
**Information technology — Database
languages — SQL multimedia and
application packages —**

**Part 3:
Spatial**

*Technologies de l'information — Langages de bases de données —
Multimédia SQL et paquetages d'application —*

Partie 3: Spatial



Reference number
ISO/IEC 13249-3:2016(E)

Documents
Document Sharing Hub
© ISO/IEC 2016



COPYRIGHT PROTECTED DOCUMENT

© ISO/IEC 2016, Published in Switzerland

All rights reserved. Unless otherwise specified, no part of this publication may be reproduced or utilized otherwise in any form or by any means, electronic or mechanical, including photocopying, or posting on the internet or an intranet, without prior written permission. Permission can be requested from either ISO at the address below or ISO's member body in the country of the requester.

ISO copyright office
Ch. de Blandonnet 8 • CP 401
CH-1214 Vernier, Geneva, Switzerland
Tel. +41 22 749 01 11
Fax +41 22 749 09 47
copyright@iso.org
www.iso.org

Contents	Page
Foreword	xviii
Introduction	xix
1 Scope	1
2 Normative references	1
3 Terms and definitions, notations, and conventions	2
3.1 Terms and definitions	2
3.1.1 Terms and definitions provided in Part 1	2
3.1.2 Terms and definitions provided in Part 3	2
3.1.3 Terms and definitions taken from ISO 19107	7
3.1.4 Terms and definitions taken from ISO 19111	8
3.1.5 Terms and definitions taken from ISO 19148	8
3.2 Notations	9
3.2.1 Notations provided in Part 1	9
3.2.2 Notations provided in Part 3	9
3.3 Conventions	10
3.4 Extended BNF notation for WKT and WKB	10
4 Concepts	11
4.1 Concepts provided in Part 1	11
4.2 Geometry Types	11
4.2.1 ST_Geometry	11
4.2.2 Spatial Relationships using ST_Geometry	19
4.2.3 ST_Point	24
4.2.4 ST_Curve	24
4.2.5 ST_LineString	26
4.2.6 ST_CircularString	26
4.2.7 ST_Circle	28
4.2.8 ST_GeodesicString	28
4.2.9 ST_EllipticalCurve	29
4.2.10 ST_NURBSCurve	30
4.2.11 ST_Clothoid	31
4.2.12 ST_SpiralCurve	32
4.2.13 ST_CompoundCurve	33
4.2.14 ST_Surface	33
4.2.15 ST_CurvePolygon	34
4.2.16 ST_Polygon	35
4.2.17 ST_Triangle	35
4.2.18 ST_PolyhedralSurface	36
4.2.19 ST_TIN	36
4.2.20 ST_CompoundSurface	37
4.2.21 ST_Solid	38
4.2.22 ST_BRepSolid	38
4.2.23 ST_GeomCollection	38
4.2.24 ST_MultiPoint	39
4.2.25 ST_MultiCurve	39
4.2.26 ST_MultiLineString	40
4.2.27 ST_MultiSurface	41
4.2.28 ST_MultiPolygon	42
4.3 Topology-Geometry	42
4.3.1 <topology-name>.ST_NODE	43
4.3.2 <topology-name>.ST_EDGE	43
4.3.3 <topology-name>.ST_FACE	46

4.4	Topology-Network.....	49
4.4.1	<network-name>.ST_NODE	49
4.4.2	<network-name>.ST_LINK	49
4.5	General Routines	52
4.5.1	ST_ShortestUndPath Function	52
4.5.2	ST_ShortestDirPath Function	53
4.6	Spatial Reference System Type.....	53
4.6.1	ST_SpatialRefSys.....	53
4.7	Linear Referencing Types	53
4.7.1	ST_PositionExp	53
4.7.2	ST_LinearElement	54
4.7.3	ST_LRFeature	55
4.7.4	ST_LRCurve	55
4.7.5	ST_LRDirectedEdge	56
4.7.6	ST_StartValue	56
4.7.7	ST_LRM	56
4.7.8	ST_DistanceExp	57
4.7.9	ST_LRMeasure	58
4.7.10	ST_Referent	59
4.7.11	ST_LatOffsetExp	59
4.7.12	ST_VerOffsetExp	59
4.7.13	ST_VectorOffsetExp	60
4.8	Angle and Direction Types.....	60
4.8.1	ST_Angle	60
4.8.2	ST_Direction	61
4.9	Support Types	63
4.9.1	ST_TINElement	63
4.9.2	ST_Vector	63
4.9.3	ST_AffinePlacement	64
4.9.4	ST_NURBSPoint.....	65
4.9.5	ST_Knot	65
4.10	Support Routines	65
4.10.1	ST_Geometry ARRAY and ST_Vector ARRAY Support Routines	65
4.11	Tables with columns using geometry types	67
4.12	The Spatial Information Schema	67
5	Geometry Types	68
5.1	ST_Geometry Type and Routines	68
5.1.1	ST_Geometry Type	68
5.1.2	ST_Dimension Method	83
5.1.3	ST_CoordDim Method	84
5.1.4	ST_GeometryType Method	85
5.1.5	ST_SRID Methods	87
5.1.6	ST_Transform Method	88
5.1.7	ST_IsEmpty Method	89
5.1.8	ST_IsSimple Method	90
5.1.9	ST_3DIsSimple Method	91
5.1.10	ST_IsValid Method	92
5.1.11	ST_Is3D Method	93
5.1.12	ST_IsMeasured Method	94
5.1.13	ST_LocateAlong Method	95
5.1.14	ST_3DLocateAlong Method	96
5.1.15	ST_LocateBetween Method	97
5.1.16	ST_3DLocateBetween Method.....	99
5.1.17	ST_Boundary Method	101
5.1.18	ST_3DBoundary Method	102
5.1.19	ST_Envelope Method	103
5.1.20	ST_EnvelopeAsPts Method	104
5.1.21	ST_MinX Method	105
5.1.22	ST_MaxX Method	106
5.1.23	ST_MinY Method	107

