Jelte Fennema)

5385, Take out interruptability of ST\_AsMVT as it causes backend crash under intense conditions (Regina Obe, Paul Ramsey)

## A.3 PostGIS 3.3.3

2023/05/29

This version requires PostgreSQL 11-16beta1, GEOS 3.6 or higher, and Proj 4.9+ Additional features are enabled if you are running GEOS 3.9+ (and ST\_MakeValid enhancements with 3.10+), Proj 6.1+, and PostgreSQL 15+.

### A.3.1 Bug Fixes and Enhancements

5384, Fix crash in ST\_AsGML when given id is longer than given prefix (Sandro Santilli)

5380, Fix 2.5 upgrades with views using geography based ST\_Distance (Sandro Santilli)

5303, Don't try removing closed edges endpoint from RemoveUnusedPrimitives (Sandro Santilli)

5289, Fix misleading message in RemoveUnusedPrimitives about doubly connected edges healing (Sandro Santilli)

5298, Fix CopyTopology exception with hierarchical layers (Sandro Santilli)

5299, Corrections to address\_standardizer\_data\_us lex (Regina Obe)

5106, Fix segfault in ST\_RemEdgeNewFace/ST\_RemEdgeModFace when no edge side-faces are found (Sandro Santilli)

5329, Fix downgrade protection introduced in 3.3.0 (Sandro Santilli)

5332, Keep proj errors from percolating into PgSQL log (Paul Ramsey)

5313, Fix memory access issue in ST\_InterpolateRaster (Paul Ramsey)

5338, Dump/Restore of raster table fails on enforce\_coverage\_tile\_rast constraint (Regina Obe)

5315, #5318, #5319, #5320, #5342 crashes on infinite coordinates (Regina Obe, Paul Ramsey)

5344, Include topology id sequence state in dumps (Sandro Santilli)

5288, ST\_LineCrossingDirection multi-cross error (Paul Ramsey)

5347, \_ST\_BestSRID crashes on ARM with infinite geometries (Regina Obe)

5331, [postgis\_tiger\_geocoder] reverse\_geocode, prefer addressable ranges (Regina Obe, Locance)

5363, Make ST\_ClusterDBScan interruptable (Paul Ramsey)

5257, #5261, #5277, #5304, #5308, #5374, Support for PG16 (Paul Ramsey, Regina Obe, Sandro , Laurenz Albe)

5371, ST\_Union segfaults (PG 11-12) (Regina Obe)

5378, ST\_Buffer(geography) drops SRID (Paul Ramsey)

5388, [postgis\_tiger\_geocoder] Fix faces table tiger\_gload, caused census rerelease of faces data (Regina Obe)

## A.4 PostGIS 3.3.2

This version requires PostgreSQL 11-15, GEOS 3.6 or higher, and Proj 4.9+ Additional features are enabled if you are running GEOS 3.9+ (and ST\_MakeValid enhancements with 3.10+), Proj 6.1+, and PostgreSQL 15+.

---

## A.4.1 Bug and Security Fixes

5248, Marc21 documentation missing (Regina Obe)

5240, ST\_DumpPoints and ST\_DumpSegments crash with empty polygon (Regina Obe)

[security] Add schema qual to upgrade util (Regina Obe)

5255, GH598, Remove forced static linking on shp2pgsql, pgsq2shp (Maxim Kochetkov)

5255, GH702, Remove forced static linking on raster2pgsql (Thomas Petazzoni)

4648, [security] Check function ownership at extension packaging time (Sandro Santilli) Thanks to Sven Klemm (Timescale) for the report

5241, Crash on ST\_SnapToGrid with empty multis (Regina Obe)

5081, Fix error in topology import of self-intersecting rings (Sandro Santilli)

5234, Fix 2.5d topology building regression (Sandro Santilli)

5280, Handle load of dbase character fields with no width specified (Regina Obe)

5264, Fix topology install on Alpine (Louis Descoteaux)

5244, postgis\_tiger\_geocoder update for TIGER 2022 (Regina Obe)

5285, Fix import/export of topologies with names needing quotes (Sandro Santilli)

4988, SFCGAL specify name of unknown type when SFCGAL can't process (Regina Obe)

5084, Bad rasterization of linestring (Gilles Vuidel)

## A.5 PostGIS 3.3.1

2022/09/09

This version requires PostgreSQL 11 - 15, GEOS 3.6 or higher, and Proj 5.2+. Additional features are enabled if you are running GEOS 3.9+ ST\_MakeValid enhancements with 3.10+, numerous additional enhancements with GEOS 3.11+. Requires SFCGAL 1.4.1+ for ST\_AlphaShape and ST\_OptimalAlphaShape.

### A.5.1 Bug Fixes

5227, typo in ST\_LineLocatePoint error message (Sandro Santilli)

5231, PG15 no longer compiles because SQL/JSON removed PG upstream (Regina Obe)

## A.6 PostGIS 3.3.0

2022/08/26

This version requires PostgreSQL 11 or higher, GEOS 3.6 or higher, and Proj 5.2+. Additional features are enabled if you are running GEOS 3.9+ ST\_MakeValid enhancements with 3.10+, numerous additional enhancements with GEOS 3.11+. Requires SFCGAL 1.4.1+ for ST\_AlphaShape and ST\_OptimalAlphaShape.

NOTE: GEOS 3.11.0 details at [GEOS 3.11.0 release notes](#)

The new configure --enable-lto flag improves speed of math computations. This new feature is disabled by default because on some platforms, causes compilation errors (BSD and MingW64 issues have been raised)

## A.6.1 New features

- 5116, Topology export/import scripts (Sandro Santilli)
- ST\_Letters creates geometries that look like letters (Paul Ramsey)
- 5037, postgis\_sfcgal: ST\_3DConvexHull (Loïc Bartoletti)
- postgis\_sfcgal: sfcgal\_full\_version - reports BOOST and CGAL version (Loïc Bartoletti)
- GH 659, MARC21/XML, ST\_GeomFromMARC21, ST\_AsMARC21 (Jim Jones)
- 5132, GH 683, sfcgal: ST\_3DUnion aggregate function (Sergei Shoulbakov)
- 5143, SFCGAL ST\_AlphaShape and ST\_OptimalAlphaShape Requires SFCGAL 1.4.1+ (Loïc Bartoletti)
- 5162, ST\_TriangulatePolygon with GEOS 3.11+ (Paul Ramsey, Martin Davis)
- 5162, ST\_SimplifyPolygonHull with GEOS 3.11+ (Paul Ramsey, Martin Davis)
- 5183, topology.RemoveUnusedPrimitives (Sandro Santilli)

## A.6.2 Breaking Changes

- Drop support for PostgreSQL 9.6 and 10 (Regina Obe)
- Change output for WKT MULTIPOINT. All points now wrapped in parens. (Even Rouault)
- GH 674, geometry validation and fixing is disabled for ST\_DumpAsPolygons and ST\_Polygon so it works faster but might produce invalid polygons. (Aliaksandr Kalenik)

## A.6.3 Enhancements

- 2861, Add index on topology.node(containing\_face) speeding up splitting and merging of faces (Sandro Santilli)
- 2083, Speed up ST\_RemEdge topology functions adding index on relation(element\_id) and edge\_data(abs\_next\*) (Sandro Santilli)
- 5118, Allow dropping topologies with missing topogeometry sequences (Sandro Santilli)
- 5111, faster topology face MBR computation (Sandro Santilli)
- postgis\_extensions\_upgrade() support for upgrades from any PostGIS version, including yet to be released ones (Sandro Santilli)
- 5040, add postgis\_sfcgal\_full\_version (Loïc Bartoletti)
- GH 655, GiST: balance the tree splits better in recursive calls (Darafei Praliaskouski)
- GH 657, GiST: do not call no-op decompress function (Aliaksandr Kalenik)
- 4939, 5161, ST\_LineMerge now has option to keep the directions of input linestrings, useful when processing road graphs. Requires GEOS 3.11. (Sergei Shoulbakov)
- ST\_ConcaveHull GEOS 3.11+ native implementation (Paul Ramsey, Martin Davis)
- ST\_ConcaveHull GEOS 3.11+ polygon-respecting native implementation (Paul Ramsey, Martin Davis)
- 4574, GH 678, 5121 Enable Link-Time Optimizations using --enable-lto (Sergei Shoulbakov)
- GH 676, faster ST\_Clip (Aliaksandr Kalenik)
- 5135, Fast GiST index build is enabled by default for PostgreSQL 15+ (Sergei Shoulbakov)
- 4939, 5161, ST\_LineMerge now has option to keep the directions of input linestrings, useful when processing road graphs. Requires GEOS 3.11. (Sergei Shoulbakov)
- 5158, pgtopo\_import / pgtopo\_export manpages (Sandro Santilli)
- 5170, add a optional max\_rows\_per\_copy to -Y option to raster2pgsql to control number of rows per copy statement. Default to 50 when not specified (Regina Obe)
- GH 698, support parallel aggregate for ST\_Union (Sergei Shoulbakov)
- 5024, Update spatial\_ref\_sys as part of ALTER EXTENSION update postgis (Paul Ramsey)

## A.6.4 Bug Fixes

These are fixes issues in prior minors not backported

[4912](#), GiST: fix crash on STORAGE EXTERNAL for geography (Aliaksandr Kalenik)

[5088](#), Memory corruption in mvt\_agg\_transfn (Victor Collod)

[5137](#), resetting interrupt flags before query execution (Sergei Shoulbakov)

[5148](#), ST\_Clip is more robust to alignment of raster and clip geometry (Sergei Shoulbakov)

[4932](#), Bug with geography ST\_Intersects / ST\_Distance (Paul Ramsey)

[5089](#), ST\_Reverse also reverses components of CompoundCurve (Paul Ramsey)

## A.7 PostGIS 3.3.0rc2

2022/08/22

This version requires PostgreSQL 11 or higher, GEOS 3.6 or higher, and Proj 5.2+. Additional features are enabled if you are running GEOS 3.9+ ST\_MakeValid enhancements with 3.10+, numerous additional enhancements with GEOS 3.11+. Requires SFCGAL 1.4.1+ for ST\_AlphaShape and ST\_OptimalAlphaShape.

NOTE: GEOS 3.11.0 was recently released, details at [GEOS 3.11.0 release notes](#)

The new `--enable-lto` flag improves speed of math computations. This new feature is disabled by default because on some platforms, causes compilation errors (BSD and MingW64 issues have been raised)

### A.7.1 Bug Fixes

[5089](#), ST\_Reverse also reverses components of CompoundCurve (Paul Ramsey)

[5181](#), Reset proj error state after failed parse (Paul Ramsey)

[5171](#), Short circuit geodesic distance when inputs equal (Paul Ramsey)

## A.8 PostGIS 3.3.0rc1

2022/08/08

This version requires PostgreSQL 11 or higher, GEOS 3.6 or higher, and Proj 5.2+. Additional features are enabled if you are running GEOS 3.9+ ST\_MakeValid enhancements with 3.10+, numerous additional enhancements with GEOS 3.11+. Requires SFCGAL 1.4.1+ for ST\_AlphaShape and ST\_OptimalAlphaShape.

NOTE: GEOS 3.11.0 was recently released, details at [GEOS 3.11.0 release notes](#)

The new `--enable-lto` flag improves speed of math computations. This new feature is disabled by default because on some platforms, causes compilation errors (BSD and MingW64 issues have been raised)

Use below to enable it.

```
./configure --enable-lto
```

Changes since PostGIS 3.3.0beta2:

## A.8.1 Bug Fixes

[5154](#), raster ST\_Value is undercosted (Regina Obe)

[5157](#), Revise minimum\_bounding\_circle Cunit test to be tolerant of small 32-bit floating point differences (Regina Obe)

[5191](#), Functions should use integer instead of int4 (Regina Obe)

[5139](#), PostGIS causes to\_jsonb to no longer be parallel safe, ST\_AsGeoJSON and ST\_AsGML are also parallel unsafe (Regina Obe, Paul Ramsey)

[5025](#), Ensure that additional operators are not appended when the function and opfamily disagree about dimensionality (Paul Ramsey)

[5195](#), #5196 Change address\_standardizer and postgis\_tiger\_geocoder CREATE EXTENSION to use CREATE instead of CREATE OR REPLACE. (Regina Obe)

[5202](#), Guard against downgrade (Sandro Santilli)

[5104](#), postgis\_extensions\_upgrade() fails with pgextwlist (Regina Obe)

## A.9 PostGIS 3.3.0beta2

2022/07/13

This version requires PostgreSQL 11 or higher, GEOS 3.6 or higher, and Proj 5.2+. Additional features are enabled if you are running GEOS 3.9+ ST\_MakeValid enhancements with 3.10+, numerous additional enhancements with GEOS 3.11+. Requires SFCGAL 1.4.1+ for ST\_AlphaShape and ST\_OptimalAlphaShape.

NOTE: GEOS 3.11.0 was recently released, details at [GEOS 3.11.0 release notes](#)

The new `--enable-lto` flag improves speed of math computations. This new feature is disabled by default because on some platforms, causes compilation errors (BSD and MingW64 issues have been raised)

Use below to enable it.

```
./configure --enable-lto
```

Changes since PostGIS 3.3.0beta1:

### A.9.1 New Features

[5183](#), topology.RemoveUnusedPrimitives (Sandro Santilli)

### A.9.2 Enhancements

[GH698](#), support parallel aggregate for ST\_Union (Sergei Shoulbakov)

### A.9.3 Bug Fixes

[5179](#), pgsq2shp syntax error on big-endian (Bas Couwenberg)

## A.10 PostGIS 3.3.0beta1

2022/07/03

This version requires PostgreSQL 11 or higher, GEOS 3.6 or higher, and Proj 5.2+. Additional features are enabled if you are running GEOS 3.9+ ST\_MakeValid enhancements with 3.10+, numerous additional enhancements with GEOS 3.11+.

Requires SFCGAL 1.4.1+ for ST\_AlphaShape and ST\_OptimalAlphaShape.

NOTE: GEOS 3.11.0 was recently released, details at [GEOS 3.11.0 release notes](#)

The new --enable-lto flag improves math computations. This new feature is disabled by default because on some platforms, causes compilation errors (BSD and MingW64 issues have been raised)

Use below to enable it.

```
./configure --enable-lto
```

### A.10.1 Enhancements

[5158](#), pgtopo\_import / pgtopo\_export manpages (Sandro Santilli)

[5170](#), add a optional max\_rows\_per\_copy to -Y option to raster2pgsql to control number of rows per copy statement. Default to 50 when not specified (Regina Obe)

[4939](#), [5161](#), ST\_LineMerge now has option to keep the directions of input linestrings, useful when processing road graphs. Requires GEOS 3.11. (Sergei Shoulbakov)

ST\_ConcaveHull GEOS 3.11+ polygon-respecting native implementation (Paul Ramsey, Martin Davis)

[5039](#), postgis\_tiger\_geocoder TIGER 2021 (Regina Obe)

### A.10.2 New features

>[5169](#), ST\_SimplifyPolygonHull (requires GEOS 3.11) (Paul Ramsey, Martin Davis)

[5162](#), ST\_TriangulatePolygon with GEOS 3.11+ (Paul Ramsey, Martin Davis)

### A.10.3 Bug Fix

[5173](#) st\_asflatgeobuf detoast crash (Paul Ramsey)

[4932](#), Bug with geography ST\_Intersects / ST\_Distance (Paul Ramsey)

[5114](#), pgsq2shp segfault with long or many truncated columns

## A.11 PostGIS 3.3.0alpha1

2022/05/21

This version requires PostgreSQL 11 or higher, GEOS 3.6 or higher, and Proj 4.9+. Additional features are enabled if you are running GEOS 3.9+ (precision feature of many processing functions) ST\_MakeValid enhancements with 3.10+, ST\_ConcaveHull native GEOS implementation with GEOS 3.11+

Requires SFCGAL 1.4.1+ for ST\_AlphaShape and ST\_OptimalAlphaShape.

The new --enable-lto flag improves math computations. This new feature is disabled by default because on some platforms, causes compilation errors (BSD and MingW64 issues have been raised)

Use below to enable it.

```
./configure --enable-lto
```

### A.11.1 Breaking changes

Drop support for PostgreSQL 9.6 and 10 (Regina Obe)

Change output for WKT MULTIPOINT. All points now wrapped in parens. (Even Roualt)

GH674, geometry validation and fixing is disabled for ST\_DumpAsPolygons and ST\_Polygon so it works faster but might produce invalid polygons. (Aliaksandr Kalenik)

### A.11.2 Enhancements

2861, Add index on topology.node(containing\_face) speeding up splitting and merging of faces (Sandro Santilli)

2083, Speed up ST\_RemEdge topology functions adding index on relation(element\_id) and edge\_data(abs\_next\*) (Sandro Santilli)

5118, Allow dropping topologies with missing topogeometry sequences (Sandro Santilli)

5111, faster topology face MBR computation (Sandro Santilli)

postgis\_extensions\_upgrade() support for upgrades from any PostGIS version, including yet to be released ones (Sandro Santilli)

5040, add postgis\_sfcgal\_full\_version (Loïc Bartoletti)

GH655, GiST: balance the tree splits better in recursive calls (Darafei Praliaskouski)

GH657, GiST: do not call no-op decompress function (Aliaksandr Kalenik)

4912, GiST: fix crash on STORAGE EXTERNAL for geography (Aliaksandr Kalenik)

ST\_ConcaveHull GEOS 3.11+ native implementation (Paul Ramsey, Martin Davis)

4574, GH678, #5121 Enable Link-Time Optimizations using --enable-lto (Sergei Shoulbakov)

GH676, faster ST\_Clip (Aliaksandr Kalenik)

5135, Fast GiST index build is enabled by default for PostgreSQL 15+ (Sergei Shoulbakov)

### A.11.3 New features

5116, Topology export/import scripts (Sandro Santilli)

ST\_Letters creates geometries that look like letters (Paul Ramsey)

5037, postgis\_sfcgal: ST\_3DConvexHull (Loïc Bartoletti)

postgis\_sfcgal: sfcgal\_full\_version - reports BOOST and CGAL version (Loïc Bartoletti)

GH659, MARC21/XML, ST\_GeomFromMARC21, ST\_AsMARC21 (Jim Jones)

5132, GH683, sfcgal: ST\_3DUnion aggregate function (Sergei Shoulbakov)

5143, SFCGAL ST\_AlphaShape and ST\_OptimalAlphaShape (Loïc Bartoletti)

### A.11.4 Bug Fix

5100, Support for PostgreSQL 15 (atoi removal) (Laurenz Albe)

5123, Support for PostgreSQL 15 - PG15 now exposes json types and functions, do not include for PG15+ (Regina Obe)

5088, Memory corruption in mvt\_agg\_transfn (Victor Collod)

5137, resetting interrupt flags before query execution (Sergei Shoulbakov)

5148, ST\_Clip is more robust to alignment of raster and clip geometry (Sergei Shoulbakov)

## A.12 PostGIS 3.2.0 (Olivier Courtin Edition)

2021/12/18

This version requires PostgreSQL 9.6 or higher, GEOS 3.6 or higher, and Proj 4.9+ Additional features are enabled if you are running GEOS 3.9+ (and ST\_MakeValid enhancements with 3.10+), Proj 6.1+, and PostgreSQL 14+.

Due to some query performance degradation with the new PG14 fast index build , we have decided to disable the feature by default until we get more user testing as to the true impact of real-world queries. If you are running PG14+, you can reenale it by doing:

```
ALTER OPERATOR FAMILY gist_geometry_ops_2d USING gist
  ADD FUNCTION 11 (geometry)
  geometry_gist_sortsupport_2d (internal);
```

To revert the change:

```
ALTER OPERATOR FAMILY gist_geometry_ops_2d using gist
  DROP FUNCTION 11 (geometry);
```

and then reindex your gist indexes

### A.12.1 Breaking changes

5008, Empty geometries are not reported as being within Infinite distance by ST\_DWithin (Sandro Santilli)

4824, Removed `--without-wagyu` build option. Using Wagyu is now mandatory to build with MVT support.

4933, topology.GetFaceByPoint will not work with topologies having invalid edge linking.

4981, ST\_StartPoint support any geometry. No longer returns null for non-linestrings.

4149, ST\_AsMVTGeom now preserves more of original geometry's details at scale close to target extent. If you need previous simplifying behaviour, you can ST\_Simplify the geometry in advance. (Darafei Praliaskouski)

- Proj 4.9 or higher is required

5000, Turn off Window support in ST\_AsMVT aggregate as no real use-case for it and it crashes with random input (Paul Ramsey)

### A.12.2 Enhancements

4997, FlatGeobuf format input/output (Björn Harrtell)

4575, GRANT SELECT on topology metadata tables to PUBLIC (Sandro Santilli)

2592, Do not allow CreateTopology to define topologies with SRID < 0 (Sandro Santilli)

3232, Prevent moving an isolated node to different face (Sandro Santilli)

- Consider collection TopoGeometries while editing topology primitives. (Sandro Santilli)

3248, Prevent removing isolated edges if used in a TopoGeometry (Sandro Santilli)

3231, Prevent removing isolated nodes if used in a TopoGeometry (Sandro Santilli)

3239, Prevent headling topology edges if the connecting node is used in the definition of a TopoGeometry (Sandro Santilli)

4950, Speed up checking containing\_face for nodes in ValidateTopology (Sandro Santilli)

4945, Multi-shell face check in ValidateTopology (Sandro Santilli)

4944, Side-location conflict check in ValidateTopology (Sandro Santilli)

3042, ValidateTopology check for edge linking (Sandro Santilli)

- 3276, ValidateTopology check for face's mbr (Sandro Santilli)
- 4936, Bounding box limited ValidateTopology (Sandro Santilli)
- 4933, Speed up topology building in presence of big faces (Sandro Santilli)
- 3233, ValidateTopology check for node's containing\_face (Sandro Santilli)
- 4830, ValidateTopology check for edges side face containment (Sandro Santilli)
- 4827, Allow NaN coordinates in WKT input (Paul Ramsey)
- ST\_Value() accepts resample parameter to add bilinear option (Paul Ramsey)
- 3778, #4401, ST\_Boundary now works for TIN and does not linearize curves (Aliaksandr Kalenik)
- 4881, #4884, Store sign of edge\_id for lineal TopoGeometry in relation table to retain direction (Sandro Santilli)
- 4628, Add an option to disable ANALYZE when loading shapefiles (Stefan Corneliu Petrea)
- 4924, Faster ST\_RemoveRepeatedPoints on large multipoints, O(NlogN) instead of O(N<sup>2</sup>) (Aliaksandr Kalenik, Darafei Praliaskouski)
- 4925, fix ST\_DumpPoints to not overlook points (Aliaksandr Kalenik)
- ST\_SRID(topogeometry) override, to speedup lookups (Sandro Santilli)
- 2175, Avoid creating additional nodes when adding same closed line to topology (Sandro Santilli)
- 4974, Upgrade path for address\_standardizer\_data\_us (Jan Katins of Aiven, Regina Obe)
- 4975, PostGIS upgrade change to not use temp tables (Jan Katins of Aiven)
- 4981, ST\_StartPoint support any geometry (Aliaksandr Kalenik)
- 4799, Include srs in GeoJSON where it exists in spatial\_ref\_sys.
- 4986, GIST indexes on Postgres 14 are now created faster using Hilbert-sorting method. (Han Wang, Aliaksandr Kalenik, Darafei Praliaskouski, Giuseppe Broccolo)
- 4949, Use proj\_normalize\_for\_visualization to hand "axis swap" decisions (Paul Ramsey)
- GH647, ST\_PixelAsCentroids, ST\_PixelAsCentroid reimplemented on top of a C function (Sergei Shoulbakov)
- GH648, ST\_AsMVTGeom now uses faster clipping (Aliaksandr Kalenik)
- 5018, psql2shp basic support for WITH CTE clause (Regina Obe)
- 5019, address\_standardizer: Add support for pcre2 (Paul Ramsey)

### A.12.3 New features

- 4923, topology.ValidateTopologyRelation (Sandro Santilli)
  - 4933, topology.GetFaceContainingPoint (Sandro Santilli)
  - 2175, ST\_Scroll (Sandro Santilli)
  - 4841, FindTopology to quickly get a topology record (Sandro Santilli)
  - 4869, FindLayer to quickly get a layer record (Sandro Santilli)
  - 4851, TopoGeom\_addTopoGeom function (Sandro Santilli)
  - ST\_MakeValid(geometry, options) allows alternative validity building algorithms with GEOS 3.10 (Paul Ramsey)
  - ST\_InterpolateRaster() fills in raster cells between sample points using one of a number of algorithms (inverse weighted distance, average, etc) using algorithms from GDAL (Paul Ramsey)
  - ST\_Contour() generates contour lines from raster values using algorithms from GDAL (Paul Ramsey)
  - ST\_SetZ()/ST\_SetM() fills in z/m coordinates of a geometry using data read from a raster (Paul Ramsey)
-

New `postgis.gdal_vsi_options` GUC allows out-db rasters on VSI network services to be accessed with authentication keys, etc. (Paul Ramsey)

`ST_DumpSegments` returns a set of segments of input geometry (Aliaksandr Kalenik)

4859, `ST_Point`, `ST_PointZ`, `ST_PointM`, `ST_PointZM`, constructors with SRID parameter (Paul Ramsey)

4808, `ST_ClusterKMeans` now supports `max_radius` argument. Use it when you're not sure what is the number of clusters but you know what the size of clusters should be. (Darafei Praliaskouski)

## A.13 PostGIS 3.2.0beta3

2021/12/04

This version requires PostgreSQL 9.6 or higher, GEOS 3.6 or higher, and Proj 4.9+ Additional features are enabled if you are running GEOS 3.9+ (and `ST_MakeValid` enhancements with 3.10+), Proj 6.1+, and PostgreSQL 14+.

Due to some query performance degradation with the new PG14 fast index build , we have decided to disable the feature by default until we get more user testing as to the true impact of real-world queries. If you are running PG14+, you can reenale it by doing:

```
ALTER OPERATOR FAMILY gist_geometry_ops_2d USING gist
  ADD FUNCTION 11 (geometry)
  geometry_gist_sortsupport_2d (internal);
```

To revert the change:

```
ALTER OPERATOR FAMILY gist_geometry_ops_2d using gist
  DROP FUNCTION 11 (geometry);
```

and then reindex your gist indexes

Changes since PostGIS 3.2.0beta2 release:

### A.13.1 Breaking changes / fixes

5028, `ST_AsFlatGeobuf` crashes on mixed geometry input (Björn Harrtell)

5029, `ST_AsFlatGeobuf` indexed output corruption (Björn Harrtell)

5014, Crash on `ST_TableFromFlatGeobuf` (Björn Harrtell)

Rename `ST_TableFromFlatGeobuf` to `ST_FromFlatGeobufToTable` (Björn Harrtell)

PG14 fast index building disabled by default. (Paul Ramsey)

## A.14 Release 3.2.0beta2

Release date: 2021/11/26

This version requires PostgreSQL 9.6 or higher, GEOS 3.6 or higher, and Proj 4.9+ Additional features are enabled if you are running GEOS 3.9+ (and `ST_MakeValid` enhancements with 3.10+), Proj 6.1+, and PostgreSQL 14+. Changes since PostGIS 3.2.0beta1 release:

### A.14.1 Breaking changes / fixes

5016, loader (shp2pgsq): Respect `LDFLAGS` (Greg Troxel)

5005, `ST_AsFlatGeoBuf` crashes on tables when geometry column is not the first column (Björn Harrtell)

5017, `topology.ValidateTopology` error relation "shell\_check" already exists (Sandro Santilli)

## A.14.2 Enhancements

5018, `pgsql2shp` basic support for WITH CTE clause (Regina Obe)

5019, `address_standardizer`: Add support for `pcr2` (Paul Ramsey)

GH647, `ST_AsMVTGeom` now uses faster clipping (Aliaksandr Kalenik)

GH648, `ST_PixelAsCentroids`, `ST_PixelAsCentroid` reimplemented on top of a C function (Sergei Shoulbakov)

## A.15 Release 3.2.0beta1

Release date: 2021/10/23

This version requires PostgreSQL 9.6 or higher, GEOS 3.6 or higher, and Proj 4.9+ Additional features are enabled if you are running GEOS 3.9+ (and `ST_MakeValid` enhancements with 3.10+), Proj 6.1+, and PostgreSQL 14+.

### A.15.1 Bug Fixes and Breaking Changes

5012, Clean regress against released GEOS 3.10.0 (Regina Obe, Paul Ramsey)

5000, Turn off Window support in `ST_AsMVT` aggregate as no real use-case for it and it crashes with random input (Paul Ramsey)

4994, `shp2pgsql` is sometimes missing the INSERT statements (Sandro Santilli)

4990, `getfacecontainingpoint` fails on i386 (Sandro Santilli)

5008, Have `ST_DWithin` with EMPTY operand always return false (Sandro Santilli)

5002, `liblwgeom` should build with warning flags by default (Sandro Santilli)

### A.15.2 Enhancements

4997, FlatGeobuf format input/output (Björn Harrtell)

## A.16 Release 3.2.0alpha1

Release date: 2021/09/10

This version requires PostgreSQL 9.6 or higher, GEOS 3.6 or higher, and Proj 4.9 or higher Additional features are enabled if you are running GEOS 3.9+ (more with GEOS 3.10+), Proj 6.1+, or PostgreSQL 14+.

### A.16.1 Breaking changes

#4824, Removed `--without-wagyu` build option. Using Wagyu is now mandatory to build with MVT support.

#4933, `topology.GetFaceByPoint` will not work with topologies having invalid edge linking.

#4981, `ST_StartPoint` support any geometry. No longer returns null for non-linestrings.

#4149, `ST_AsMVTGeom` now preserves more of original geometry's details at scale close to target extent. If you need previous simplifying behaviour, you can `ST_Simplify` the geometry in advance. (Darafei Praliaskouski)

Proj 4.9 or higher is required.

---

## A.16.2 Enhancements

- #2592, Do not allow CreateTopology to define topologies with SRID > 0 (Sandro Santilli)
- #3232, Prevent moving an isolated node to different face (Sandro Santilli)
- Consider collection TopoGeometries while editing topology primitives. (Sandro Santilli)
- #3248, Prevent removing isolated edges if used in a TopoGeometry (Sandro Santilli)
- #3231, Prevent removing isolated nodes if used in a TopoGeometry (Sandro Santilli)
- #3239, Prevent headling topology edges if the connecting node is used in the definition of a TopoGeometry (Sandro Santilli)
- #4950, Speed up checking containing\_face for nodes in ValidateTopology (Sandro Santilli)
- #4945, Multi-shell face check in ValidateTopology (Sandro Santilli)
- #4944, Side-location conflict check in ValidateTopology (Sandro Santilli)
- #3042, ValidateTopology check for edge linking (Sandro Santilli)
- #3276, ValidateTopology check for face's mbr (Sandro Santilli)
- #4936, Bounding box limited ValidateTopology (Sandro Santilli)
- #4933, Speed up topology building in presence of big faces (Sandro Santilli)
- #3233, ValidateTopology check for node's containing\_face (Sandro Santilli)
- #4830, ValidateTopology check for edges side face containment (Sandro Santilli)
- #4827, Allow NaN coordinates in WKT input (Paul Ramsey)
- ST\_Value() accepts resample parameter to add bilinear option (Paul Ramsey)
- #3778, #4401, ST\_Boundary now works for TIN and does not linearize curves (Aliaksandr Kalenik)
- #4881, #4884, Store sign of edge\_id for lineal TopoGeometry in relation table to retain direction (Sandro Santilli)
- #4628, Add an option to disable ANALYZE when loading shapefiles (Stefan Corneliu Petrea)
- #4924, Faster ST\_RemoveRepeatedPoints on large multipoints, O(NlogN) instead of O(N<sup>2</sup>) (Aliaksandr Kalenik, Darafei Praliaskouski)
- #4925, fix ST\_DumpPoints to not overlook points (Aliaksandr Kalenik)
- ST\_SRID(topogeometry) override, to speedup lookups (Sandro Santilli)
- #2175, Avoid creating additional nodes when adding same closed line to topology (Sandro Santilli)
- #4974, Upgrade path for address\_standardizer\_data\_us (Jan Katins of Aiven, Regina Obe)
- #4975, PostGIS upgrade change to not use temp tables (Jan Katins of Aiven)
- #4981, ST\_StartPoint support any geometry (Aliaksandr Kalenik)
- #4799, Include srs in GeoJSON where it exists in spatial\_ref\_sys.
- #4986, GIST indexes on Postgres 14 are now created faster using Hilbert-sorting method. (Han Wang, Aliaksandr Kalenik, Darafei Praliaskouski, Giuseppe Broccolo)
- #4949, Use proj\_normalize\_for\_visualization to hand "axis swap" decisions (Paul Ramsey)

## A.16.3 New features

- #4923, topology.ValidateTopologyRelation (Sandro Santilli)
- #4933, topology.GetFaceContainingPoint (Sandro Santilli)
- #2175, ST\_Scroll (Sandro Santilli)
- #4841, FindTopology to quickly get a topology record (Sandro Santilli)

#4869, FindLayer to quickly get a layer record (Sandro Santilli)

#4851, TopoGeom\_addTopoGeom function (Sandro Santilli)

ST\_MakeValid(geometry, options) allows alternative validity building algorithms with GEOS 3.10 (Paul Ramsey)

ST\_InterpolateRaster() fills in raster cells between sample points using one of a number of algorithms (inverse weighted distance, average, etc) using algorithms from GDAL (Paul Ramsey)

ST\_Contour() generates contour lines from raster values using algorithms from GDAL (Paul Ramsey)

ST\_SetZ()/ST\_SetM() fills in z/m coordinates of a geometry using data read from a raster (Paul Ramsey)

New postgis.gdal\_vsi\_options GUC allows out-db rasters on VSI network services to be accessed with authentication keys, etc. (Paul Ramsey)

ST\_DumpSegments returns a set of segments of input geometry (Aliaksandr Kalenik)

#4859, ST\_Point, ST\_PointZ, ST\_PointM, ST\_PointZM, constructors with SRID parameter (Paul Ramsey)

#4808, ST\_ClusterKMeans now supports max\_radius argument. Use it when you're not sure what is the number of clusters but you know what the size of clusters should be. (Darafei Praliaskouski)

## A.17 Release 3.1.0beta1

Release date: 2020/12/09

Only changes since 3.1.0alpha2 are listed. This version requires PostgreSQL 9.6-13 and GEOS >= 3.6+ Additional features and enhancements enabled if you are running Proj6+, PostgreSQL 12+, and GEOS 3.9.0dev

### A.17.1 Breaking changes

4214, Deprecated ST\_Count(tablename,...), ST\_ApproxCount(tablename, ...), ST\_SummaryStats(tablename, ..), ST\_Histogram(tablename, ...), ST\_ApproxHistogram(tablename, ...), ST\_Quantile(tablename, ...), ST\_ApproxQuantile(tablename, ...) removed. (Darafei Praliaskouski)

### A.17.2 Enhancements

4801, ST\_ClusterKMeans supports weights in POINT[Z]M geometries (Darafei Praliaskouski)

4804, ST\_ReducePrecision (GEOS 3.9+) allows valid precision reduction (Paul Ramsey)

4805, \_ST\_SortableHash exposed to work around parallel sorting performance issue in Postgres. If your table is huge, use ORDER BY \_ST\_SortableHash(geom) instead of ORDER BY geom to make parallel sort faster (Darafei Praliaskouski)

4625, Correlation statistics now calculated. Run ANALYZE for BRIN indexes to start kicking in. (Darafei Praliaskouski)

Fix axis order issue with urn:ogc:def:crs:EPSG in ST\_GeomFromGML() (Even Roualt)

## A.18 Release 3.1.0alpha3

Release date: 2020/11/19

Only changes since 3.1.0alpha2 are listed. This version requires PostgreSQL 9.6-13 and GEOS >= 3.6+ Additional features and enhancements enabled if you are running Proj6+, PostgreSQL 12+, and GEOS 3.9.0dev

### A.18.1 Breaking changes

4737, Bump minimum protobuf-c requirement to 1.1.0 (Raúl Marín) The configure step will now fail if the requirement isn't met or explicitly disabled (--without-protobuf)

4258, Untangle postgis\_sfcgal from postgis into its own lib file (Regina Obe)

## A.18.2 New features

4698, Add a precision parameter to ST\_AsEWKT (Raúl Marín)

Add a gridSize optional parameter to ST\_Union, ST\_UnaryUnion, ST\_Difference, ST\_Intersection, ST\_SymDifference, ST\_Subdivide  
Requires GEOS 3.9 (Sandro Santilli)

## A.18.3 Enhancements

4789, Speed up TopoJSON output for areal TopoGeometry with many holes (Sandro Santilli)

4758, Improve topology noding robustness (Sandro Santilli)

Make ST\_Subdivide interruptable (Sandro Santilli)

4660, Changes in double / coordinate printing (Raúl Marín) - Use the shortest representation (enough to guarantee roundtrip).  
- Uses scientific notation for absolute numbers smaller than 1e-8. The previous behaviour was to output 0 for absolute values smaller than 1e-12 and fixed notation for anything bigger than that. - Uses scientific notation for absolute numbers greater than 1e+15 (same behaviour). - The precision parameter now also affects the scientific notation (before it was fixed [5-8]). - All output functions now respect the requested precision (without any limits). - The default precision is the same (9 for GeoJSON, 15 for everything else).

4729, WKT/KML: Print doubles directly into stringbuffers (Raúl Marín)

4533, Use the standard coordinate printing system for box types (Raúl Marín)

4686, Avoid decompressing geographies when possible (Raúl Marín) Affects ANALYZE, \_ST\_PointOutside, postgis\_geobbox, ST\_CombineBbox(box2d, geometry), ST\_ClipByBox2D when the geometry is fully inside or outside the bbox and ST\_BoundingDiagon

4741, Don't use ST\_PointInsideCircle if you need indexes, use ST\_DWithin instead. Documentation adjusted (Darafei Praliaskouski)

4737, Improve performance and reduce memory usage in ST\_AsMVT, especially in queries involving parallelism (Raúl Marín)

4746, Micro optimizations to the serialization process (Raúl Marín)

4719, Fail fast when srids don't match ST\_Intersection(geometry,raster) Also schema qualify calls in function. (Regina Obe)

4784, Add ST\_CollectionExtract(geometry) with default behaviour of extracting the components of highest coordinate dimension. (Paul Ramsey)

## A.18.4 Bug Fixes

4691, Fix segfault during gist index creation with empty geometries (Raúl Marín)

Fix handling of bad WKB inputs (Oracle types) and unit tests for malformed WKB. Remove memory leaks in malformed WKB cases. (Paul Ramsey)

4740, Round values in geography\_distance\_tree as we do on geography\_distance (Raúl Marín, Paul Ramsey, Regina Obe)

4739, Ensure all functions using postgis\_oid initialize the internal cache (Raúl Marín)

4767, #4768, #4771, #4772, Fix segfault when parsing invalid WKB (Raúl Marín)

4769, Fix segfault in st\_addband (Raúl Marín)

4790, Fix ST\_3Dintersects calculations with identical vertices (Nicklas Avén)

4742, tiger geocoder reverted to 2018 version on tiger upgrade (Regina Obe)

3372, TopoElementArray cannot be null - change domain constraint (Regina Obe)

## A.19 Release 3.1.0alpha2

Release date: 2020/07/18

Only changes since 3.1.0alpha1 are listed. This version requires PostgreSQL 9.6-13 and GEOS >= 3.6+ Additional features and enhancements enabled if you are running Proj6+, PostgreSQL 12+, and GEOS 3.9.0dev

### A.19.1 New Features

- 4656, Cast a `geojson_text::geometry` for implicit GeoJSON ingestion (Raúl Marín)
- 4687, Expose GEOS `MaximumInscribedCircle` (Paul Ramsey)
- 4710, `ST_ClusterKMeans` now works with 3D geometries (Darafei Praliaskouski)

### A.19.2 Enhancements

- 4675, `topology.GetRingEdges` now implemented in C (Sandro Santilli)
- 4681, `ST_GetFaceGeometry`: print corruption information (Sandro Santilli)
- 4651, `ST_Simplify`: Don't copy if nothing is removed (Raúl Marín)
- 4657, Avoid De-TOASTing where possible (Paul Ramsey)
- 4490, Tweak function costs (Raúl Marín)
- 4672, Cache `getSRSbySRID` and `getSRIDbySRS` (Raúl Marín)
- 4676, Avoid decompressing toasted geometries to read only the header (Raúl Marín) Optimize cast to Postgresql point type (Raúl Marín)
- 4620, Update internal `wagyu` to 0.5.0 (Raúl Marín)
- 4623, Optimize `varlena` returning functions (Raúl Marín)
- 4677, Share `gserialized` objects between different cache types (Raúl Marín)
- Fix compilation with MSVC compiler / Standardize shebangs (Loïc Bartoletti)

### A.19.3 Bug fixes

- 4652, Fix several memory related bugs in `ST_GeomFromGML` (Raúl Marín)
- 4661, Fix access to `spatial_ref_sys` with a non default schema (Raúl Marín)
- 4670, `ST_AddPoint`: Fix bug when a positive position is requested (Raúl Marín)
- 4699, crash on null input to `ST_Union(raster, otherarg)` (Jaime Casanova, 2ndQuadrant)
- 4716, Fix several issues with `pkg-config` in the `configure` script (Raúl Marín)

## A.20 Release 3.1.0alpha1

Release date: 2020/02/01

This version requires PostgreSQL 9.6+-13 and GEOS >= 3.6+ Additional features and enhancements enabled if you are running Proj6+, PostgreSQL 12+, and GEOS 3.8.0

### A.20.1 Breaking Changes

- svn number replaced by git hash in version output (Sandro Santilli, Raúl Marín)
- 4577, Drop support for PostgreSQL 9.5 (Raúl Marín)
- 4579, Drop `postgis_proc_set_search_path.pl` (Raúl Marín)
- 4601, `ST_TileEnvelope` signature changed.
- 3057, `ST_Force3D`, `ST_Force3DZ`, `ST_Force3DM` and `ST_Force4D` signatures changed.