5.1.24	ST_MaxY Method	108
5.1.25	ST_MinZ Method	109
5.1.26	ST_MaxZ Method.....	110
5.1.27	ST_MinM Method.....	111
5.1.28	ST_MaxM Method.....	112
5.1.29	ST_ConvexHull Method	113
5.1.30	ST_Buffer Methods	114
5.1.31	ST_Intersection Method	116
5.1.32	ST_3DIntersection Method.....	117
5.1.33	ST_Union Method.....	118
5.1.34	ST_3DUnion Method.....	119
5.1.35	ST_Difference Method	120
5.1.36	ST_3DDifference Method.....	121
5.1.37	ST_SymDifference Method	122
5.1.38	ST_3DSymDifference Method.....	123
5.1.39	Return Types from ST_Intersection, ST_Union, ST_Difference, and ST_SymDifference ..	124
5.1.40	Return Types from ST_3DIntersection, ST_3DUnion, ST_3DDifference, and ST_3DSymDifference.....	127
5.1.41	ST_Distance Methods.....	128
5.1.42	ST_3DDistance Methods	130
5.1.43	ST_Equals Method	132
5.1.44	ST_3DEquals Method	133
5.1.45	ST_Relate Method	134
5.1.46	ST_Disjoint Method.....	137
5.1.47	ST_3DDisjoint Method	138
5.1.48	ST_Intersects Method.....	139
5.1.49	ST_3DIntersects Method	140
5.1.50	ST_Touches Method	141
5.1.51	ST_Crosses Method.....	142
5.1.52	ST_Within Method	143
5.1.53	ST_Contains Method	144
5.1.54	ST_Overlaps Method	145
5.1.55	Cast.....	146
5.1.56	ST_WKTToSQL Method.....	164
5.1.57	ST_AsText Method.....	165
5.1.58	ST_WKBToSQL Method	166
5.1.59	ST_AsBinary Method.....	167
5.1.60	ST_GMLToSQL Method.....	168
5.1.61	ST_AsGML Method	171
5.1.62	ST_GeomFromText Functions.....	172
5.1.63	ST_GeomFromWKB Functions	173
5.1.64	ST_GeomFromGML Functions	174
5.1.65	ST_Geometry Ordering Definition	176
5.1.66	SQL Transform Functions.....	177
5.1.67	<well-known text representation>	178
5.1.68	<well-known binary representation>.....	203
6	Point Types	253
6.1	ST_Point Type and Routines	253
6.1.1	ST_Point Type	253
6.1.2	ST_Point Methods.....	258
6.1.3	ST_X Methods	265
6.1.4	ST_Y Methods	266
6.1.5	ST_Z Methods.....	267
6.1.6	ST_M Methods.....	268
6.1.7	ST_ExplicitPoint Method	269
6.1.8	ST_PointFromText Functions	270
6.1.9	ST_PointFromWKB Functions.....	271
6.1.10	ST_PointFromGML Functions	272
7	Curve Types.....	273