## A.20.2 New features

- 4601, Add ST\_TileEnvelope margin argument (Yuri Astrakhan)
- 2972, Add quiet mode (-q) to pgsq2shp (Kristian Thy)
- 4617, Add configure switch `--without-phony-revision` (Raúl Marín)
- 3057, Optional value params for Force3D\*, Force4D functions (Kristian Thy)
- 4624, ST\_HexagonGrid and ST\_SquareGrid, set returning functions to generate tilings of the plane (Paul Ramsey)

## A.20.3 Enhancements

- 4539, Unify libm includes (Raúl Marín)
- 4569, Allow unknown SRID geometry insertion into typmod SRID column (Paul Ramsey)
- 4149, ST\_Simplify(geom, 0) is now O(N). ST\_Affine (ST\_Translate, ST\_TransScale, ST\_Rotate) optimized. ST\_SnapToGrid optimized. (Darafei Praliaskouski)
- 4574, Link Time Optimizations enabled (Darafei Praliaskouski)
- 4578, Add parallism and cost properties to brin functions (Raúl Marín)
- 4473, Silence yacc warnings (Raúl Marín)
- 4589, Disable C asserts when building without "--enable-debug" (Raúl Marín)
- 4543, Introduce ryu to print doubles (Raúl Marín)
- 4626, Support pkg-config for libxml2 (Bas Couwenberg)
- 4615, Speed up gejson output (Raúl Marín)

## A.21 Release 3.0.0

Release date: 2019/10/20

This version requires PostgreSQL 9.5+-12 and GEOS >= 3.6+ Additional features and enhancements enabled if you are running Proj6+, PostgreSQL 12, and GEOS 3.8.0

### A.21.1 New Features

- 2902, postgis\_geos\_noop (Sandro Santilli)
- 4128, ST\_AsMVT support for Feature ID (Stepan Kuzmin)
- 4230, SP-GiST and GiST support for ND box operators overlaps, contains, within, equals (Esteban Zimányi and Arthur Lesuisse from Université Libre de Bruxelles (ULB), Darafei Praliaskouski)
- 4171, ST\_3DLineInterpolatePoint (Julien Cabieces, Vincent Mora)
- 4311, Introduce WAGYU to validate MVT polygons. This option requires a C++11 compiler and will use CXXFLAGS (not CFLAGS). Add `--without-wagyu` to disable this option and keep the behaviour from 2.5 (Raúl Marín)
- 1833, ST\_AsGeoJSON(row) generates full GeoJSON Features (Joe Conway)
- 3687, Casts json(geometry) and jsonb(geometry) for implicit GeoJSON generation (Paul Ramsey)
- 4198, Add ST\_ConstrainedDelaunayTriangles SFCGAL function (Darafei Praliaskouski)

## A.21.2 Breaking Changes

4267, Bump minimum GEOS version to 3.6 (Regina Obe, Darafei Praliaskouski)

3888, Raster support now available as a separate extension (Sandro Santilli)

3807, Extension library files no longer include the minor version. Use New configure switch `--with-library-minor-version` if you need the old behavior (Regina Obe)

4230, ND box operators (overlaps, contains, within, equals) now don't look on dimensions that aren't present in both operands. Please REINDEX your ND indexes after upgrade. (Darafei Praliaskouski)

4229, Dropped support for PostgreSQL < 9.5. (Darafei Praliaskouski)

4260, liblwgeom headers are not installed anymore. If your project depends on them available, please use `librttopo` instead. (Darafei Praliaskouski)

4258, Remove SFCGAL support for `ST_Area`, `ST_Distance`, `ST_Intersection`, `ST_Difference`, `ST_Union`, `ST_Intersects`, `ST_3DIntersect`, `ST_3DDistance` and `postgis.backend` switch (Darafei Praliaskouski)

4267, Enable Proj 6 deprecated APIs (Darafei Praliaskouski, Raúl Marín)

4268, Bump minimum SFCGAL version to 1.3.1 (Darafei Praliaskouski)

4331, `ST_3DMakeBox` now returns error instead of a miniscule box (Regina Obe)

4342, Removed "versioned" variants of `ST_AsGeoJSON` and `ST_AsKML` (Paul Ramsey)

4356, `ST_Accum` removed. Use `array_agg` instead. (Darafei Praliaskouski)

4414, Include version number in `address_standardizer` lib (Raúl Marín)

4334, Fix upgrade issues related to renamed function parameters (Raúl Marín)

4442, `raster2pgsql` now skips NODATA tiles. Use `-k` option if you still want them in database for some reason. (Darafei Praliaskouski)

4433, 32-bit hash fix (requires reindexing `hash(geometry)` indexes) (Raúl Marín)

3383, Sorting now uses Hilbert curve and Postgres Abbreviated Compare. You need to REINDEX your btree indexes if you had them. (Darafei Praliaskouski)

## A.21.3 Enhancements

4341, Using "support function" API in PostgreSQL 12+ to replace SQL inlining as the mechanism for providing index support under `ST_Intersects`, et al

4330, `postgis_restore` OOM when output piped to an intermediate process (Hugh Ranalli)

4322, Support for Proj 6+ API, bringing more accurate datum transforms and support for WKT projections

4153, `ST_Segmentize` now splits segments proportionally (Darafei Praliaskouski).

4162, `ST_DWithin` documentation examples for storing geometry and radius in table (Darafei Praliaskouski, github user Boscop).

4161 and #4294, `ST_AsMVTGeom`: Shortcut geometries smaller than the resolution (Raúl Marín)

4176, `ST_Intersects` supports `GEOMETRYCOLLECTION` (Darafei Praliaskouski)

4181, `ST_AsMVTGeom`: Avoid type changes due to validation (Raúl Marín)

4183, `ST_AsMVTGeom`: Drop invalid geometries after simplification (Raúl Marín)

4196, Have `postgis_extensions_upgrade()` package unpackaged extensions (Sandro Santilli)

4215, Use floating point compare in `ST_DumpAsPolygons` (Darafei Praliaskouski)

4155, Support for `GEOMETRYCOLLECTION`, `POLYGON`, `TIN`, `TRIANGLE` in `ST_LocateBetween` and `ST_LocateBetweenElevation` (Darafei Praliaskouski)

2767, Documentation for `AddRasterConstraint` optional parameters (Sunveer Singh)

- 4244, Avoid unaligned memory access in BOX2D\_out (Raúl Marín)
- 4139, Make mixed-dimension ND index build tree correctly (Darafei Praliaskouski, Arthur Lesuisse, Andrew Gierth, Raúl Marín)
- 4262, Document MULTISURFACE compatibility of ST\_LineToCurve (Steven Ottens)
- 4276, ST\_AsGeoJSON documentation refresh (Darafei Praliaskouski)
- 4292, ST\_AsMVT: parse JSON numeric values with decimals as doubles (Raúl Marín)
- 4300, ST\_AsMVTGeom: Always return the simplest geometry (Raúl Marín)
- 4301, ST\_Subdivide: fix endless loop on coordinates near coincident to bounds (Darafei Praliaskouski)
- 4289, ST\_AsMVTGeom: Transform coordinates space before clipping (Raúl Marín)
- 4272, Improve notice message when unable to compute stats (Raúl Marín)
- 4313, #4307, PostgreSQL 12 compatibility (Laurenz Albe, Raúl Marín)
- 4299, #4304, ST\_GeneratePoints is now VOLATILE. IMMUTABLE version with seed parameter added. (Mike Taves)
- 4278, ST\_3DDistance and ST\_3DIntersects now support Solid TIN and Solid POLYHEDRALSURFACE (Darafei Praliaskouski)
- 4348, ST\_AsMVTGeom (GEOS): Enforce validation at all times (Raúl Marín)
- 4295, Allow GEOMETRYCOLLECTION in ST\_Overlaps, ST\_Contains, ST\_ContainsProperly, ST\_Covers, ST\_CoveredBy, ST\_Crosses, ST\_Touches, ST\_Disjoint, ST\_Relate, ST\_Equals (Esteban Zimányi)
- 4340, ST\_Union aggregate now can handle more than 1 GB of geometries (Darafei Praliaskouski)
- 4378, Allow passing TINs as input to GEOS-backed functions (Darafei Praliaskouski)
- 4368, Reorder LWGEOM struct members to minimize extra padding (Raúl Marín)
- 4141, Use uint64 to handle row counts in the topology extension (Raúl Marín)
- 4412, Support ingesting rasters with NODATA=NaN (Darafei Praliaskouski)
- 4413, Raster tile size follows GeoTIFF block size on raster2pgsql -t auto (Darafei Praliaskouski)
- 4422, Modernize Python 2 code to get ready for Python 3 (Christian Clauss)
- 4352, Use CREATE OR REPLACE AGGREGATE for PG12+ (Raúl Marín)
- 4394, Allow FULL OUTER JOIN on geometry equality operator (Darafei Praliaskouski)
- 4441, Make GiST penalty friendly to multi-column indexes and build single-column ones faster. (Darafei Praliaskouski)
- 4403, Support for shp2pgsql ability to reproject with copy mode (-D) (Regina Obe)
- 4410, More descriptive error messages about SRID mismatch (Darafei Praliaskouski)
- 4399, TIN and Triangle output support in all output functions (Darafei Praliaskouski)
- 3719, Impose minimum number of segments per arc during linearization (Dan Baston / City of Helsinki, Raúl Marín)
- 4277, ST\_GeomFromGeoJSON now marks SRID=4326 by default as per RFC7946, ST\_AsGeoJSON sets SRID in JSON output if it differs from 4326. (Darafei Praliaskouski)
- 3979, postgis\_sfcgal\_noop() round trip function (Lucas C. Villa Real)
- 4328, ST\_3DIntersects for 2D TINs. (Darafei Praliaskouski)
- 4509, Update geocoder for tiger 2019 (Regina Obe)

## A.22 Release 3.0.0rc2

Release date: 2019/10/13

If compiling with PostgreSQL+JIT, LLVM >= 6 is required

Supported PostgreSQL versions for this release are: PostgreSQL 9.5 - PostgreSQL 12 GEOS >= 3.6. Additional features enabled if you running Proj6+ and/or PostgreSQL 12. Performance enhancements if running GEOS 3.8+

### A.22.1 Major highlights

4534, Fix leak in lwcurvepoly\_from\_wkb\_state (Raúl Marín)

4536, Fix leak in lwcollection\_from\_wkb\_state (Raúl Marín)

4537, Fix leak in WKT collection parser (Raúl Marín)

4535, WKB: Avoid buffer overflow (Raúl Marín)

## A.23 Release 3.0.0rc1

Release date: 2019/10/08

If compiling with PostgreSQL+JIT, LLVM  $\geq 6$  is required

Supported PostgreSQL versions for this release are: PostgreSQL 9.5 - PostgreSQL 12 GEOS  $\geq 3.6$ . Additional features enabled if you running Proj6+ and/or PostgreSQL 12. Performance enhancements if running GEOS 3.8+

### A.23.1 Major highlights

4519, Fix getSRIDbySRS crash (Raúl Marín)

4520, Use a clean environment when detecting C++ libraries (Raúl Marín)

Restore ST\_Union() aggregate signature so drop agg not required and re-work performance/size enhancement to continue to avoid using Array type during ST\_Union(), hopefully avoiding Array size limitations. (Paul Ramsey)

## A.24 Release 3.0.0beta1

Release date: 2019/09/28

If compiling with PostgreSQL+JIT, LLVM  $\geq 6$  is required

Supported PostgreSQL versions for this release are: PostgreSQL 9.5 - PostgreSQL 12 GEOS  $\geq 3.6$ . Additional features enabled if you running Proj6+ and/or PostgreSQL 12. Performance enhancements if running GEOS 3.8+

### A.24.1 Major highlights

4492, Fix ST\_Simplify ignoring the value of the 3rd parameter (Raúl Marín)

4494, Fix ST\_Simplify output having an outdated bbox (Raúl Marín)

4493, Fix ST\_RemoveRepeatedPoints output having an outdated bbox (Raúl Marín)

4495, Fix ST\_SnapToGrid output having an outdated bbox (Raúl Marín)

4496, Make ST\_Simplify(TRIANGLE) collapse if requested (Raúl Marín)

4501, Allow postgis\_tiger\_geocoder to be installable by non-super users (Regina Obe)

4503, Speed up the calculation of cartesian bbox (Raúl Marín)

4504, shp2pgsql -D not working with schema qualified tables (Regina Obe)

4505, Speed up conversion of geometries to/from GEOS (Dan Baston)

4507, Use GEOSMakeValid and GEOSBuildArea for GEOS 3.8+ (Dan Baston)

4491, Speed up ST\_RemoveRepeatedPoints (Raúl Marín)

4509, Update geocoder for tiger 2019 (Regina Obe)

4338, Census block level data (tabblock table) not loading (Regina Obe)

## A.25 Release 3.0.0alpha4

Release date: 2019/08/11

If compiling with PostgreSQL+JIT, LLVM  $\geq 6$  is required

Supported PostgreSQL versions for this release are: PostgreSQL 9.5 - PostgreSQL 12 GEOS  $\geq 3.6$ . Additional features enabled if you running Proj6+ and/or PostgreSQL 12

### A.25.1 Major highlights

- 4433, 32-bit hash fix (requires reindexing hash(geometry) indexes) (Raúl Marín)
- 4445, Fix a bug in geometry\_le (Raúl Marín)
- 4451, Fix the calculation of gserialized\_max\_header\_size (Raúl Marín)
- 4450, Speed up ST\_GeometryType (Raúl Marín)
- 4452, Add ST\_TileEnvelope() (Paul Ramsey)
- 4403, Support for shp2pgsql ability to reproject with copy mode (-D) (Regina Obe)
- 4417, Update spatial\_ref\_sys with new entries (Paul Ramsey)
- 4449, Speed up ST\_X, ST\_Y, ST\_Z and ST\_M (Raúl Marín)
- 4454, Speed up \_ST\_OrderingEquals (Raúl Marín)
- 4453, Speed up ST\_IsEmpty (Raúl Marín)
- 4271, postgis\_extensions\_upgrade() also updates after pg\_upgrade (Raúl Marín)
- 4466, Fix undefined behaviour in \_postgis\_gserialized\_stats (Raúl Marín)
- 4209, Handle NULL geometry values in pgsq2shp (Paul Ramsey)
- 4419, Use protobuf version to enable/disable mvt/geobuf (Paul Ramsey)
- 4437, Handle POINT EMPTY in shape loader/dumper (Paul Ramsey)
- 4456, add Rasbery Pi 32-bit jenkins bot for testing (Bruce Rindahl, Regina Obe)
- 4420, update path does not exists for address\_standardizer extension (Regina Obe)

## A.26 Release 3.0.0alpha3

Release date: 2019/07/01

If compiling with PostgreSQL+JIT, LLVM  $\geq 6$  is required

Supported PostgreSQL versions for this release are: PostgreSQL 9.5 - PostgreSQL 12 GEOS  $\geq 3.6$

### A.26.1 Major highlights

- 4414, Include version number in address\_standardizer lib (Raúl Marín)
- 4352, Use CREATE OR REPLACE AGGREGATE for PG12+ (Raúl Marín)
- 4334, Fix upgrade issues related to renamed parameters (Raúl Marín)
- 4388, AddRasterConstraints: Ignore NULLs when generating constraints (Raúl Marín)
- 4327, Avoid pfree'ing the result of getenv (Raúl Marín)
- 4406, Throw on invalid characters when decoding geohash (Raúl Marín)

4429, Avoid resource leaks with PROJ6 (Raúl Marín)

4372, PROJ6: Speed improvements (Raúl Marín)

3437, Speed up ST\_Intersects with Points (Raúl Marín)

4438, Update serialization to support extended flags area (Paul Ramsey)

4443, Fix wagyu configure dropping CPPFLAGS (Raúl Marín)

4440, Type lookups in FDW fail (Paul Ramsey)

4442, raster2pgsql now skips NODATA tiles. Use -k option if you still want them in database for some reason. (Darafei Praliaskouski)

4441, Make GiST penalty friendly to multi-column indexes and build single-column ones faster. (Darafei Praliaskouski)

## A.27 Release 3.0.0alpha2

Release date: 2019/06/02

If compiling with PostgreSQL+JIT, LLVM >= 6 is required

Supported PostgreSQL versions for this release are: PostgreSQL 9.5 - PostgreSQL 12 GEOS >= 3.6

### A.27.1 Major highlights

#4404, Fix selectivity issue with support functions (Paul Ramsey)

#4311, Make wagyu the default option to validate polygons. This option requires a C++11 compiler and will use CXXFLAGS (not CFLAGS). It is only enabled if built with MVT support (protobuf) Add `--without-wagyu` to disable this option and keep the behaviour from 2.5 (Raúl Marín)

#4198, Add ST\_ConstrainedDelaunayTriangles SFCGAL function (Darafei Praliaskouski)

## A.28 Release 3.0.0alpha1

Release date: 2019/05/26

If compiling with PostgreSQL+JIT, LLVM >= 6 is required

Supported PostgreSQL versions for this release are: PostgreSQL 9.5 - PostgreSQL 12 GEOS >= 3.6

### A.28.1 New Features

additional features enabled if you are running Proj6+

Read the NEWS file in the included tarball for more details

## A.29 Release 2.5.0

Release date: 2018/09/23

If compiling with PostgreSQL+JIT, LLVM >= 6 is required

Supported PostgreSQL versions for this release are: PostgreSQL 9.4 - PostgreSQL 12 (in development) GEOS >= 3.5

---

## A.29.1 New Features

#1847, spgist 2d and 3d support for PG 11+ (Esteban Zimányi and Arthur Lesuisse from Université Libre de Bruxelles (ULB), Darafei Praliaskouski)

#4056, ST\_FilterByM (Nicklas Avén)

#4050, ST\_ChaikinSmoothing (Nicklas Avén)

#3989, ST\_Buffer single sided option (Stephen Knox)

#3876, ST\_Angle function (Rémi Cura)

#3564, ST\_LineInterpolatePoints (Dan Baston)

#3896, PostGIS\_Extensions\_Upgrade() (Regina Obe)

#3913, Upgrade when creating extension from unpackaged (Sandro Santilli)

#2256, \_postgis\_index\_extent() for extent from index (Paul Ramsey)

#3176, Add ST\_OrientedEnvelope (Dan Baston)

#4029, Add ST\_QuantizeCoordinates (Dan Baston)

#4063, Optional false origin point for ST\_Scale (Paul Ramsey)

#4082, Add ST\_BandFileSize and ST\_BandFileTimestamp, extend ST\_BandMetadata (Even Rouault)

#2597, Add ST\_Grayscale (Bborie Park)

#4007, Add ST\_SetBandPath (Bborie Park)

#4008, Add ST\_SetBandIndex (Bborie Park)

## A.29.2 Breaking Changes

Upgrade scripts from multiple old versions are now all symlinks to a single upgrade script (Sandro Santilli)

#3944, Update to EPSG register v9.2 (Even Rouault)

#3927, Parallel implementation of ST\_AsMVT

#3925, Simplify geometry using map grid cell size before generating MVT

#3899, BTree sort order is now defined on collections of EMPTY and same-prefix geometries (Darafei Praliaskouski)

#3864, Performance improvement for sorting POINT geometries (Darafei Praliaskouski)

#3900, GCC warnings fixed, make -j is now working (Darafei Praliaskouski) - TopoGeo\_addLinestring robustness improvements (Sandro Santilli) #1855, #1946, #3718, #3838

#3234, Do not accept EMPTY points as topology nodes (Sandro Santilli)

#1014, Hashable geometry, allowing direct use in CTE signatures (Paul Ramsey)

#3097, Really allow MULTILINESTRING blades in ST\_Split() (Paul Ramsey)

#3942, geojson: Do not include private header for json-c >= 0.13 (Björn Esser)

#3954, ST\_GeometricMedian now supports point weights (Darafei Praliaskouski)

#3965, #3971, #3977, #4071 ST\_ClusterKMeans rewritten: better initialization, faster convergence, K=2 even faster (Darafei Praliaskouski)

#3982, ST\_AsEncodedPolyline supports LINESTRING EMPTY and MULTIPOINT EMPTY (Darafei Praliaskouski)

#3986, ST\_AsText now has second argument to limit decimal digits (Marc Ducobu, Darafei Praliaskouski)

#4020, Casting from box3d to geometry now returns correctly connected PolyhedralSurface (Matthias Bay)

#2508, ST\_OffsetCurve now works with collections (Darafei Praliaskouski)

- #4006, ST\_GeomFromGeoJSON support for json and jsonb as input (Paul Ramsey, Regina Obe)
- #4038, ST\_Subdivide now selects pivot for geometry split that reuses input vertices. (Darafei Praliaskouski)
- #4025, #4032 Fixed precision issue in ST\_ClosestPointOfApproach, ST\_DistanceCPA, and ST\_CPAWithin (Paul Ramsey, Darafei Praliaskouski)
- #4076, Reduce use of GEOS in topology implementation (Björn Harrtell)
- #4080, Add external raster band index to ST\_BandMetaData - Add Raster Tips section to Documentation for information about Raster behavior (e.g. Out-DB performance, maximum open files)
- #4084: Fixed wrong code-comment regarding front/back of BOX3D (Matthias Bay)
- #4060, #4094, PostgreSQL JIT support (Raúl Marín, Laurenz Albe)
- #3960, ST\_Centroid now uses lwgeom\_centroid (Darafei Praliaskouski)
- #4027, Remove duplicated code in lwgeom\_geos (Darafei Praliaskouski, Daniel Baston)
- #4115, Fix a bug that created MVTs with incorrect property values under parallel plans (Raúl Marín).
- #4120, ST\_AsMVTGeom: Clip using tile coordinates (Raúl Marín).
- #4132, ST\_Intersection on Raster now works without throwing TopologyException (Vinícius A.B. Schmidt, Darafei Praliaskouski)
- #4177, #4180 Support for PostgreSQL 12 dev branch (Laurenz Albe, Raúl Marín)
- #4156, ST\_ChaikinSmoothing: also smooth start/end point of polygon by default (Darafei Praliaskouski)

## A.30 Release 2.4.5

Release date: 2018/09/12

This is a bug fix and performance improvement release.

### A.30.1 Bug Fixes

- #4031, Survive to big MaxError tolerances passed to ST\_CurveToLine (Sandro Santilli)
- #4058, Fix infinite loop in linearization of a big radius small arc (Sandro Santilli)
- #4071, ST\_ClusterKMeans crash on NULL/EMPTY fixed (Darafei Praliaskouski)
- #4079, ensure St\_AsMVTGeom outputs CW oriented polygons (Paul Ramsey)
- #4070, use standard interruption error code on GEOS interruptions (Paul Ramsey)
- #3980, delay freeing input until processing complete (lucasvr)
- #4090, PG 11 support (Paul Ramsey, Raúl Marín)
- #4077, Serialization failure for particular empty geometry cases (Paul Ramsey)
- #3997, fix bug in lwgeom\_median and avoid division by zero (Raúl Marín)
- #4093, Inconsistent results from qsort callback (yugr)
- #4081, Geography DWithin() issues for certain cases (Paul Ramsey)
- #4105, Parallel build of tarball (Bas Couwenberg)
- #4163, MVT: Fix resource leak when the first geometry is NULL (Raúl Marín)

## A.31 Release 2.4.4

Release date: 2018/04/08

This is a bug fix and performance improvement release.

### A.31.1 Bug Fixes

- #3055, [raster] ST\_Clip() on a raster without band crashes the server (Regina Obe)
- #3942, geojson: Do not include private header for json-c >= 0.13 (Björn Esser)
- #3952, ST\_Transform fails in parallel mode (Paul Ramsey)
- #3978, Fix KNN when upgrading from 2.1 or older (Sandro Santilli)
- #4003, lwpoly\_construct\_circle: Avoid division by zero (Raúl Marín Rodríguez)
- #4004, Avoid memory exhaustion when building a btree index (Edmund Horner)
- #4016, proj 5.0.0 support (Raúl Marín Rodríguez)
- #4017, lwgeom lexer memory corruption (Peter E)
- #4020, Casting from box3d to geometry now returns correctly connected PolyhedralSurface (Matthias Bay)
- #4025, #4032 Incorrect answers for temporally "almost overlapping" ranges (Paul Ramsey, Darafei Praliaskouski)
- #4052, schema qualify several functions in geography (Regina Obe)
- #4055, ST\_ClusterIntersecting drops SRID (Daniel Baston)

### A.31.2 Enhancements

- #3946, Compile support for PostgreSQL 11 (Paul Ramsey)
- #3992, Use PKG\_PROG\_PKG\_CONFIG macro from pkg.m4 to detect pkg-config (Bas Couwenberg)
- #4044, Upgrade support for PostgreSQL 11 (Regina Obe)

## A.32 Release 2.4.3

Release date: 2018/01/17

This is a bug fix and performance improvement release.

### A.32.1 Bug Fixes and Enhancements

- #3713, Support encodings that happen to output a ` character
- #3827, Set configure default to not do interrupt testing, was causing false negatives for many people. (Regina Obe) revised to be standards compliant in #3988 (Greg Troxel)
- #3930, Minimum bounding circle issues on 32-bit platforms
- #3965, ST\_ClusterKMeans used to lose some clusters on initialization (Darafei Praliaskouski)
- #3956, Brin opclass object does not upgrade properly (Sandro Santilli)
- #3982, ST\_AsEncodedPolyline supports LINestring EMPTY and MULTIPOINT EMPTY (Darafei Praliaskouski)
- #3975, ST\_Transform runs query on spatial\_ref\_sys without schema qualification. Was causing restore issues. (Paul Ramsey)

## A.33 Release 2.4.2

Release date: 2017/11/15

This is a bug fix and performance improvement release.

### A.33.1 Bug Fixes and Enhancements

#3917, Fix zcta5 load

#3667, Fix for bug in geography ST\_Segmentize

#3926, Add missing 2.2.6 and 2.3.4 upgrade paths (Muhammad Usama)

## A.34 Release 2.4.1

Release date: 2017/10/18

This is a bug fix and performance improvement release.

### A.34.1 Bug Fixes and Enhancements

#3864, Fix memory leaks in BTREE operators

#3869, Fix build with "gold" linker

#3845, Gracefully handle short-measure issue

#3871, Performance tweak for geometry cmp function

#3879, Division by zero in some arc cases

#3878, Single defn of signum in header

#3880, Undefined behaviour in TYPMOD\_GET\_SRID

#3875, Fix undefined behaviour in shift operation

#3864, Performance improvements for b-tree geometry sorts

#3874, lw\_dist2d\_pt\_arc division by zero

#3882, undefined behaviour in zigzag with negative inputs

#3891, undefined behaviour in pointarray\_to\_encoded\_polyline

#3895, throw error on malformed WKB input

#3886, fix rare missing boxes in geometry subdivision

#3907, Allocate enough space for all possible GBOX string outputs (Raúl Marín Rodríguez)

## A.35 Release 2.4.0

Release date: 2017/09/30

---

### A.35.1 New Features

- #3822, Have `postgis_full_version()` also show and check version of PostgreSQL the scripts were built against (Sandro Santilli)
- #2411, curves support in `ST_Reverse` (Sandro Santilli)
- #2951, `ST_Centroid` for geography (Danny Götte)
- #3788, Allow `postgis_restore.pl` to work on directory-style (-Fd) dumps (Roger Crew)
- #3772, Direction agnostic `ST_CurveToLine` output (Sandro Santilli / KKGeo)
- #2464, `ST_CurveToLine` with `MaxError` tolerance (Sandro Santilli / KKGeo)
- #3599, Geobuf output support via `ST_AsGeobuf` (Björn Harrtell)
- #3661, Mapbox vector tile output support via `ST_AsMVT` (Björn Harrtell / CartoDB)
- #3689, Add orientation checking and forcing functions (Dan Baston)
- #3753, Gist penalty speed improvements for 2D and ND points (Darafei Praliaskouski, Andrey Borodin)
- #3677, `ST_FrechetDistance` (Shinichi Sugiyama)
- Most aggregates (raster and geometry), and all stable / immutable (raster and geometry) marked as parallel safe
- #2249, `ST_MakeEmptyCoverage` for raster (David Zwarg, ainomieli)
- #3709, Allow signed distance for `ST_Project` (Darafei Praliaskouski)
- #524, Covers support for polygon on polygon, line on line, point on line for geography (Danny Götte)

### A.35.2 Enhancements and Fixes

Many corrections to docs and several translations almost complete. Andreas Schild who provided many corrections to core docs. PostGIS Japanese translation team first to reach completion of translation.

Support for PostgreSQL 10

Preliminary support for PostgreSQL 11

- #3645, Avoid loading logically deleted records from shapefiles
- #3747, Add `zip4` and `address_alphanumeric` as attributes to `norm_addy_tiger_geocoder` type.
- #3748, `address_standardizer` lookup tables update so `page_normalize_address` better standardizes abbreviations
- #3647, better handling of nodding in `ST_Node` using `GEOSNode` (Wouter Geraedts)
- #3684, Update to EPSG register v9 (Even Rouault)
- #3830, Fix initialization of incompatible type ( $\geq 9.6$ ) `address_standardizer`
- #3662, Make `shp2pgsql` work in debug mode by sending debug to `stderr`
- #3405, Fixed memory leak in `lwgeom_to_points`
- #3832, Support wide integer fields as `int8` in `shp2pgsql`
- #3841, Deterministic sorting support for empty geometries in `btree` geography
- #3844, Make `=` operator a strict equality test, and `<` `>` to rough "spatial sorting"
- #3855, `ST_AsTWKB` memory and speed improvements

### A.35.3 Breaking Changes

Dropped support for PostgreSQL 9.2.

#3810, GEOS 3.4.0 or above minimum required to compile

Most aggregates now marked as parallel safe, which means most aggs have to be dropped / recreated. If you have views that utilize PostGIS aggs, you'll need to drop before upgrade and recreate after upgrade

#3578, ST\_NumInteriorRings(POLYGON EMPTY) now returns 0 instead of NULL

\_ST\_DumpPoints removed, was no longer needed after PostGIS 2.1.0 when ST\_DumpPoints got reimplemented in C

B-Tree index operators < = > changed to provide better spatial locality on sorting and have expected behavior on GROUP BY. If you have btree index for geometry or geography, you need to REINDEX it, or review if it was created by accident and needs to be replaced with GiST index. If your code relies on old left-to-right box compare ordering, update it to use << >> operators.

## A.36 Release 2.3.3

Release date: 2017/07/01

This is a bug fix and performance improvement release.

### A.36.1 Bug Fixes and Enhancements

#3777, GROUP BY anomaly with empty geometries

#3711, Azimuth error upon adding 2.5D edges to topology

#3726, PDF manual from dblatex renders fancy quotes for programlisting (Mike Toews)

#3738, raster: Using -s without -Y in raster2pgsql transforms raster data instead of setting srid

#3744, ST\_Subdivide loses subparts of inverted geometries (Darafei Praliaskouski Komzpa)

#3750, @ and ~ operator not always schema qualified in geometry and raster functions. Causes restore issues. (Shane StClair of Axiom Data Science)

#3682, Strange fieldlength for boolean in result of pgsq2shp

#3701, Escape double quotes issue in pgsq2shp

#3704, ST\_AsX3D crashes on empty geometry

#3730, Change ST\_Clip from Error to Notice when ST\_Clip can't compute a band

## A.37 Release 2.3.2

Release date: 2017/01/31

This is a bug fix and performance improvement release.

### A.37.1 Bug Fixes and Enhancements

#3418, KNN recheck in 9.5+ fails with index returned tuples in wrong order

#3675, Relationship functions not using an index in some cases

#3680, PostGIS upgrade scripts missing GRANT for views

#3683, Unable to update postgis after postgres pg\_upgrade going from < 9.5 to pg > 9.4

#3688, ST\_AsLatLonText: round minutes

## A.38 Release 2.3.1

Release date: 2016/11/28

This is a bug fix and performance improvement release.

### A.38.1 Bug Fixes and Enhancements

#1973, st\_concavehull() returns sometimes empty geometry collection Fix from gde

#3501, add raster constraint max extent exceeds array size limit for large tables

#3643, PostGIS not building on latest OSX XCode

#3644, Deadlock on interrupt

#3650, Mark ST\_Extent, ST\_3DExtent and ST\_Mem\* agg functions as parallel safe so they can be parallelized

#3652, Crash on Collection(MultiCurve())

#3656, Fix upgrade of aggregates from 2.2 or lower version

#3659, Crash caused by raster GUC define after CREATE EXTENSION using wrong memory context. (manaem)

#3665, Index corruption and memory leak in BRIN indexes patch from Julien Rouhaud (Dalibo)

#3667, geography ST\_Segmentize bug patch from Hugo Mercier (Oslandia)

## A.39 Release 2.3.0

Release date: 2016/09/26

This is a new feature release, with new functions, improved performance, all relevant bug fixes from PostGIS 2.2.3, and other goodies.

### A.39.1 Important / Breaking Changes

#3466, Casting from box3d to geometry now returns a 3D geometry (Julien Rouhaud of Dalibo)

#3396, ST\_EstimatedExtent, throw WARNING instead of ERROR (Regina Obe)

### A.39.2 New Features

Add support for custom TOC in postgis\_restore.pl (Christoph Moench-Tegeder)

Add support for negative indexing in ST\_PointN and ST\_SetPoint (Rémi Cura)

Add parameters for geography ST\_Buffer (Thomas Bonfort)

TopoGeom\_addElement, TopoGeom\_remElement (Sandro Santilli)

populate\_topology\_layer (Sandro Santilli)

#454, ST\_WrapX and lwgeom\_wrapx (Sandro Santilli)

#1758, ST\_Normalize (Sandro Santilli)

#2236, shp2pgsql -d now emits "DROP TABLE IF EXISTS"

#2259, ST\_VoronoiPolygons and ST\_VoronoiLines (Dan Baston)

#2841 and #2996, ST\_MinimumBoundingRadius and new ST\_MinimumBoundingCircle implementation using Welzl's algorithm (Dan Baston)

- #2991, Enable ST\_Transform to use PROJ.4 text (Mike Toews)
- #3059, Allow passing per-dimension parameters in ST\_Expand (Dan Baston)
- #3339, ST\_GeneratePoints (Paul Ramsey)
- #3362, ST\_ClusterDBSCAN (Dan Baston)
- #3364, ST\_GeometricMedian (Dan Baston)
- #3391, Add table inheritance support in ST\_EstimatedExtent (Alessandro Pasotti)
- #3424, ST\_MinimumClearance (Dan Baston)
- #3428, ST\_Points (Dan Baston)
- #3465, ST\_ClusterKMeans (Paul Ramsey)
- #3469, ST\_MakeLine with MULTIPOINTs (Paul Norman)
- #3549, Support PostgreSQL 9.6 parallel query mode, as far as possible (Paul Ramsey, Regina Obe)
- #3557, Geometry function costs based on query stats (Paul Norman)
- #3591, Add support for BRIN indexes. PostgreSQL 9.4+ required. (Giuseppe Broccolo of 2nd Quadrant, Julien Rouhaud and Ronan Dunklau of Dalibo)
- #3496, Make postgis non-relocateable for extension install, schema qualify calls in functions (Regina Obe) Should resolve once and for all for extensions #3494, #3486, #3076
- #3547, Update tiger geocoder to support TIGER 2016 and to support both http and ftp.
- #3613, Segmentize geography using equal length segments (Hugo Mercier of Oslandia)

### A.39.3 Bug Fixes

All relevant bug fixes from PostGIS 2.2.3

- #2841, ST\_MinimumBoundingCircle not covering original
- #3604, pgcommon/Makefile.in orders CFLAGS incorrectly leading to wrong liblwgeom.h (Greg Troxel)

### A.39.4 Performance Enhancements

- #75, Enhancement to PIP short circuit (Dan Baston)
- #3383, Avoid deserializing small geometries during index operations (Dan Baston)
- #3400, Minor optimization of PIP routines (Dan Baston)
- Make adding a line to topology interruptible (Sandro Santilli)
- Documentation updates from Mike Toews

## A.40 Release 2.2.2

Release date: 2016/03/22

This is a bug fix and performance improvement release.

---

## A.40.1 New Features

- #3463, Fix crash on face-collapsing edge change
- #3422, Improve ST\_Split robustness on standard precision double systems (arm64, ppc64el, s390c, powerpc, ...)
- #3427, Update spatial\_ref\_sys to EPSG version 8.8
- #3433, ST\_ClusterIntersecting incorrect for MultiPoints
- #3435, ST\_AsX3D fix rendering of concave geometries
- #3436, memory handling mistake in parray\_clone\_deep
- #3437, ST\_Intersects incorrect for MultiPoints
- #3461, ST\_GeomFromKML crashes Postgres when there are innerBoundaryIs and no outerBoundaryIs
- #3429, upgrading to 2.3 or from 2.1 can cause loop/hang on some platforms
- #3460, ST\_ClusterWithin 'Tolerance not defined' error after upgrade
- #3490, Raster data restore issues, materialized views. Scripts postgis\_proc\_set\_search\_path.sql, rtpostgis\_proc\_set\_search\_path.sql refer to [http://postgis.net/docs/manual-2.2/RT\\_FAQ.html#faq\\_raster\\_data\\_not\\_restore](http://postgis.net/docs/manual-2.2/RT_FAQ.html#faq_raster_data_not_restore)
- #3426, failing POINT EMPTY tests on fun architectures

## A.41 Release 2.2.1

Release date: 2016/01/06

This is a bug fix and performance improvement release.

### A.41.1 New Features

- #2232, avoid accumulated error in SVG rounding
- #3321, Fix performance regression in topology loading
- #3329, Fix robustness regression in TopoGeo\_addPoint
- #3349, Fix installation path of postgis\_topology scripts
- #3351, set endnodes isolation on ST\_RemoveIsoEdge (and lwt\_RemIsoEdge)
- #3355, geography ST\_Segmentize has geometry bbox
- #3359, Fix toTopoGeom loss of low-id primitives from TopoGeometry definition
- #3360, \_raster\_constraint\_info\_scale invalid input syntax
- #3375, crash in repeated point removal for collection(point)
- #3378, Fix handling of hierarchical TopoGeometries in presence of multiple topologies
- #3380, #3402, Decimate lines on topology load
- #3388, #3410, Fix missing end-points in ST\_Removepoints
- #3389, Buffer overflow in lwgeom\_to\_geojson
- #3390, Compilation under Alpine Linux 3.2 gives an error when compiling the postgis and postgis\_topology extension
- #3393, ST\_Area NaN for some polygons
- #3401, Improve ST\_Split robustness on 32bit systems
- #3404, ST\_ClusterWithin crashes backend

#3407, Fix crash on splitting a face or an edge defining multiple TopoGeometry objects  
#3411, Clustering functions not using spatial index  
#3412, Improve robustness of snapping step in TopoGeo\_addLinestring  
#3415, Fix OSX 10.9 build under pkgsrc  
Fix memory leak in lwt\_ChangeEdgeGeom [liblwgeom]

## A.42 Release 2.2.0

Release date: 2015/10/07

This is a new feature release, with new functions, improved performance, and other goodies.