7.1	ST_Curve Type and Routines	273
7.1.1	ST_Curve Type	273
7.1.2	ST_Length Methods	278
7.1.3	ST_3DLength Methods	280
7.1.4	ST_StartPoint Method	282
7.1.5	ST_EndPoint Method	283
7.1.6	ST_IsClosed Method	284
7.1.7	ST_3DIsClosed Method	285
7.1.8	ST_IsRing Method	286
7.1.9	ST_3DIsRing Method	287
7.1.10	ST_CurveToLine Method	288
7.1.11	ST_DistanceToPoint Methods	289
7.1.12	ST_3DDistanceToPt Methods	291
7.1.13	ST_PointAtDistance Methods	293
7.1.14	ST_3DPtAtDistance Methods	295
7.1.15	ST_PerpPoints Method	297
7.2	ST_LineString Type and Routines	298
7.2.1	ST_LineString Type	298
7.2.2	ST_LineString Methods	301
7.2.3	ST_Points Methods	304
7.2.4	ST_NumPoints Method	306
7.2.5	ST_PointN Method	307
7.2.6	ST_StartPoint Method	308
7.2.7	ST_EndPoint Method	309
7.2.8	ST_LineFromText Functions	310
7.2.9	ST_LineFromWKB Functions	311
7.2.10	ST_LineFromGML Functions	312
7.3	ST_CircularString Type and Routines	313
7.3.1	ST_CircularString Type	313
7.3.2	ST_CircularString Methods	320
7.3.3	ST_Points Methods	325
7.3.4	ST_NumPoints Method	327
7.3.5	ST_PointN Method	328
7.3.6	ST_NumSegments Method	329
7.3.7	ST_SegmentN Method	330
7.3.8	ST_MidPointRep Method	331
7.3.9	ST_Bulge Method	332
7.3.10	ST_BulgeNormal Method	333
7.3.11	ST_Center Method	334
7.3.12	ST_Radius Method	335
7.3.13	ST_StartAngle Method	337
7.3.14	ST_EndAngle Method	338
7.3.15	ST_StartPoint Method	339
7.3.16	ST_EndPoint Method	340
7.3.17	ST_CircularFromTxt Functions	341
7.3.18	ST_CircularFromWKB Functions	342
7.3.19	ST_CircularFromGML Functions	343
7.4	ST_Circle Type and Routines	344
7.4.1	ST_Circle Type	344
7.4.2	ST_Circle Methods	349
7.4.3	ST_Points Methods	353
7.4.4	ST_PointN Method	355
7.4.5	ST_Radius Method	356
7.4.6	ST_Center Method	358
7.4.7	ST_Normal Method	359
7.4.8	ST_StartPoint Method	360
7.4.9	ST_EndPoint Method	361
7.4.10	ST_CircleFromTxt Functions	362
7.4.11	ST_CircleFromWKB Functions	363
7.4.12	ST_CircleFromGML Functions	364

7.5	ST_GeodesicString Type and Routines.....	365
7.5.1	ST_GeodesicString Type.....	365
7.5.2	ST_GeodesicString Methods	369
7.5.3	ST_Points Methods.....	372
7.5.4	ST_NumPoints Method.....	374
7.5.5	ST_PointN Method	375
7.5.6	ST_StartPoint Method.....	376
7.5.7	ST_EndPoint Method	377
7.5.8	ST_GeodesicFromTxt Functions.....	378
7.5.9	ST_GeodesicFromWKB Functions	379
7.5.10	ST_GeodesicFromGML Functions	380
7.6	ST_EllipticalCurve Type and Routines	381
7.6.1	ST_EllipticalCurve Type	381
7.6.2	ST_EllipticalCurve Methods.....	390
7.6.3	ST_RefLocation Methods	398
7.6.4	ST_UAxisLength Methods.....	400
7.6.5	ST_VAxisLength Methods.....	403
7.6.6	ST_StartAngle Methods.....	406
7.6.7	ST_EndAngle Methods	407
7.6.8	ST_StartM Methods.....	408
7.6.9	ST_EndM Methods	410
7.6.10	ST_StartPoint Method.....	412
7.6.11	ST_EndPoint Method	413
7.6.12	ST_EllipticFromTxt Functions	414
7.6.13	ST_EllipticFromWKB Functions.....	415
7.6.14	ST_EllipticFromGML Functions	416
7.7	ST_NURBSCurve Type and Routines	417
7.7.1	ST_NURBSCurve Type	417
7.7.2	ST_NURBSCurve Methods.....	423
7.7.3	ST_Degree Method.....	427
7.7.4	ST_ControlPoints Methods	428
7.7.5	ST_Knots Methods.....	430
7.7.6	ST_StartM Methods.....	431
7.7.7	ST_EndM Methods	433
7.7.8	ST_StartPoint Method.....	435
7.7.9	ST_EndPoint Method	436
7.7.10	ST_NURBSFromTxt Functions	437
7.7.11	ST_NURBSFromWKB Functions.....	438
7.7.12	ST_NURBSFromGML Functions	439
7.8	ST_Clothoid Type and Routines.....	440
7.8.1	ST_Clothoid Type.....	440
7.8.2	ST_Clothoid Methods	448
7.8.3	ST_RefLocation Methods	455
7.8.4	ST_ScaleFactor Methods	457
7.8.5	ST_StartDistance Methods	458
7.8.6	ST_EndDistance Methods.....	461
7.8.7	ST_StartM Methods.....	464
7.8.8	ST_EndM Methods	466
7.8.9	ST_StartPoint Method.....	468
7.8.10	ST_EndPoint Method	469
7.8.11	ST_ClothoidFromTxt Functions	470
7.8.12	ST_ClothoidFromWKB Functions	471
7.8.13	ST_ClothoidFromGML Functions.....	472
7.9	ST_SpiralCurve Type and Routines	473
7.9.1	ST_SpiralCurve Type	473
7.9.2	ST_SpiralCurve Methods.....	481
7.9.3	ST_RefLocation Methods.....	489
7.9.4	ST_Length Methods.....	491
7.9.5	ST_StartCurvature Methods	494
7.9.6	ST_EndCurvature Methods.....	495