### A.42.1 New Features

Topology API in liblwgeom (Sandro Santilli / Regione Toscana - SITA)

New lwgeom\_unaryunion method in liblwgeom

New lwgeom\_linemerge method in liblwgeom

New lwgeom\_is\_simple method in liblwgeom

#3169, Add SFCGAL 1.1 support: add ST\_3DDifference, ST\_3DUnion, ST\_Volume, ST\_MakeSolid, ST\_IsSolid (Vincent Mora / Oslandia)

#3169, ST\_ApproximateMedialAxis (Sandro Santilli)

ST\_CPAWithin (Sandro Santilli / Boundless)

Add |= operator with CPA semantic and KNN support with PostgreSQL 9.5+ (Sandro Santilli / Boundless)

#3131, KNN support for the geography type (Paul Ramsey / CartoDB)

#3023, ST\_ClusterIntersecting / ST\_ClusterWithin (Dan Baston)

#2703, Exact KNN results for all geometry types, aka "KNN re-check" (Paul Ramsey / CartoDB)

#1137, Allow a tolerance value in ST\_RemoveRepeatedPoints (Paul Ramsey / CartoDB)

#3062, Allow passing M factor to ST\_Scale (Sandro Santilli / Boundless)

#3139, ST\_BoundingDiagonal (Sandro Santilli / Boundless)

#3129, ST\_IsValidTrajectory (Sandro Santilli / Boundless)

#3128, ST\_ClosestPointOfApproach (Sandro Santilli / Boundless)

#3152, ST\_DistanceCPA (Sandro Santilli / Boundless)

Canonical output for index key types

ST\_SwapOrdinates (Sandro Santilli / Boundless)

#2918, Use GeographicLib functions for geodetics (Mike Toews)

#3074, ST\_Subdivide to break up large geometry (Paul Ramsey / CartoDB)

#3040, KNN GiST index based centroid (<<<->>) n-D distance operators (Sandro Santilli / Boundless)

Interruptibility API for liblwgeom (Sandro Santilli / CartoDB)

#2939, ST\_ClipByBox2D (Sandro Santilli / CartoDB)

#2247, ST\_Retile and ST\_CreateOverview: in-db raster overviews creation (Sandro Santilli / Vizzuality)

#899, -m shp2pgsql attribute names mapping -m switch (Regina Obe / Sandro Santilli)

#1678, Added GUC postgis.gdal\_datapath to specify GDAL config variable GDAL\_DATA

#2843, Support reprojection on raster import (Sandro Santilli / Vizzuality)

#2349, Support for encoded\_polyline input/output (Kashif Rasul)

#2159, report libjson version from postgis\_full\_version()

#2770, ST\_MemSize(raster)

Add postgis\_noop(raster)

Added missing variants of ST\_TPI(), ST\_TRI() and ST\_Roughness()

Added GUC postgis.gdal\_enabled\_drivers to specify GDAL config variable GDAL\_SKIP

Added GUC postgis.enable\_outdb\_rasters to enable access to rasters with out-db bands

#2387, address\_standardizer extension as part of PostGIS (Stephen Woodbridge / imaptools.com, Walter Sinclair, Regina Obe)

#2816, address\_standardizer\_data\_us extension provides reference lex,gaz,rules for address\_standardizer (Stephen Woodbridge / imaptools.com, Walter Sinclair, Regina Obe)

#2341, New mask parameter for ST\_MapAlgebra

#2397, read encoding info automatically in shapefile loader

#2430, ST\_ForceCurve

#2565, ST\_SummaryStatsAgg()

#2567, ST\_CountAgg()

#2632, ST\_AsGML() support for curved features

#2652, Add --upgrade-path switch to run\_test.pl

#2754, sfcgal wrapped as an extension

#2227, Simplification with Visvalingam-Whyatt algorithm ST\_SimplifyVW, ST\_SetEffectiveArea (Nicklas Avén)

Functions to encode and decode TWKB ST\_AsTWKB, ST\_GeomFromTWKB (Paul Ramsey / Nicklas Avén / CartoDB)

## A.42.2 Enhancements

#3223, Add memcmp short-circuit to ST\_Equals (Daniel Baston)

#3227, Tiger geocoder upgraded to support Tiger 2015 census

#2278, Make liblwgeom compatible between minor releases

#897, ST\_AsX3D support for GeoCoordinates and systems "GD" "WE" ability to flip x/y axis (use option = 2, 3)

ST\_Split: allow splitting lines by multilines, multipoints and (multi)polygon boundaries

#3070, Simplify geometry type constraint

#2839, Implement selectivity estimator for functional indexes, speeding up spatial queries on raster tables. (Sandro Santilli / Vizzuality)

#2361, Added spatial\_index column to raster\_columns view

#2390, Testsuite for pgsq2shp

#2527, Added -k flag to raster2pgsql to skip checking that band is NODATA

#2616, Reduce text casts during topology building and export

#2717, support startpoint, endpoint, pointn, numpoints for compoundcurve

#2747, Add support for GDAL 2.0

#2754, SFCGAL can now be installed with CREATE EXTENSION (Vincent Mora @ Oslandia)

- #2828, Convert ST\_Envelope(raster) from SQL to C
- #2829, Shortcut ST\_Clip(raster) if geometry fully contains the raster and no NODATA specified
- #2906, Update tiger geocoder to handle tiger 2014 data
- #3048, Speed up geometry simplification (J.Santana @ CartoDB)
- #3092, Slow performance of geometry\_columns with many tables

## A.43 Release 2.1.8

Release date: 2015-07-07

This is a critical bug fix release.

### A.43.1 Bug Fixes

- #3159, do not force a bbox cache on ST\_Affine
  - #3018, GROUP BY geography sometimes returns duplicate rows
  - #3084, shp2pgsql - illegal number format when specific system locale set
  - #3094, Malformed GeoJSON inputs crash backend
  - #3104, st\_asgml introduces random characters in ID field
  - #3155, Remove liblwgeom.h on make uninstall
  - #3177, gserialized\_is\_empty cannot handle nested empty cases
- Fix crash in ST\_LineLocatePoint

## A.44 Release 2.1.7

Release date: 2015-03-30

This is a critical bug fix release.

### A.44.1 Bug Fixes

- #3086, ST\_DumpValues() crashes backend on cleanup with invalid band indexes
- #3088, Do not (re)define strcasestr in a liblwgeom.h
- #3094, Malformed GeoJSON inputs crash backend

## A.45 Release 2.1.6

Release date: 2015-03-20

This is a bug fix and performance improvement release.

### A.45.1 Enhancements

- #3000, Ensure edge splitting and healing algorithms use indexes
  - #3048, Speed up geometry simplification (J.Santana @ CartoDB)
  - #3050, Speed up geometry type reading (J.Santana @ CartoDB)
-

## A.45.2 Bug Fixes

- #2941, allow geography columns with SRID other than 4326
- #3069, small objects getting inappropriately fluffed up w/ boxes
- #3068, Have postgis\_typmod\_dims return NULL for unconstrained dims
- #3061, Allow duplicate points in JSON, GML, GML ST\_GeomFrom\* functions
- #3058, Fix ND-GiST picksplit method to split on the best plane
- #3052, Make operators <-> and <#> available for PostgreSQL < 9.1
- #3045, Fix dimensionality confusion in &&& operator
- #3016, Allow unregistering layers of corrupted topologies
- #3015, Avoid exceptions from TopologySummary
- #3020, ST\_AddBand out-db bug where height using width value
- #3031, Allow restore of Geometry(Point) tables dumped with empties in them

## A.46 Release 2.1.5

Release date: 2014-12-18

This is a bug fix and performance improvement release.

### A.46.1 Enhancements

- #2933, Speedup construction of large multi-geometry objects

### A.46.2 Bug Fixes

- #2947, Fix memory leak in lwgeom\_make\_valid for single-component collection input
- #2949, Fix memory leak in lwgeom\_mindistance2d for curve input
- #2931, BOX representation is case sensitive
- #2942, PostgreSQL 9.5 support
- #2953, 2D stats not generated when Z/M values are extreme
- #3009, Geography cast may effect underlying tuple

## A.47 Release 2.1.4

Release date: 2014-09-10

This is a bug fix and performance improvement release.

### A.47.1 Enhancements

- #2745, Speedup ST\_Simplify calls against points
  - #2747, Support for GDAL 2.0
  - #2749, Make rtpostgis\_upgrade\_20\_21.sql ACID
  - #2811, Do not specify index names when loading shapefiles/rasters
  - #2829, Shortcut ST\_Clip(raster) if geometry fully contains the raster and no NODATA specified
  - #2895, Raise cost of ST\_ConvexHull(raster) to 300 for better query plans
-

## A.47.2 Bug Fixes

#2605, armel: `_ST_Covers()` returns true for point in hole

#2911, Fix output scale on `ST_Rescale/ST_Resample/ST_Resize` of rasters with scale 1/-1 and offset 0/0.

Fix crash in `ST_Union(raster)`

#2704, `ST_GeomFromGML()` does not work properly with array of `gml:pos` (Even Roualt)

#2708, `updategeometrysruid` doesn't update `sruid` check when schema not specified. Patch from Marc Jansen

#2720, `lwpoly_add_ring` should update `maxrings` after `realloc`

#2759, Fix `postgis_restore.pl` handling of multiline object comments embedding sql comments

#2774, fix undefined behavior in `ptarray_calculate_gbox_geodetic`

Fix potential memory fault in `ST_MakeValid`

#2784, Fix handling of bogus argument to `--with-sfcgal`

#2772, Premature memory free in `RASTER_getBandPath (ST_BandPath)`

#2755, Fix regressions tests against all versions of SFCGAL

#2775, `lwline_from_lwmpoint` leaks memory

#2802, `ST_MapAlgebra` checks for valid callback function return value

#2803, `ST_MapAlgebra` handles no `userarg` and `STRICT` callback function

#2834, `ST_Estimated_Extent` and `mixedCase` table names (regression bug)

#2845, Bad geometry created from `ST_AddPoint`

#2870, Binary insert into geography column results geometry being inserted

#2872, make install builds documentation (Greg Troxell)

#2819, find `isfinite` or replacement on Centos5 / Solaris

#2899, `geocode limit 1` not returning best answer (tiger geocoder)

#2903, Unable to compile on FreeBSD

#2927 `reverse_geocode` not filling in direction prefix (tiger geocoder) get rid of deprecated `ST_Line_Locate_Point` called

## A.48 Release 2.1.3

Release date: 2014/05/13

This is a bug fix and security release.

### A.48.1 Important changes

Starting with this version offline raster access and use of GDAL drivers are disabled by default.

An environment variable is introduced to allow for enabling specific GDAL drivers: `POSTGIS_GDAL_ENABLED_DRIVERS`. By default, all GDAL drivers are disabled

An environment variable is introduced to allow for enabling out-db raster bands: `POSTGIS_ENABLE_OUTDB_RASTERS`. By default, out-db raster bands are disabled

The environment variables must be set for the PostgreSQL process, and determines the behavior of the whole cluster.

## A.48.2 Bug Fixes

#2697, invalid GeoJSON Polygon input crashes server process

#2700, Fix dumping of higher-dimension datasets with null rows

#2706, ST\_DumpPoints of EMPTY geometries crashes server

## A.49 Release 2.1.2

Release date: 2014/03/31

This is a bug fix release, addressing issues that have been filed since the 2.1.1 release.

### A.49.1 Bug Fixes

#2666, Error out at configure time if no SQL preprocessor can be found

#2534, st\_distance returning incorrect results for large geographies

#2539, Check for json-c/json.h presence/usability before json/json.h

#2543, invalid join selectivity error from simple query

#2546, GeoJSON with string coordinates parses incorrectly

#2547, Fix ST\_Simplify(TopoGeometry) for hierarchical topogeoms

#2552, Fix NULL raster handling in ST\_AsPNG, ST\_AsTIFF and ST\_AsJPEG

#2555, Fix parsing issue of range arguments of ST\_Reclass

#2556, geography ST\_Intersects results depending on insert order

#2580, Do not allow installing postgis twice in the same database

#2589, Remove use of unnecessary void pointers

#2607, Cannot open more than 1024 out-db files in one process

#2610, Ensure face splitting algorithm uses the edge index

#2615, EstimatedExtent (and hence, underlying stats) gathering wrong bbox

#2619, Empty rings array in GeoJSON polygon causes crash

#2634, regression in sphere distance code

#2638, Geography distance on M geometries sometimes wrong

#2648, #2653, Fix topology functions when "topology" is not in search\_path

#2654, Drop deprecated calls from topology

#2655, Let users without topology privileges call postgis\_full\_version()

#2674, Fix missing operator = and hash\_raster\_ops opclass on raster

#2675, #2534, #2636, #2634, #2638, Geography distance issues with tree optimization

### A.49.2 Enhancements

#2494, avoid memcopy in GiST index (hayamiz)

#2560, soft upgrade: avoid drop/recreate of aggregates that hadn't changed

## A.50 Release 2.1.1

Release date: 2013/11/06

This is a bug fix release, addressing issues that have been filed since the 2.1.0 release.

### A.50.1 Important Changes

[#2514](#), Change raster license from GPL v3+ to v2+, allowing distribution of PostGIS Extension as GPLv2.

### A.50.2 Bug Fixes

[#2396](#), Make regression tests more endian-agnostic

[#2434](#), Fix ST\_Intersection(geog,geog) regression in rare cases

[#2454](#), Fix behavior of ST\_PixelAsXXX functions regarding exclude\_nodata\_value parameter

[#2489](#), Fix upgrades from 2.0 leaving stale function signatures

[#2525](#), Fix handling of SRID in nested collections

[#2449](#), Fix potential infinite loop in index building

[#2493](#), Fix behavior of ST\_DumpValues when passed an empty raster

[#2502](#), Fix postgis\_topology\_scripts\_installed() install schema

[#2504](#), Fix segfault on bogus pgsqldata call

[#2512](#), Support for foreign tables and materialized views in raster\_columns and raster\_overviews

### A.50.3 Enhancements

[#2478](#), support for tiger 2013

[#2463](#), support for exact length calculations on arc geometries

## A.51 Release 2.1.0

Release date: 2013/08/17

This is a minor release addressing both bug fixes and performance and functionality enhancements addressing issues since 2.0.3 release. If you are upgrading from 2.0+, only a soft upgrade is required. If you are upgrading from 1.5 or earlier, a hard upgrade is required.

### A.51.1 Important / Breaking Changes

[#1653](#), Removed srid parameter from ST\_Resample(raster) and variants with reference raster no longer apply reference raster's SRID.

[#1962](#) ST\_Segmentize - As a result of the introduction of geography support, The construct: `SELECT ST_Segmentize('LINESTRING(2 3 4)', 0.5);` will result in ambiguous function error

[#2026](#), ST\_Union(raster) now unions all bands of all rasters

[#2089](#), liblwgeom: lwgeom\_set\_handlers replaces lwgeom\_init\_allocators.

[#2150](#), regular\_blocking is no longer a constraint. column of same name in raster\_columns now checks for existence of spatially\_unique and coverage\_tile constraints

`ST_Intersects(raster, geometry)` behaves in the same manner as `ST_Intersects(geometry, raster)`.

point variant of `ST_SetValue(raster)` previously did not check SRID of input geometry and raster.

`ST_Hillshade` parameters azimuth and altitude are now in degrees instead of radians.

`ST_Slope` and `ST_Aspect` return pixel values in degrees instead of radians.

**#2104**, `ST_World2RasterCoord`, `ST_World2RasterCoordX` and `ST_World2RasterCoordY` renamed to `ST_WorldToRasterCoord`, `ST_WorldToRasterCoordX` and `ST_WorldToRasterCoordY`. `ST_Raster2WorldCoord`, `ST_Raster2WorldCoordX` and `ST_Raster2WorldCoordY` renamed to `ST_RasterToWorldCoord`, `ST_RasterToWorldCoordX` and `ST_RasterToWorldCoordY`

`ST_Estimated_Extent` renamed to `ST_EstimatedExtent`

`ST_Line_Interpolate_Point` renamed to `ST_LineInterpolatePoint`

`ST_Line_Substring` renamed to `ST_LineSubstring`

`ST_Line_Locate_Point` renamed to `ST_LineLocatePoint`

`ST_Force_XXX` renamed to `ST_ForceXXX`

`ST_MapAlgebraFctNgb` and 1 and 2 raster variants of `ST_MapAlgebraFct`. Use `ST_MapAlgebra` instead

1 and 2 raster variants of `ST_MapAlgebraExpr`. Use expression variants of `ST_MapAlgebra` instead

## A.51.2 New Features

- Refer to [http://postgis.net/docs/manual-2.1/PostGIS\\_Special\\_Functions\\_Index.html#NewFunctions\\_2\\_1](http://postgis.net/docs/manual-2.1/PostGIS_Special_Functions_Index.html#NewFunctions_2_1) for complete list of new functions

**#310**, `ST_DumpPoints` converted to a C function (Nathan Wagner) and much faster

**#739**, `UpdateRasterSRID()`

**#945**, improved join selectivity, N-D selectivity calculations, user accessible selectivity and stats reader functions for testing (Paul Ramsey / OpenGeo)

`toTopoGeom` with `TopoGeometry` sink (Sandro Santilli / Vizzuality)

`clearTopoGeom` (Sandro Santilli / Vizzuality)

`ST_Segmentize(geography)` (Paul Ramsey / OpenGeo)

`ST_DelaunayTriangles` (Sandro Santilli / Vizzuality)

`ST_NearestValue`, `ST_Neighborhood` (Bborie Park / UC Davis)

`ST_PixelAsPoint`, `ST_PixelAsPoints` (Bborie Park / UC Davis)

`ST_PixelAsCentroid`, `ST_PixelAsCentroids` (Bborie Park / UC Davis)

`ST_Raster2WorldCoord`, `ST_World2RasterCoord` (Bborie Park / UC Davis)

Additional raster/raster spatial relationship functions (`ST_Contains`, `ST_ContainsProperly`, `ST_Covers`, `ST_CoveredBy`, `ST_Disjoint`, `ST_Overlaps`, `ST_Touches`, `ST_Within`, `ST_DWithin`, `ST_DFullyWithin`) (Bborie Park / UC Davis)

Added array variants of `ST_SetValues()` to set many pixel values of a band in one call (Bborie Park / UC Davis)

**#1293**, `ST_Resize(raster)` to resize rasters based upon width/height

**#1627**, package `tiger_geocoder` as a PostgreSQL extension

**#1643**, **#2076**, Upgrade tiger geocoder to support loading tiger 2011 and 2012 (Regina Obe / Paragon Corporation) Funded by Hunter Systems Group

GEOMETRYCOLLECTION support for `ST_MakeValid` (Sandro Santilli / Vizzuality)

**#1709**, `ST_NotSameAlignmentReason(raster, raster)`

**#1818**, `ST_GeomFromGeoHash` and friends (Jason Smith (darkpanda))

#1856, reverse geocoder rating setting for prefer numbered highway name

ST\_PixelOfValue (Bborie Park / UC Davis)

Casts to/from PostgreSQL geotypes (point/path/polygon).

Added geomval array variant of ST\_SetValues() to set many pixel values of a band using a set of geometries and corresponding values in one call (Bborie Park / UC Davis)

ST\_Tile(raster) to break up a raster into tiles (Bborie Park / UC Davis)

#1895, new r-tree node splitting algorithm (Alex Korotkov)

#2011, ST\_DumpValues to output raster as array (Bborie Park / UC Davis)

#2018, ST\_Distance support for CircularString, CurvePolygon, MultiCurve, MultiSurface, CompoundCurve

#2030, n-raster (and n-band) ST\_MapAlgebra (Bborie Park / UC Davis)

#2193, Utilize PASC parser as drop in replacement for tiger normalizer (Steve Woodbridge, Regina Obe)

#2210, ST\_MinConvexHull(raster)

lwgeom\_from\_geojson in liblwgeom (Sandro Santilli / Vizzuality)

#1687, ST\_Simplify for TopoGeometry (Sandro Santilli / Vizzuality)

#2228, TopoJSON output for TopoGeometry (Sandro Santilli / Vizzuality)

#2123, ST\_FromGDALRaster

#613, ST\_SetGeoReference with numerical parameters instead of text

#2276, ST\_AddBand(raster) variant for out-db bands

#2280, ST\_Summary(raster)

#2163, ST\_TPI for raster (Nathaniel Clay)

#2164, ST\_TRI for raster (Nathaniel Clay)

#2302, ST\_Roughness for raster (Nathaniel Clay)

#2290, ST\_ColorMap(raster) to generate RGBA bands

#2254, Add SFCGAL backend support. (Backend selection through postgis.backend var) Functions available both through GEOS or SFCGAL: ST\_Intersects, ST\_3DIntersects, ST\_Intersection, ST\_Area, ST\_Distance, ST\_3DDistance New functions available only with SFCGAL backend: ST\_3DIntersection, ST\_Tesselate, ST\_3DArea, ST\_Extrude, ST\_ForceLHR ST\_Orientation, ST\_Minkowski, ST\_StraightSkeleton postgis\_sfcgal\_version New function available in PostGIS: ST\_ForceSFS (Olivier Courtin and Hugo Mercier / Oslandia)

### A.51.3 Enhancements

For detail of new functions and function improvements, please refer to Section [15.12.10](#).

Much faster raster ST\_Union, ST\_Clip and many more function additions operations

For geometry/geography better planner selectivity and a lot more functions.

#823, tiger geocoder: Make loader\_generate\_script download portion less greedy

#826, raster2pgsql no longer defaults to padding tiles. Flag -P can be used to pad tiles

#1363, ST\_AddBand(raster, ...) array version rewritten in C

#1364, ST\_Union(raster, ...) aggregate function rewritten in C

#1655, Additional default values for parameters of ST\_Slope

#1661, Add aggregate variant of ST\_SameAlignment

#1719, Add support for Point and GeometryCollection ST\_MakeValid inputs

- #1780, support ST\_GeoHash for geography
  - #1796, Big performance boost for distance calculations in geography
  - #1802, improved function interruptibility.
  - #1823, add parameter in ST\_AsGML to use id column for GML 3 output (become mandatory since GML 3.2.1)
  - #1856, tiger geocoder: reverse geocoder rating setting for prefer numbered highway name
  - #1938, Refactor basic ST\_AddBand to add multiple new bands in one call
  - #1978, wrong answer when calculating length of a closed circular arc (circle)
  - #1989, Preprocess input geometry to just intersection with raster to be clipped
  - #2021, Added multi-band support to ST\_Union(raster, ...) aggregate function
  - #2006, better support of ST\_Area(geography) over poles and dateline
  - #2065, ST\_Clip(raster, ...) now a C function
  - #2069, Added parameters to ST\_Tile(raster) to control padding of tiles
  - #2078, New variants of ST\_Slope, ST\_Aspect and ST\_HillShade to provide solution to handling tiles in a coverage
  - #2097, Added RANGE uniontype option for ST\_Union(raster)
  - #2105, Added ST\_Transform(raster) variant for aligning output to reference raster
  - #2119, Rasters passed to ST\_Resample(), ST\_Rescale(), ST\_Reskew(), and ST\_SnapToGrid() no longer require an SRID
  - #2141, More verbose output when constraints fail to be added to a raster column
  - #2143, Changed blocksize constraint of raster to allow multiple values
  - #2148, Addition of coverage\_tile constraint for raster
  - #2149, Addition of spatially\_unique constraint for raster
- TopologySummary output now includes unregistered layers and a count of missing TopoGeometry objects from their natural layer.
- ST\_HillShade(), ST\_Aspect() and ST\_Slope() have one new optional parameter to interpolate NODATA pixels before running the operation.
- Point variant of ST\_SetValue(raster) is now a wrapper around geomval variant of ST\_SetValues(rast).
- Proper support for raster band's isnodata flag in core API and loader.
- Additional default values for parameters of ST\_Aspect and ST\_HillShade
- #2178, ST\_Summary now advertises presence of known srid with an [S] flag
  - #2202, Make libjson-c optional (--without-json configure switch)
  - #2213, Add support libjson-c 0.10+
  - #2231, raster2pgsql supports user naming of filename column with -n
  - #2200, ST\_Union(raster, uniontype) unions all bands of all rasters
  - #2264, postgis\_restore.pl support for restoring into databases with postgis in a custom schema
  - #2244, emit warning when changing raster's georeference if raster has out-db bands
  - #2222, add parameter OutAsIn to flag whether ST\_AsBinary should return out-db bands as in-db bands
-

## A.51.4 Fixes

- #1839, handling of subdatasets in GeoTIFF in raster2pgsql.
  - #1840, fix logic of when to compute # of tiles in raster2pgsql.
  - #1870, align the docs and actual behavior of raster's ST\_Intersects
  - #1872, fix ST\_ApproxSummarystats to prevent division by zero
  - #1875, ST\_SummaryStats returns NULL for all parameters except count when count is zero
  - #1932, fix raster2pgsql of syntax for index tablespaces
  - #1936, ST\_GeomFromGML on CurvePolygon causes server crash
  - #1939, remove custom data types: summarystats, histogram, quantile, valuecount
  - #1951, remove crash on zero-length linestrings
  - #1957, ST\_Distance to a one-point LineString returns NULL
  - #1976, Geography point-in-ring code overhauled for more reliability
  - #1981, cleanup of unused variables causing warnings with gcc 4.6+
  - #1996, support POINT EMPTY in GeoJSON output
  - #2062, improve performance of distance calculations
  - #2057, Fixed linking issue for raster2pgsql to libpq
  - #2077, Fixed incorrect values returning from ST\_Hillshade()
  - #2019, ST\_FlipCoordinates does not update bbox
  - #2100, ST\_AsRaster may not return raster with specified pixel type
  - #2126, Better handling of empty rasters from ST\_ConvexHull()
  - #2165, ST\_NumPoints regression failure with CircularString
  - #2168, ST\_Distance is not always commutative
  - #2182, Fix issue with outdb rasters with no SRID and ST\_Resize
  - #2188, Fix function parameter value overflow that caused problems when copying data from a GDAL dataset
  - #2198, Fix incorrect dimensions used when generating bands of out-db rasters in ST\_Tile()
  - #2201, ST\_GeoHash wrong on boundaries
  - #2203, Changed how rasters with unknown SRID and default geotransform are handled when passing to GDAL Warp API
  - #2215, Fixed raster exclusion constraint for conflicting name of implicit index
  - #2251, Fix bad dimensions when rescaling rasters with default geotransform matrix
  - #2133, Fix performance regression in expression variant of ST\_MapAlgebra
  - #2257, GBOX variables not initialized when testing with empty geometries
  - #2271, Prevent parallel make of raster
  - #2282, Fix call to undefined function nd\_stats\_to\_grid() in debug mode
  - #2307, ST\_MakeValid outputs invalid geometries
  - #2309, Remove confusing INFO message when trying to get SRS info
  - #2336, FIPS 20 (KS) causes wildcard expansion to wget all files
  - #2348, Provide raster upgrade path for 2.0 to 2.1
  - #2351, st\_distance between geographies wrong
  - #2359, Fix handling of schema name when adding overview constraints
  - #2371, Support GEOS versions with more than 1 digit in micro
  - #2383, Remove unsafe use of \ from raster warning message
  - #2384, Incorrect variable datatypes for ST\_Neighborhood
-

## A.51.5 Known Issues

#2111, Raster bands can only reference the first 256 bands of out-db rasters

## A.52 Release 2.0.5

Release date: 2014/03/31

This is a bug fix release, addressing issues that have been filed since the 2.0.4 release. If you are using PostGIS 2.0+ a soft upgrade is required. For users of PostGIS 1.5 or below, a hard upgrade is required.

### A.52.1 Bug Fixes

#2494, avoid memcpy in GIST index

#2502, Fix postgis\_topology\_scripts\_installed() install schema

#2504, Fix segfault on bogus pgsq2shp call

#2528, Fix memory leak in ST\_Split / lwline\_split\_by\_line

#2532, Add missing raster/geometry commutator operators

#2533, Remove duplicated signatures

#2552, Fix NULL raster handling in ST\_AsPNG, ST\_AsTIFF and ST\_AsJPEG

#2555, Fix parsing issue of range arguments of ST\_Reclass

#2589, Remove use of unnecessary void pointers

#2607, Cannot open more than 1024 out-db files in process

#2610, Ensure face splitting algorithm uses the edge index

#2619, Empty ring array in GeoJSON polygon causes crash

#2638, Geography distance on M geometries sometimes wrong

### A.52.2 Important Changes

##2514, Change raster license from GPL v3+ to v2+, allowing distribution of PostGIS Extension as GPLv2.

## A.53 Release 2.0.4

Release date: 2013/09/06

This is a bug fix release, addressing issues that have been filed since the 2.0.3 release. If you are using PostGIS 2.0+ a soft upgrade is required. For users of PostGIS 1.5 or below, a hard upgrade is required.

### A.53.1 Bug Fixes

#2110, Equality operator between EMPTY and point on origin

Allow adding points at precision distance with TopoGeo\_addPoint

#1968, Fix missing edge from toTopoGeom return

#2165, ST\_NumPoints regression failure with CircularString

- #2168, ST\_Distance is not always commutative
- #2186, gui progress bar updates too frequent
- #2201, ST\_GeoHash wrong on boundaries
- #2257, GBOX variables not initialized when testing with empty geometries
- #2271, Prevent parallel make of raster
- #2267, Server crash from analyze table
- #2277, potential segfault removed
- #2307, ST\_MakeValid outputs invalid geometries
- #2351, st\_distance between geographies wrong
- #2359, Incorrect handling of schema for overview constraints
- #2371, Support GEOS versions with more than 1 digit in micro
- #2372, Cannot parse space-padded KML coordinates
- Fix build with systemwide liblwgeom installed
- #2383, Fix unsafe use of \ in warning message
- #2410, Fix segmentize of collinear curve
- #2412, ST\_LineToCurve support for lines with less than 4 vertices
- #2415, ST\_Multi support for COMPOUNDCURVE and CURVEPOLYGON
- #2420, ST\_LineToCurve: require at least 8 edges to define a full circle
- #2423, ST\_LineToCurve: require all arc edges to form the same angle
- #2424, ST\_CurveToLine: add support for COMPOUNDCURVE in MULTICURVE
- #2427, Make sure to retain first point of curves on ST\_CurveToLine

### A.53.2 Enhancements

- #2269, Avoid uselessly detoasting full geometries on ANALYZE

### A.53.3 Known Issues

- #2111, Raster bands can only reference the first 256 bands of out-db rasters

## A.54 Release 2.0.3

Release date: 2013/03/01

This is a bug fix release, addressing issues that have been filed since the 2.0.2 release. If you are using PostGIS 2.0+ a soft upgrade is required. For users of PostGIS 1.5 or below, a hard upgrade is required.

### A.54.1 Bug Fixes

#2126, Better handling of empty rasters from ST\_ConvexHull()

#2134, Make sure to process SRS before passing it off to GDAL functions

Fix various memory leaks in liblwgeom

#2173, Fix robustness issue in splitting a line with own vertex also affecting topology building (#2172)

#2174, Fix usage of wrong function lwpoly\_free()

#2176, Fix robustness issue with ST\_ChangeEdgeGeom

#2184, Properly copy topologies with Z value

postgis\_restore.pl support for mixed case geometry column name in dumps

#2188, Fix function parameter value overflow that caused problems when copying data from a GDAL dataset

#2216, More memory errors in MultiPolygon GeoJSON parsing (with holes)

Fix Memory leak in GeoJSON parser

### A.54.2 Enhancements

#2141, More verbose output when constraints fail to be added to a raster column

Speedup ST\_ChangeEdgeGeom

## A.55 Release 2.0.2

Release date: 2012/12/03

This is a bug fix release, addressing issues that have been filed since the 2.0.1 release.

### A.55.1 Bug Fixes

#1287, Drop of "gist\_geometry\_ops" broke a few clients package of legacy\_gist.sql for these cases

#1391, Errors during upgrade from 1.5

#1828, Poor selectivity estimate on ST\_DWithin

#1838, error importing tiger/line data

#1869, ST\_AsBinary is not unique added to legacy\_minor/legacy.sql scripts

#1885, Missing field from tabblock table in tiger2010 census\_loader.sql

#1891, Use LDFFLAGS environment when building liblwgeom

#1900, Fix pgsq2shp for big-endian systems

#1932, Fix raster2pgsql for invalid syntax for setting index tablespace

#1936, ST\_GeomFromGML on CurvePolygon causes server crash

#1955, ST\_ModEdgeHeal and ST\_NewEdgeHeal for doubly connected edges

#1957, ST\_Distance to a one-point LineString returns NULL

#1976, Geography point-in-ring code overhauled for more reliability

#1978, wrong answer calculating length of closed circular arc (circle)

#1981, Remove unused but set variables as found with gcc 4.6+

- #1987, Restore 1.5.x behaviour of ST\_Simplify
  - #1989, Preprocess input geometry to just intersection with raster to be clipped
  - #1991, geocode really slow on PostgreSQL 9.2
  - #1996, support POINT EMPTY in GeoJSON output
  - #1998, Fix ST\_{Mod,New}EdgeHeal joining edges sharing both endpoints
  - #2001, ST\_CurveToLine has no effect if the geometry doesn't actually contain an arc
  - #2015, ST\_IsEmpty('POLYGON(EMPTY)') returns False
  - #2019, ST\_FlipCoordinates does not update bbox
  - #2025, Fix side location conflict at TopoGeo\_AddLineString
  - #2026, improve performance of distance calculations
  - #2033, Fix adding a splitting point into a 2.5d topology
  - #2051, Fix excess of precision in ST\_AsGeoJSON output
  - #2052, Fix buffer overflow in lwgeom\_to\_geojson
  - #2056, Fixed lack of SRID check of raster and geometry in ST\_SetValue()
  - #2057, Fixed linking issue for raster2psql to libpq
  - #2060, Fix "dimension" check violation by GetTopoGeomElementArray
  - #2072, Removed outdated checks preventing ST\_Intersects(raster) from working on out-db bands
  - #2077, Fixed incorrect answers from ST\_Hillshade(raster)
  - #2092, Namespace issue with ST\_GeomFromKML, ST\_GeomFromGML for libxml 2.8+
  - #2099, Fix double free on exception in ST\_OffsetCurve
  - #2100, ST\_AsRaster() may not return raster with specified pixel type
  - #2108, Ensure ST\_Line\_Interpolate\_Point always returns POINT
  - #2109, Ensure ST\_Centroid always returns POINT
  - #2117, Ensure ST\_PointOnSurface always returns POINT
  - #2129, Fix SRID in ST\_Homogenize output with collection input
  - #2130, Fix memory error in MultiPolygon GeoJson parsing
- Update URL of Maven jar

### A.55.2 Enhancements

- #1581, ST\_Clip(raster, ...) no longer imposes NODATA on a band if the corresponding band from the source raster did not have NODATA
- #1928, Accept array properties in GML input multi-geom input (Kashif Rasul and Shoaib Burq / SpacialDB)
- #2082, Add indices on start\_node and end\_node of topology edge tables
- #2087, Speedup topology.GetRingEdges using a recursive CTE

## A.56 Release 2.0.1

Release date: 2012/06/22

This is a bug fix release, addressing issues that have been filed since the 2.0.0 release.

---

## A.56.1 Bug Fixes

- #1264, fix `st_dwithin(geog, geog, 0)`.
- #1468 shp2pgsql-gui table column schema get shifted
- #1694, fix building with clang. (vince)
- #1708, improve restore of pre-PostGIS 2.0 backups.
- #1714, more robust handling of high topology tolerance.
- #1755, `ST_GeographyFromText` support for higher dimensions.
- #1759, loading transformed shapefiles in raster enabled db.
- #1761, handling of subdatasets in NetCDF, HDF4 and HDF5 in raster2pgsql.
- #1763, `topology.toTopoGeom` use with custom `search_path`.
- #1766, don't let `ST_RemEdge*` destroy peripheral TopoGeometry objects.
- #1774, Clearer error on setting an edge geometry to an invalid one.
- #1775, `ST_ChangeEdgeGeom` collision detection with 2-vertex target.
- #1776, fix `ST_SymDifference(empty, geom)` to return geom.
- #1779, install SQL comment files.
- #1782, fix spatial reference string handling in raster.
- #1789, fix false edge-node crossing report in `ValidateTopology`.
- #1790, fix `toTopoGeom` handling of duplicated primitives.
- #1791, fix `ST_Azimuth` with very close but distinct points.
- #1797, fix `(ValidateTopology(xxx)).*` syntax calls.
- #1805, put back the 900913 SRID entry.
- #1813, Only show readable relations in metadata tables.
- #1819, fix floating point issues with `ST_World2RasterCoord` and `ST_Raster2WorldCoord` variants.
- #1820 compilation on 9.2beta1.
- #1822, topology load on PostgreSQL 9.2beta1.
- #1825, fix prepared geometry cache lookup
- #1829, fix uninitialized read in GeoJSON parser
- #1834, revise postgis extension to only backup user specified `spatial_ref_sys`
- #1839, handling of subdatasets in GeoTIFF in raster2pgsql.
- #1840, fix logic of when to compute # of tiles in raster2pgsql.
- #1851, fix `spatial_ref_system` parameters for EPSG:3844
- #1857, fix failure to detect endpoint mismatch in `ST_AddEdge*Face*`
- #1865, data loss in `postgis_restore.pl` when data rows have leading dashes.
- #1867, catch invalid topology name passed to `topogeo_add*`
- #1872, fix `ST_ApproxSummarystats` to prevent division by zero
- #1873, fix `ptarray_locate_point` to return interpolated Z/M values for on-the-line case
- #1875, `ST_SummaryStats` returns NULL for all parameters except count when count is zero
- #1881, shp2pgsql-gui -- editing a field sometimes triggers removing row
- #1883, Geocoder install fails trying to run `create_census_base_tables()` (Brian Panulla)

## A.56.2 Enhancements

More detailed exception message from topology editing functions.

#1786, improved build dependencies

#1806, speedup of ST\_BuildArea, ST\_MakeValid and ST\_GetFaceGeometry.

#1812, Add lwgeom\_normalize in LIBLWGEOM for more stable testing.

## A.57 Release 2.0.0

Release date: 2012/04/03

This is a major release. A hard upgrade is required. Yes this means a full dump reload and some special preparations if you are using obsolete functions. Refer to Section 3.4.2 for details on upgrading. Refer to Section 15.12.12 for more details and changed/new functions.

### A.57.1 Testers - Our unsung heroes

We are most indebted to the numerous members in the PostGIS community who were brave enough to test out the new features in this release. No major release can be successful without these folk.

Below are those who have been most valiant, provided very detailed and thorough bug reports, and detailed analysis.

Andrea Peri - Lots of testing on topology, checking for correctness

Andreas Forø Tollefsen - raster testing

Chris English - topology stress testing loader functions

Salvatore Larosa - topology robustness testing

Brian Hamlin - Benchmarking (also experimental experimental branches before they are folded into core) , general testing of various

Mike Pease - Tiger geocoder testing - very detailed reports of issues

Tom van Tilburg - raster testing

### A.57.2 Important / Breaking Changes

#722, #302, Most deprecated functions removed (over 250 functions) (Regina Obe, Paul Ramsey)

Unknown SRID changed from -1 to 0. (Paul Ramsey)

-- (most deprecated in 1.2) removed non-ST variants buffer, length, intersects (and internal functions renamed) etc.

-- If you have been using deprecated functions CHANGE your apps or suffer the consequences. If you don't see a function documented -- it ain't supported or it is an internal function. Some constraints in older tables were built with deprecated functions. If you restore you may need to rebuild table constraints with populate\_geometry\_columns(). If you have applications or tools that rely on deprecated functions, please refer to [?qandaentry] for more details.

#944 geometry\_columns is now a view instead of a table (Paul Ramsey, Regina Obe) for tables created the old way reads (srid, type, dims) constraints for geometry columns created with type modifiers reads from column definition

#1081, #1082, #1084, #1088 - Management functions support typmod geometry column creation functions now default to typmod creation (Regina Obe)

#1083 probe\_geometry\_columns(), rename\_geometry\_table\_constraints(), fix\_geometry\_columns(); removed - now obsolete with geometry\_column view (Regina Obe)

#817 Renaming old 3D functions to the convention ST\_3D (Nicklas Avén)

#548 (sorta), ST\_NumGeometries, ST\_GeometryN now returns 1 (or the geometry) instead of null for single geometries (Sandro Santilli, Maxime van Noppen)

### A.57.3 New Features

**KNN Gist index based centroid (<->) and box (<#>) distance operators (Paul Ramsey / funded by Vizzuality)**

Support for TIN and PolyHedralSurface and enhancement of many functions to support 3D (Olivier Courtin / Oslandia)

**Raster support integrated and documented** (Pierre Racine, Jorge Arévalo, Mateusz Loskot, Sandro Santilli, David Zwarg, Regina Obe, Bborie Park) (Company developer and funding: University Laval, Deimos Space, CadCorp, Michigan Tech Research Institute, Azavea, Paragon Corporation, UC Davis Center for Vectorborne Diseases)

Making spatial indexes 3D aware - in progress (Paul Ramsey, Mark Cave-Ayland)

Topology support improved (more functions), documented, testing (Sandro Santilli / Faunalia for RT-SIGTA), Andrea Peri, Regina Obe, Jose Carlos Martinez Llari

3D relationship and measurement support functions (Nicklas Avén)

ST\_3DDistance, ST\_3DClosestPoint, ST\_3DIntersects, ST\_3DShortestLine and more...

N-Dimensional spatial indexes (Paul Ramsey / OpenGeo)

ST\_Split (Sandro Santilli / Faunalia for RT-SIGTA)

ST\_IsValidDetail (Sandro Santilli / Faunalia for RT-SIGTA)

ST\_MakeValid (Sandro Santilli / Faunalia for RT-SIGTA)

ST\_RemoveRepeatedPoints (Sandro Santilli / Faunalia for RT-SIGTA)

ST\_GeometryN and ST\_NumGeometries support for non-collections (Sandro Santilli)

ST\_IsCollection (Sandro Santilli, Maxime van Noppen)

ST\_SharedPaths (Sandro Santilli / Faunalia for RT-SIGTA)

ST\_Snap (Sandro Santilli)

ST\_RelateMatch (Sandro Santilli / Faunalia for RT-SIGTA)

ST\_ConcaveHull (Regina Obe and Leo Hsu / Paragon Corporation)

ST\_UnaryUnion (Sandro Santilli / Faunalia for RT-SIGTA)

ST\_AsX3D (Regina Obe / Arrival 3D funding)

ST\_OffsetCurve (Sandro Santilli, Rafal Magda)

**ST\_GeomFromGeoJSON (Kashif Rasul, Paul Ramsey / Vizzuality funding)**

### A.57.4 Enhancements

Made shape file loader tolerant of truncated multibyte values found in some free worldwide shapefiles (Sandro Santilli)

Lots of bug fixes and enhancements to shp2pgsql Beefing up regression tests for loaders Reproject support for both geometry and geography during import (Jeff Adams / Azavea, Mark Cave-Ayland)

pgsql2shp conversion from predefined list (Loic Dachary / Mark Cave-Ayland)

Shp-pgsql GUI loader - support loading multiple files at a time. (Mark Leslie)

Extras - upgraded tiger\_geocoder from using old TIGER format to use new TIGER shp and file structure format (Stephen Frost)

Extras - revised tiger\_geocoder to work with TIGER census 2010 data, addition of reverse geocoder function, various bug fixes, accuracy enhancements, limit max result return, speed improvements, loading routines. (Regina Obe, Leo Hsu / Paragon Corporation / funding provided by Hunter Systems Group)

Overall Documentation proofreading and corrections. (Kasif Rasul)

Cleanup PostGIS JDBC classes, revise to use Maven build. (Maria Arias de Reyna, Sandro Santilli)

## A.57.5 Bug Fixes

#1335 ST\_AddPoint returns incorrect result on Linux (Even Rouault)

## A.57.6 Release specific credits

We thank [U.S Department of State Human Information Unit \(HIU\)](#) and [Vizzuality](#) for general monetary support to get PostGIS 2.0 out the door.

## A.58 Release 1.5.4

Release date: 2012/05/07

This is a bug fix release, addressing issues that have been filed since the 1.5.3 release.

### A.58.1 Bug Fixes

#547, ST\_Contains memory problems (Sandro Santilli)

#621, Problem finding intersections with geography (Paul Ramsey)

#627, PostGIS/PostgreSQL process die on invalid geometry (Paul Ramsey)

#810, Increase accuracy of area calculation (Paul Ramsey)

#852, improve spatial predicates robustness (Sandro Santilli, Nicklas Avén)

#877, ST\_Estimated\_Extent returns NULL on empty tables (Sandro Santilli)

#1028, ST\_AsSVG kills whole postgres server when fails (Paul Ramsey)

#1056, Fix boxes of arcs and circle stroking code (Paul Ramsey)

#1121, populate\_geometry\_columns using deprecated functions (Regin Obe, Paul Ramsey)

#1135, improve testsuite predictability (Andreas 'ads' Scherbaum)

#1146, images generator crashes (bronaugh)

#1170, North Pole intersection fails (Paul Ramsey)

#1179, ST\_AsText crash with bad value (kjurka)

#1184, honour DESTDIR in documentation Makefile (Bryce L Nordgren)

#1227, server crash on invalid GML

#1252, SRID appearing in WKT (Paul Ramsey)

#1264, st\_dwithin(g, g, 0) doesn't work (Paul Ramsey)

#1344, allow exporting tables with invalid geometries (Sandro Santilli)

#1389, wrong proj4text for SRID 31300 and 31370 (Paul Ramsey)

#1406, shp2pgsql crashes when loading into geography (Sandro Santilli)

#1595, fixed SRID redundancy in ST\_Line\_SubString (Sandro Santilli)

#1596, check SRID in UpdateGeometrySRID (Mike Toews, Sandro Santilli)

#1602, fix ST\_Polygonize to retain Z (Sandro Santilli)

#1697, fix crash with EMPTY entries in GiST index (Paul Ramsey)

#1772, fix ST\_Line\_Locate\_Point with collapsed input (Sandro Santilli)

#1799, Protect ST\_Segmentize from max\_length=0 (Sandro Santilli)

Alter parameter order in 900913 (Paul Ramsey)

Support builds with "gmake" (Greg Troxel)

## A.59 Release 1.5.3

Release date: 2011/06/25

This is a bug fix release, addressing issues that have been filed since the 1.5.2 release. If you are running PostGIS 1.3+, a soft upgrade is sufficient otherwise a hard upgrade is recommended.

### A.59.1 Bug Fixes

- #1056, produce correct bboxes for arc geometries, fixes index errors (Paul Ramsey)
- #1007, ST\_IsValid crash fix requires GEOS 3.3.0+ or 3.2.3+ (Sandro Santilli, reported by Birgit Laggner)
- #940, support for PostgreSQL 9.1 beta 1 (Regina Obe, Paul Ramsey, patch submitted by stl)
- #845, ST\_Intersects precision error (Sandro Santilli, Nicklas Avén) Reported by cdestigter
- #884, Unstable results with ST\_Within, ST\_Intersects (Chris Hodgson)
- #779, shp2pgsql -S option seems to fail on points (Jeff Adams)
- #666, ST\_DumpPoints is not null safe (Regina Obe)
- #631, Update NZ projections for grid transformation support (jpalmer)
- #630, Peculiar Null treatment in arrays in ST\_Collect (Chris Hodgson) Reported by David Bitner
- #624, Memory leak in ST\_GeogFromText (ryang, Paul Ramsey)
- #609, Bad source code in manual section 5.2 Java Clients (simoc, Regina Obe)
- #604, shp2pgsql usage touchups (Mike Toews, Paul Ramsey)
- #573 ST\_Union fails on a group of linestrings Not a PostGIS bug, fixed in GEOS 3.3.0
- #457 ST\_CollectionExtract returns non-requested type (Nicklas Avén, Paul Ramsey)
- #441 ST\_AsGeoJson Bbox on GeometryCollection error (Olivier Courtin)
- #411 Ability to backup invalid geometries (Sando Santilli) Reported by Regione Toscana
- #409 ST\_AsSVG - degraded (Olivier Courtin) Reported by Sdikiy
- #373 Documentation syntax error in hard upgrade (Paul Ramsey) Reported by psvensso

## A.60 Release 1.5.2

Release date: 2010/09/27

This is a bug fix release, addressing issues that have been filed since the 1.5.1 release. If you are running PostGIS 1.3+, a soft upgrade is sufficient otherwise a hard upgrade is recommended.

### A.60.1 Bug Fixes

Loader: fix handling of empty (0-verticed) geometries in shapefiles. (Sandro Santilli)

#536, Geography ST\_Intersects, ST\_Covers, ST\_CoveredBy and Geometry ST\_Equals not using spatial index (Regina Obe, Nicklas Aven)

#573, Improvement to ST\_Contains geography (Paul Ramsey)

Loader: Add support for command-q shutdown in Mac GTK build (Paul Ramsey)

#393, Loader: Add temporary patch for large DBF files (Maxime Guillaud, Paul Ramsey)

- #507, Fix wrong OGC URN in GeoJSON and GML output (Olivier Courtin)
- spatial\_ref\_sys.sql Add datum conversion for projection SRID 3021 (Paul Ramsey)
- Geography - remove crash for case when all geographies are out of the estimate (Paul Ramsey)
- #469, Fix for array\_aggregation error (Greg Stark, Paul Ramsey)
- #532, Temporary geography tables showing up in other user sessions (Paul Ramsey)
- #562, ST\_Dwithin errors for large geographies (Paul Ramsey)
- #513, shape loading GUI tries to make spatial index when loading DBF only mode (Paul Ramsey)
- #527, shape loading GUI should always append log messages (Mark Cave-Ayland)
- #504, shp2pgsql should rename xmin/xmax fields (Sandro Santilli)
- #458, postgis\_comments being installed in contrib instead of version folder (Mark Cave-Ayland)
- #474, Analyzing a table with geography column crashes server (Paul Ramsey)
- #581, LWGEOM-expand produces inconsistent results (Mark Cave-Ayland)
- #513, Add dbf filter to shp2pgsql-gui and allow uploading dbf only (Paul Ramsey)
- Fix further build issues against PostgreSQL 9.0 (Mark Cave-Ayland)
- #572, Password whitespace for Shape File (Mark Cave-Ayland)
- #603, shp2pgsql: "-w" produces invalid WKT for MULTI\* objects. (Mark Cave-Ayland)

## A.61 Release 1.5.1

Release date: 2010/03/11

This is a bug fix release, addressing issues that have been filed since the 1.4.1 release. If you are running PostGIS 1.3+, a soft upgrade is sufficient otherwise a hard upgrade is recommended.

### A.61.1 Bug Fixes

- #410, update embedded bbox when applying ST\_SetPoint, ST\_AddPoint ST\_RemovePoint to a linestring (Paul Ramsey)
  - #411, allow dumping tables with invalid geometries (Sandro Santilli, for Regione Toscana-SIGTA)
  - #414, include geography\_columns view when running upgrade scripts (Paul Ramsey)
  - #419, allow support for multilinestring in ST\_Line\_Substring (Paul Ramsey, for Lidwala Consulting Engineers)
  - #421, fix computed string length in ST\_AsGML() (Olivier Courtin)
  - #441, fix GML generation with heterogeneous collections (Olivier Courtin)
  - #443, incorrect coordinate reversal in GML 3 generation (Olivier Courtin)
  - #450, #451, wrong area calculation for geography features that cross the date line (Paul Ramsey)
- Ensure support for upcoming 9.0 PostgreSQL release (Paul Ramsey)

## A.62 Release 1.5.0

Release date: 2010/02/04

This release provides support for geographic coordinates (lat/lon) via a new GEOGRAPHY type. Also performance enhancements, new input format support (GML,KML) and general upkeep.

### A.62.1 API Stability

The public API of PostGIS will not change during minor (0.0.X) releases.

The definition of the `=~` operator has changed from an exact geometric equality check to a bounding box equality check.

### A.62.2 Compatibility

GEOS, Proj4, and LibXML2 are now mandatory dependencies

The library versions below are the minimum requirements for PostGIS 1.5

PostgreSQL 8.3 and higher on all platforms

GEOS 3.1 and higher only (GEOS 3.2+ to take advantage of all features)

LibXML2 2.5+ related to new `ST_GeomFromGML/KML` functionality

Proj4 4.5 and higher only

### A.62.3 New Features

Section [15.12.14](#)

Added Hausdorff distance calculations ([#209](#)) (Vincent Picavet)

Added `parameters` argument to `ST_Buffer` operation to support one-sided buffering and other buffering styles (Sandro Santilli)

Addition of other Distance related visualization and analysis functions (Nicklas Aven)

- `ST_ClosestPoint`
- `ST_DFullyWithin`
- `ST_LongestLine`
- `ST_MaxDistance`
- `ST_ShortestLine`

`ST_DumpPoints` (Maxime van Noppen)

KML, GML input via `ST_GeomFromGML` and `ST_GeomFromKML` (Olivier Courtin)

Extract homogeneous collection with `ST_CollectionExtract` (Paul Ramsey)

Add measure values to an existing linestring with `ST_AddMeasure` (Paul Ramsey)

History table implementation in `utils` (George Silva)

Geography type and supporting functions

- Spherical algorithms (Dave Skea)
  - Object/index implementation (Paul Ramsey)
  - Selectivity implementation (Mark Cave-Ayland)
  - Serializations to KML, GML and JSON (Olivier Courtin)
  - `ST_Area`, `ST_Distance`, `ST_DWithin`, `ST_GeogFromText`, `ST_GeogFromWKB`, `ST_Intersects`, `ST_Covers`, `ST_Buffer` (Paul Ramsey)
-

## A.62.4 Enhancements

Performance improvements to ST\_Distance (Nicklas Aven)  
Documentation updates and improvements (Regina Obe, Kevin Neufeld)  
Testing and quality control (Regina Obe)  
PostGIS 1.5 support PostgreSQL 8.5 trunk (Guillaume Lelarge)  
Win32 support and improvement of core shp2pgsql-gui (Mark Cave-Ayland)  
In place 'make check' support (Paul Ramsey)

## A.62.5 Bug fixes

<http://trac.osgeo.org/postgis/query?status=closed&milestone=PostGIS+1.5.0&order=priority>

## A.63 Release 1.4.0

Release date: 2009/07/24

This release provides performance enhancements, improved internal structures and testing, new features, and upgraded documentation. If you are running PostGIS 1.1+, a soft upgrade is sufficient otherwise a hard upgrade is recommended.

### A.63.1 API Stability

As of the 1.4 release series, the public API of PostGIS will not change during minor releases.

### A.63.2 Compatibility

The versions below are the \*minimum\* requirements for PostGIS 1.4

PostgreSQL 8.2 and higher on all platforms

GEOS 3.0 and higher only

PROJ4 4.5 and higher only

### A.63.3 New Features

ST\_Union() uses high-speed cascaded union when compiled against GEOS 3.1+ (Paul Ramsey)

ST\_ContainsProperly() requires GEOS 3.1+

ST\_Intersects(), ST\_Contains(), ST\_Within() use high-speed cached prepared geometry against GEOS 3.1+ (Paul Ramsey / funded by Zonar Systems)

Vastly improved documentation and reference manual (Regina Obe & Kevin Neufeld)

Figures and diagram examples in the reference manual (Kevin Neufeld)

ST\_IsValidReason() returns readable explanations for validity failures (Paul Ramsey)

ST\_GeoHash() returns a geohash.org signature for geometries (Paul Ramsey)

GTK+ multi-platform GUI for shape file loading (Paul Ramsey)

ST\_LineCrossingDirection() returns crossing directions (Paul Ramsey)

ST\_LocateBetweenElevations() returns sub-string based on Z-ordinate. (Paul Ramsey)

Geometry parser returns explicit error message about location of syntax errors (Mark Cave-Ayland)

ST\_AsGeoJSON() return JSON formatted geometry (Olivier Courtin)

Populate\_Geometry\_Columns() -- automatically add records to geometry\_columns for TABLES and VIEWS (Kevin Neufeld)

ST\_MinimumBoundingCircle() -- returns the smallest circle polygon that can encompass a geometry (Bruce Rindahl)

#### A.63.4 Enhancements

Core geometry system moved into independent library, liblwgeom. (Mark Cave-Ayland)

New build system uses PostgreSQL "pgxs" build bootstrapper. (Mark Cave-Ayland)

Debugging framework formalized and simplified. (Mark Cave-Ayland)

All build-time #defines generated at configure time and placed in headers for easier cross-platform support (Mark Cave-Ayland)

Logging framework formalized and simplified (Mark Cave-Ayland)

Expanded and more stable support for CIRCULARSTRING, COMPOUNDCURVE and CURVEPOLYGON, better parsing, wider support in functions (Mark Leslie & Mark Cave-Ayland)

Improved support for OpenSolaris builds (Paul Ramsey)

Improved support for MSVC builds (Mateusz Loskot)

Updated KML support (Olivier Courtin)

Unit testing framework for liblwgeom (Paul Ramsey)

New testing framework to comprehensively exercise every PostGIS function (Regine Obe)

Performance improvements to all geometry aggregate functions (Paul Ramsey)

Support for the upcoming PostgreSQL 8.4 (Mark Cave-Ayland, Talha Bin Rizwan)

Shp2pgsql and pgsq2shp re-worked to depend on the common parsing/unparsing code in liblwgeom (Mark Cave-Ayland)

Use of PDF DbLatex to build PDF docs and preliminary instructions for build (Jean David Techer)

Automated User documentation build (PDF and HTML) and Developer Doxygen Documentation (Kevin Neufeld)

Automated build of document images using ImageMagick from WKT geometry text files (Kevin Neufeld)

More attractive CSS for HTML documentation (Dane Springmeyer)

#### A.63.5 Bug fixes

<http://trac.osgeo.org/postgis/query?status=closed&milestone=PostGIS+1.4.0&order=priority>

### A.64 Release 1.3.6

Release date: 2009/05/04

If you are running PostGIS 1.1+, a soft upgrade is sufficient otherwise a hard upgrade is recommended. This release adds support for PostgreSQL 8.4, exporting prj files from the database with shape data, some crash fixes for shp2pgsql, and several small bug fixes in the handling of "curve" types, logical error importing dbf only files, improved error handling of AddGeometryColumns.

### A.65 Release 1.3.5

Release date: 2008/12/15

If you are running PostGIS 1.1+, a soft upgrade is sufficient otherwise a hard upgrade is recommended. This release is a bug fix release to address a failure in ST\_Force\_Collection and related functions that critically affects using MapServer with LINE layers.

## A.66 Release 1.3.4

Release date: 2008/11/24

This release adds support for GeoJSON output, building with PostgreSQL 8.4, improves documentation quality and output aesthetics, adds function-level SQL documentation, and improves performance for some spatial predicates (point-in-polygon tests).

Bug fixes include removal of crashers in handling circular strings for many functions, some memory leaks removed, a linear referencing failure for measures on vertices, and more. See the NEWS file for details.

## A.67 Release 1.3.3

Release date: 2008/04/12

This release fixes bugs shp2pgsql, adds enhancements to SVG and KML support, adds a ST\_SimplifyPreserveTopology function, makes the build more sensitive to GEOS versions, and fixes a handful of severe but rare failure cases.

## A.68 Release 1.3.2

Release date: 2007/12/01

This release fixes bugs in ST\_EndPoint() and ST\_Envelope, improves support for JDBC building and OS/X, and adds better support for GML output with ST\_AsGML(), including GML3 output.

## A.69 Release 1.3.1

Release date: 2007/08/13

This release fixes some oversights in the previous release around version numbering, documentation, and tagging.

## A.70 Release 1.3.0

Release date: 2007/08/09

This release provides performance enhancements to the relational functions, adds new relational functions and begins the migration of our function names to the SQL-MM convention, using the spatial type (SP) prefix.

### A.70.1 Added Functionality

JDBC: Added Hibernate Dialect (thanks to Norman Barker)

Added ST\_Covers and ST\_CoveredBy relational functions. Description and justification of these functions can be found at <http://lin-ear-th-inking.blogspot.com/2007/06/subtleties-of-ogc-covers-spatial.html>

Added ST\_DWithin relational function.

### A.70.2 Performance Enhancements

Added cached and indexed point-in-polygon short-circuits for the functions ST\_Contains, ST\_Intersects, ST\_Within and ST\_Disjoint

Added inline index support for relational functions (except ST\_Disjoint)

---

### A.70.3 Other Changes

Extended curved geometry support into the geometry accessor and some processing functions

Began migration of functions to the SQL-MM naming convention; using a spatial type (ST) prefix.

Added initial support for PostgreSQL 8.3

## A.71 Release 1.2.1

Release date: 2007/01/11

This release provides bug fixes in PostgreSQL 8.2 support and some small performance enhancements.

### A.71.1 Changes

Fixed point-in-polygon shortcut bug in `Within()`.

Fixed PostgreSQL 8.2 NULL handling for indexes.

Updated RPM spec files.

Added short-circuit for `Transform()` in no-op case.

JDBC: Fixed JTS handling for multi-dimensional geometries (thanks to Thomas Marti for hint and partial patch). Additionally, now JavaDoc is compiled and packaged. Fixed classpath problems with GCJ. Fixed pgjdbc 8.2 compatibility, losing support for jdk 1.3 and older.

## A.72 Release 1.2.0

Release date: 2006/12/08

This release provides type definitions along with serialization/deserialization capabilities for SQL-MM defined curved geometries, as well as performance enhancements.

### A.72.1 Changes

Added curved geometry type support for serialization/deserialization

Added point-in-polygon shortcircuit to the `Contains` and `Within` functions to improve performance for these cases.

## A.73 Release 1.1.6

Release date: 2006/11/02

This is a bugfix release, in particular fixing a critical error with GEOS interface in 64bit systems. Includes an updated of the SRS parameters and an improvement in reprojections (take Z in consideration). Upgrade is *encouraged*.

### A.73.1 **Upgrade**

If you are upgrading from release 1.0.3 or later follow the **soft upgrade** procedure.

If you are upgrading from a release *between 1.0.0RC6 and 1.0.2* (inclusive) and really want a live upgrade read the **upgrade section** of the 1.0.3 release notes chapter.

Upgrade from any release prior to 1.0.0RC6 requires an **hard upgrade**.

---

### A.73.2 Bug fixes

fixed CAPI change that broke 64-bit platforms

loader/dumper: fixed regression tests and usage output

Fixed setSRID() bug in JDBC, thanks to Thomas Marti

### A.73.3 Other changes

use Z ordinate in reprojections

spatial\_ref\_sys.sql updated to EPSG 6.11.1

Simplified Version.config infrastructure to use a single pack of version variables for everything.

Include the Version.config in loader/dumper USAGE messages

Replace hand-made, fragile JDBC version parser with Properties

## A.74 Release 1.1.5

Release date: 2006/10/13

This is an bugfix release, including a critical segfault on win32. Upgrade is *encouraged*.

### A.74.1 Upgrade procedure;

If you are upgrading from release 1.0.3 or later follow the [soft upgrade](#) procedure.

If you are upgrading from a release *between 1.0.0RC6 and 1.0.2* (inclusive) and really want a live upgrade read the [upgrade section](#) of the 1.0.3 release notes chapter.

Upgrade from any release prior to 1.0.0RC6 requires an [hard upgrade](#).

### A.74.2 Bug fixes

Fixed MingW link error that was causing pgsq2shp to segfault on Win32 when compiled for PostgreSQL 8.2

fixed nullpointer Exception in Geometry.equals() method in Java

Added EJB3Spatial.odt to fulfill the GPL requirement of distributing the "preferred form of modification"

Removed obsolete synchronization from JDBC Jts code.

Updated heavily outdated README files for shp2pgsql/pgsq2shp by merging them with the manpages.

Fixed version tag in jdbc code that still said "1.1.3" in the "1.1.4" release.

### A.74.3 New Features

Added -S option for non-multi geometries to shp2pgsql

## A.75 Release 1.1.4

Release date: 2006/09/27

This is an bugfix release including some improvements in the Java interface. Upgrade is *encouraged*.

---

### A.75.1

If you are upgrading from release 1.0.3 or later follow the [soft upgrade](#) procedure.

If you are upgrading from a release *between 1.0.0RC6 and 1.0.2* (inclusive) and really want a live upgrade read the [upgrade section](#) of the 1.0.3 release notes chapter.

Upgrade from any release prior to 1.0.0RC6 requires an [hard upgrade](#).

### A.75.2 Bug fixes

Fixed support for PostgreSQL 8.2

Fixed bug in collect() function discarding SRID of input

Added SRID match check in MakeBox2d and MakeBox3d

Fixed regress tests to pass with GEOS-3.0.0

Improved pgsq2shp run concurrency.

### A.75.3 Java changes

reworked JTS support to reflect new upstream JTS developers' attitude to SRID handling. Simplifies code and drops build depend on GNU trove.

Added EJB2 support generously donated by the "Geodetix s.r.l. Company"

Added EJB3 tutorial / examples donated by Norman Barker <nbarker@ittvis.com>

Reorganized java directory layout a little.

## A.76 Release 1.1.3

Release date: 2006/06/30

This is an bugfix release including also some new functionalities (most notably long transaction support) and portability enhancements. Upgrade is *encouraged*.

### A.76.1

If you are upgrading from release 1.0.3 or later follow the [soft upgrade](#) procedure.

If you are upgrading from a release *between 1.0.0RC6 and 1.0.2* (inclusive) and really want a live upgrade read the [upgrade section](#) of the 1.0.3 release notes chapter.

Upgrade from any release prior to 1.0.0RC6 requires an [hard upgrade](#).

### A.76.2 Bug fixes / correctness

BUGFIX in distance(poly,poly) giving wrong results.

BUGFIX in pgsq2shp successful return code.

BUGFIX in shp2pgsql handling of MultiLine WKT.

BUGFIX in affine() failing to update bounding box.

WKT parser: forbidden construction of multigeometries with EMPTY elements (still supported for GEOMETRYCOLLECTION).

---

### A.76.3 New functionalities

NEW Long Transactions support.

NEW DumpRings() function.

NEW AsHEXEWKB(geom, XDRINDR) function.

### A.76.4 JDBC changes

Improved regression tests: MultiPoint and scientific ordinates

Fixed some minor bugs in jdbc code

Added proper accessor functions for all fields in preparation of making those fields private later

### A.76.5 Other changes

NEW regress test support for loader/dumper.

Added --with-proj-libdir and --with-geos-libdir configure switches.

Support for build Tru64 build.

Use Jade for generating documentation.

Don't link postgres to more libs than required.

Initial support for PostgreSQL 8.2.

## A.77 Release 1.1.2

Release date: 2006/03/30

This is a bugfix release including some new functions and portability enhancements. Upgrade is *encouraged*.

### A.77.1 Upgrade procedure

If you are upgrading from release 1.0.3 or later follow the **soft upgrade** procedure.

If you are upgrading from a release *between 1.0.0RC6 and 1.0.2* (inclusive) and really want a live upgrade read the **upgrade section** of the 1.0.3 release notes chapter.

Upgrade from any release prior to 1.0.0RC6 requires an **hard upgrade**.

### A.77.2 Bug fixes

BUGFIX in SnapToGrid() computation of output bounding box

BUGFIX in EnforceRHR()

jdbc2 SRID handling fixes in JTS code

Fixed support for 64bit archs

### A.77.3 New functionalities

Regress tests can now be run *before* postgis installation

New affine() matrix transformation functions

New rotate{,X,Y,Z}() function

Old translating and scaling functions now use affine() internally

Embedded access control in estimated\_extent() for builds against postgresql >= 8.0.0

### A.77.4 Other changes

More portable ./configure script

Changed ./run\_test script to have more sane default behaviour

## A.78 Release 1.1.1

Release date: 2006/01/23

This is an important Bugfix release, upgrade is *highly recommended*. Previous version contained a bug in postgis\_restore.pl preventing **hard upgrade** procedure to complete and a bug in GEOS-2.2+ connector preventing GeometryCollection objects to be used in topological operations.

### A.78.1 Upgrade procedure

If you are upgrading from release 1.0.3 or later follow the **soft upgrade** procedure.

If you are upgrading from a release *between 1.0.0RC6 and 1.0.2* (inclusive) and really want a live upgrade read the **upgrade section** of the 1.0.3 release notes chapter.

Upgrade from any release prior to 1.0.0RC6 requires an **hard upgrade**.

### A.78.2 Bug fixes

Fixed a premature exit in postgis\_restore.pl

BUGFIX in geometrycollection handling of GEOS-CAPI connector

Solaris 2.7 and MingW support improvements

BUGFIX in line\_locate\_point()

Fixed handling of postgresql paths

BUGFIX in line\_substring()

Added support for localized cluster in regress tester

### A.78.3 New functionalities

New Z and M interpolation in line\_substring()

New Z and M interpolation in line\_interpolate\_point()

added NumInteriorRing() alias due to OpenGIS ambiguity

## A.79 Release 1.1.0

Release date: 2005/12/21

This is a Minor release, containing many improvements and new things. Most notably: build procedure greatly simplified; transform() performance drastically improved; more stable GEOS connectivity (CAPI support); lots of new functions; draft topology support.

It is *highly recommended* that you upgrade to GEOS-2.2.x before installing PostGIS, this will ensure future GEOS upgrades won't require a rebuild of the PostGIS library.

### A.79.1 Credits

This release includes code from Mark Cave Ayland for caching of proj4 objects. Markus Schaber added many improvements in his JDBC2 code. Alex Bodnaru helped with PostgreSQL source dependency relief and provided Debian specfiles. Michael Fuhr tested new things on Solaris arch. David Techer and Gerald Fenoy helped testing GEOS C-API connector. Hartmut Tschauner provided code for the azimuth() function. Devrim GUNDUZ provided RPM specfiles. Carl Anderson helped with the new area building functions. See the [credits](#) section for more names.

### A.79.2 Upgrade

If you are upgrading from release 1.0.3 or later you *DO NOT* need a dump/reload. Simply sourcing the new lwpostgis\_upgrade.sql script in all your existing databases will work. See the [soft upgrade](#) chapter for more information.

If you are upgrading from a release *between 1.0.0RC6 and 1.0.2* (inclusive) and really want a live upgrade read the [upgrade section](#) of the 1.0.3 release notes chapter.

Upgrade from any release prior to 1.0.0RC6 requires an [hard upgrade](#).

### A.79.3 New functions

scale() and transscale() companion methods to translate()

line\_substring()

line\_locate\_point()

M(point)

LineMerge(geometry)

shift\_longitude(geometry)

azimuth(geometry)

locate\_along\_measure(geometry, float8)

locate\_between\_measures(geometry, float8, float8)

SnapToGrid by point offset (up to 4d support)

BuildArea(any\_geometry)

OGC BdPolyFromText(linestring\_wkt, srid)

OGC BdMPolyFromText(linestring\_wkt, srid)

RemovePoint(linestring, offset)

ReplacePoint(linestring, offset, point)

#### **A.79.4 Bug fixes**

Fixed memory leak in polygonize()

Fixed bug in lwgeom\_as\_anytype cast functions

Fixed USE\_GEOS, USE\_PROJ and USE\_STATS elements of postgis\_version() output to always reflect library state.

#### **A.79.5 Function semantic changes**

SnapToGrid doesn't discard higher dimensions

Changed Z() function to return NULL if requested dimension is not available

#### **A.79.6 Performance improvements**

Much faster transform() function, caching proj4 objects

Removed automatic call to fix\_geometry\_columns() in AddGeometryColumns() and update\_geometry\_stats()

#### **A.79.7 JDBC2 works**

Makefile improvements

JTS support improvements

Improved regression test system

Basic consistency check method for geometry collections

Support for (Hex)(E)wkb

Autoprobing DriverWrapper for HexWKB / EWKT switching

fix compile problems in ValueSetter for ancient jdk releases.

fix EWKT constructors to accept SRID=4711; representation

added preliminary read-only support for java2d geometries

#### **A.79.8 Other new things**

Full autoconf-based configuration, with PostgreSQL source dependency relief

GEOS C-API support (2.2.0 and higher)

Initial support for topology modelling

Debian and RPM specfiles

New lwpostgis\_upgrade.sql script

#### **A.79.9 Other changes**

JTS support improvements

Stricter mapping between DBF and SQL integer and string attributes

Wider and cleaner regression test suite

old jdbc code removed from release

obsoleted direct use of postgis\_proc\_upgrade.pl

scripts version unified with release version

---

## A.80 Release 1.0.6

Release date: 2005/12/06

Contains a few bug fixes and improvements.

### A.80.1 Upgrade;

If you are upgrading from release 1.0.3 or later you *DO NOT* need a dump/reload.

If you are upgrading from a release *between 1.0.0RC6 and 1.0.2* (inclusive) and really want a live upgrade read the [upgrade section](#) of the 1.0.3 release notes chapter.

Upgrade from any release prior to 1.0.0RC6 requires an [hard upgrade](#).

### A.80.2 Bug fixes

Fixed palloc(0) call in collection deserializer (only gives problem with --enable-cassert)

Fixed bbox cache handling bugs

Fixed geom\_accum(NULL, NULL) segfault

Fixed segfault in addPoint()

Fixed short-allocation in lwcollection\_clone()

Fixed bug in segmentize()

Fixed bbox computation of SnapToGrid output

### A.80.3 Improvements

Initial support for postgresql 8.2

Added missing SRID mismatch checks in GEOS ops

## A.81 Release 1.0.5

Release date: 2005/11/25

Contains memory-alignment fixes in the library, a segfault fix in loader's handling of UTF8 attributes and a few improvements and cleanups.



#### Note

Return code of shp2pgsql changed from previous releases to conform to unix standards (return 0 on success).

---

### A.81.1 Upgrade;

If you are upgrading from release 1.0.3 or later you *DO NOT* need a dump/reload.

If you are upgrading from a release *between 1.0.0RC6 and 1.0.2* (inclusive) and really want a live upgrade read the [upgrade section](#) of the 1.0.3 release notes chapter.

Upgrade from any release prior to 1.0.0RC6 requires an [hard upgrade](#).

---

### A.81.2 Library changes

Fixed memory alignment problems

Fixed computation of null values fraction in analyzer

Fixed a small bug in the getPoint4d\_p() low-level function

Speedup of serializer functions

Fixed a bug in force\_3dm(), force\_3dz() and force\_4d()

### A.81.3 Loader changes

Fixed return code of shp2pgsql

Fixed back-compatibility issue in loader (load of null shapefiles)

Fixed handling of trailing dots in dbf numerical attributes

Segfault fix in shp2pgsql (utf8 encoding)

### A.81.4 Other changes

Schema aware postgis\_proc\_upgrade.pl, support for postgres 7.2+

New "Reporting Bugs" chapter in manual

## A.82 Release 1.0.4

Release date: 2005/09/09

Contains important bug fixes and a few improvements. In particular, it fixes a memory leak preventing successful build of GiST indexes for large spatial tables.

### A.82.1 Upgrade instructions

If you are upgrading from release 1.0.3 you *DO NOT* need a dump/reload.

If you are upgrading from a release *between 1.0.0RC6 and 1.0.2* (inclusive) and really want a live upgrade read the [upgrade section](#) of the 1.0.3 release notes chapter.

Upgrade from any release prior to 1.0.0RC6 requires an [hard upgrade](#).

### A.82.2 Bug fixes

Memory leak plugged in GiST indexing

Segfault fix in transform() handling of proj4 errors

Fixed some proj4 texts in spatial\_ref\_sys (missing +proj)

Loader: fixed string functions usage, reworked NULL objects check, fixed segfault on MULTILINESTRING input.

Fixed bug in MakeLine dimension handling

Fixed bug in translate() corrupting output bounding box

### A.82.3 Improvements

Documentation improvements  
More robust selectivity estimator  
Minor speedup in distance()  
Minor cleanups  
GiST indexing cleanup  
Looser syntax acceptance in box3d parser

## A.83 Release 1.0.3

Release date: 2005/08/08

Contains some bug fixes - *including a severe one affecting correctness of stored geometries* - and a few improvements.

### A.83.1 **Severe bug fix**

Due to a bug in a bounding box computation routine, the upgrade procedure requires special attention, as bounding boxes cached in the database could be incorrect.

An **hard upgrade** procedure (dump/reload) will force recomputation of all bounding boxes (not included in dumps). This is *required* if upgrading from releases prior to 1.0.0RC6.

If you are upgrading from versions 1.0.0RC6 or up, this release includes a perl script (utils/rebuild\_bbox\_caches.pl) to force recomputation of geometries' bounding boxes and invoke all operations required to propagate eventual changes in them (geometry statistics update, reindexing). Invoke the script after a make install (run with no args for syntax help). Optionally run utils/postgis\_proc\_upgrade.pl to refresh postgis procedures and functions signatures (see **Soft upgrade**).

### A.83.2 Bug fixes

Severe bugfix in lwgeom's 2d bounding box computation  
Bugfix in WKT (-w) POINT handling in loader  
Bugfix in dumper on 64bit machines  
Bugfix in dumper handling of user-defined queries  
Bugfix in create\_undef.pl script

### A.83.3 Improvements

Small performance improvement in canonical input function  
Minor cleanups in loader  
Support for multibyte field names in loader  
Improvement in the postgis\_restore.pl script  
New rebuild\_bbox\_caches.pl util script

## A.84 Release 1.0.2

Release date: 2005/07/04

Contains a few bug fixes and improvements.

---

### A.84.1

If you are upgrading from release 1.0.0RC6 or up you *DO NOT* need a dump/reload.

Upgrading from older releases requires a dump/reload. See the [upgrading](#) chapter for more informations.

### A.84.2 Bug fixes

Fault tolerant btree ops

Memory leak plugged in pg\_error

Rtree index fix

Cleaner build scripts (avoided mix of CFLAGS and CXXFLAGS)

### A.84.3 Improvements

New index creation capabilities in loader (-I switch)

Initial support for postgresql 8.1dev

## A.85 Release 1.0.1

Release date: 2005/05/24

Contains a few bug fixes and some improvements.

### A.85.1

If you are upgrading from release 1.0.0RC6 or up you *DO NOT* need a dump/reload.

Upgrading from older releases requires a dump/reload. See the [upgrading](#) chapter for more informations.

### A.85.2 Library changes

BUGFIX in 3d computation of length\_spheroid()

BUGFIX in join selectivity estimator

### A.85.3 Other changes/additions

BUGFIX in shp2pgsql escape functions

better support for concurrent postgis in multiple schemas

documentation fixes

jdbc2: compile with "-target 1.2 -source 1.2" by default

NEW -k switch for pgsq2shp

NEW support for custom createdb options in postgis\_restore.pl

BUGFIX in pgsq2shp attribute names unicity enforcement

BUGFIX in Paris projections definitions

postgis\_restore.pl cleanups

## A.86 Release 1.0.0

Release date: 2005/04/19

Final 1.0.0 release. Contains a few bug fixes, some improvements in the loader (most notably support for older postgis versions), and more docs.

### A.86.1 **Upgrading**

If you are upgrading from release 1.0.0RC6 you *DO NOT* need a dump/reload.

Upgrading from any other precedent release requires a dump/reload. See the [upgrading](#) chapter for more informations.

### A.86.2 Library changes

BUGFIX in transform() releasing random memory address

BUGFIX in force\_3dm() allocating less memory then required

BUGFIX in join selectivity estimator (defaults, leaks, tuplecount, sd)

### A.86.3 Other changes/additions

BUGFIX in shp2pgsql escape of values starting with tab or single-quote

NEW manual pages for loader/dumper

NEW shp2pgsql support for old (HWGEOM) postgis versions

NEW -p (prepare) flag for shp2pgsql

NEW manual chapter about OGC compliancy enforcement

NEW autoconf support for JTS lib

BUGFIX in estimator testers (support for LWGEOM and schema parsing)

## A.87 Release 1.0.0RC6

Release date: 2005/03/30

Sixth release candidate for 1.0.0. Contains a few bug fixes and cleanups.

### A.87.1 **Upgrading**

You need a dump/reload to upgrade from precedent releases. See the [upgrading](#) chapter for more informations.

### A.87.2 Library changes

BUGFIX in multi()

early return [when noop] from multi()

### A.87.3 Scripts changes

dropped {x,y}{min,max}(box2d) functions

---

#### A.87.4 Other changes

BUGFIX in postgis\_restore.pl scrip

BUGFIX in dumper's 64bit support

### A.88 Release 1.0.0RC5

Release date: 2005/03/25

Fifth release candidate for 1.0.0. Contains a few bug fixes and a improvements.

#### A.88.1 **⚠️**

If you are upgrading from release 1.0.0RC4 you *DO NOT* need a dump/reload.

Upgrading from any other precedent release requires a dump/reload. See the [upgrading](#) chapter for more informations.

#### A.88.2 Library changes

BUGFIX (segfaulting) in box3d computation (yes, another!).

BUGFIX (segfaulting) in estimated\_extent().

#### A.88.3 Other changes

Small build scripts and utilities refinements.

Additional performance tips documented.

### A.89 Release 1.0.0RC4

Release date: 2005/03/18

Fourth release candidate for 1.0.0. Contains bug fixes and a few improvements.

#### A.89.1 **⚠️**

You need a dump/reload to upgrade from precedent releases. See the [upgrading](#) chapter for more informations.

#### A.89.2 Library changes

BUGFIX (segfaulting) in geom\_accum().

BUGFIX in 64bit architectures support.

BUGFIX in box3d computation function with collections.

NEW subselects support in selectivity estimator.

Early return from force\_collection.

Consistency check fix in SnapToGrid().

Box2d output changed back to 15 significant digits.

---

### A.89.3 Scripts changes

NEW distance\_sphere() function.

Changed get\_proj4\_from\_srid implementation to use PL/PGSQL instead of SQL.

### A.89.4 Other changes

BUGFIX in loader and dumper handling of MultiLine shapes

BUGFIX in loader, skipping all but first hole of polygons.

jdbc2: code cleanups, Makefile improvements

FLEX and YACC variables set `*after*` pgsq Makefile.global is included and only if the pgsq `*stripped*` version evaluates to the empty string

Added already generated parser in release

Build scripts refinements

improved version handling, central Version.config

improvements in postgis\_restore.pl

## A.90 Release 1.0.0RC3

Release date: 2005/02/24

Third release candidate for 1.0.0. Contains many bug fixes and improvements.

### A.90.1 **Upgrading**

You need a dump/reload to upgrade from precedent releases. See the [upgrading](#) chapter for more informations.

### A.90.2 Library changes

BUGFIX in transform(): missing SRID, better error handling.

BUGFIX in memory alignment handling

BUGFIX in force\_collection() causing mapserver connector failures on simple (single) geometry types.

BUGFIX in GeometryFromText() missing to add a bbox cache.

reduced precision of box2d output.

prefixed DEBUG macros with PGIS\_ to avoid clash with pgsq one

plugged a leak in GEOS2POSTGIS converter

Reduced memory usage by early releasing query-context pallocated one.

### A.90.3 Scripts changes

BUGFIX in 72 index bindings.

BUGFIX in probe\_geometry\_columns() to work with PG72 and support multiple geometry columns in a single table

NEW bool::text cast

Some functions made IMMUTABLE from STABLE, for performance improvement.

---

## A.90.4 JDBC changes

jdbc2: small patches, box2d/3d tests, revised docs and license.  
jdbc2: bug fix and testcase in for pgjdbc 8.0 type autoregistration  
jdbc2: Removed use of jdk1.4 only features to enable build with older jdk releases.  
jdbc2: Added support for building against pg72jdbc2.jar  
jdbc2: updated and cleaned makefile  
jdbc2: added BETA support for jts geometry classes  
jdbc2: Skip known-to-fail tests against older PostGIS servers.  
jdbc2: Fixed handling of measured geometries in EWKT.

## A.90.5 Other changes

new performance tips chapter in manual  
documentation updates: pgsqll72 requirement, lwpostgis.sql  
few changes in autoconf  
BUILDDATE extraction made more portable  
fixed spatial\_ref\_sys.sql to avoid vacuuming the whole database.  
spatial\_ref\_sys: changed Paris entries to match the ones distributed with 0.x.

## A.91 Release 1.0.0RC2

Release date: 2005/01/26

Second release candidate for 1.0.0 containing bug fixes and a few improvements.

### A.91.1 **Upgrading**

You need a dump/reload to upgrade from precedent releases. See the [upgrading](#) chapter for more informations.

### A.91.2 Library changes

BUGFIX in pointarray box3d computation  
BUGFIX in distance\_spheroid definition  
BUGFIX in transform() missing to update bbox cache  
NEW jdbc driver (jdbc2)  
GEOMETRYCOLLECTION(EMPTY) syntax support for backward compatibility  
Faster binary outputs  
Stricter OGC WKB/WKT constructors

### A.91.3 Scripts changes

More correct STABLE, IMMUTABLE, STRICT uses in lwpostgis.sql  
stricter OGC WKB/WKT constructors

---

## A.91.4 Other changes

Faster and more robust loader (both i18n and not)

Initial autoconf script

## A.92 Release 1.0.0RC1

Release date: 2005/01/13

This is the first candidate of a major postgis release, with internal storage of postgis types redesigned to be smaller and faster on indexed queries.

### A.92.1 Upgrade

You need a dump/reload to upgrade from precedent releases. See the [upgrading](#) chapter for more informations.

### A.92.2 Changes

Faster canonical input parsing.

Lossless canonical output.

EWKB Canonical binary IO with PG>73.

Support for up to 4d coordinates, providing lossless shapefile->postgis->shapefile conversion.

New function: UpdateGeometrySRID(), AsGML(), SnapToGrid(), ForceRHR(), estimated\_extent(), accum().

Vertical positioning indexed operators.

JOIN selectivity function.

More geometry constructors / editors.

PostGIS extension API.

UTF8 support in loader.

---