7.9.7	ST_SpiralType Methods	496
7.9.8	ST_StartM Methods.....	497
7.9.9	ST_EndM Methods	499
7.9.10	ST_StartPoint Method.....	501
7.9.11	ST_EndPoint Method	502
7.9.12	ST_SpiralFromTxt Functions	503
7.9.13	ST_SpiralFromWKB Functions.....	504
7.9.14	ST_SpiralFromGML Functions	505
7.10	ST_CompoundCurve Type and Routines	506
7.10.1	ST_CompoundCurve Type	506
7.10.2	ST_CompoundCurve Methods	510
7.10.3	ST_Curves Methods.....	513
7.10.4	ST_NumCurves Method.....	515
7.10.5	ST_CurveN Method	516
7.10.6	ST_StartPoint Method.....	517
7.10.7	ST_EndPoint Method	518
7.10.8	ST_CompoundFromTxt Functions	519
7.10.9	ST_CompoundFromWKB Functions.....	520
7.10.10	ST_CompoundFromGML Functions	521
8	Surface Types.....	522
8.1	ST_Surface Type and Routines	522
8.1.1	ST_Surface Type	522
8.1.2	ST_Area Methods.....	525
8.1.3	ST_3DArea Methods	527
8.1.4	ST_Perimeter Methods	529
8.1.5	ST_3DPerimeter Methods	531
8.1.6	ST_Centroid Method	533
8.1.7	ST_3DCentroid Method	534
8.1.8	ST_PointOnSurface Method	535
8.1.9	ST_3DPointOnSurf Method	536
8.1.10	ST_IsWorld Method.....	537
8.1.11	ST_3DIsClosed Method	538
8.1.12	ST_IsShell Method	539
8.2	ST_CurvePolygon Type and Routines	540
8.2.1	ST_CurvePolygon Type	540
8.2.2	ST_CurvePolygon Methods	544
8.2.3	ST_ExteriorRing Methods	547
8.2.4	ST_InteriorRings Methods	550
8.2.5	ST_NumInteriorRing Method	553
8.2.6	ST_InteriorRingN Method.....	554
8.2.7	ST_CurvePolyToPoly Method	555
8.2.8	ST_CPolyFromText Functions.....	556
8.2.9	ST_CPolyFromWKB Functions	557
8.2.10	ST_CPolyFromGML Functions	558
8.3	ST_Polygon Type and Routines	559
8.3.1	ST_Polygon Type	559
8.3.2	ST_Polygon Methods.....	562
8.3.3	ST_ExteriorRing Methods	566
8.3.4	ST_InteriorRings Methods	567
8.3.5	ST_InteriorRingN Method.....	569
8.3.6	ST_PolyFromText Functions	570
8.3.7	ST_PolyFromWKB Functions	571
8.3.8	ST_PolyFromGML Functions.....	572
8.3.9	ST_BdPolyFromText Functions	573
8.3.10	ST_BdPolyFromWKB Functions	575
8.4	ST_Triangle Type and Routines	577
8.4.1	ST_Triangle Type	577
8.4.2	ST_Triangle Methods.....	581
8.4.3	ST_Points Methods.....	585
8.4.4	ST_3DSlope Method	586

8.4.5	ST_ExteriorRing Methods	587
8.4.6	ST_InteriorRings Methods	588
8.4.7	ST_InteriorRingN Method.....	589
8.4.8	ST_TriFromText Functions	590
8.4.9	ST_TriFromWKB Functions	591
8.4.10	ST_TriFromGML Functions.....	592
8.5	ST_PolyhedralSurface Type and Routines.....	593
8.5.1	ST_PolyhedralSurface Type	593
8.5.2	ST_PolyhedralSurface Methods	597
8.5.3	ST_Patches Methods	600
8.5.4	ST_NumPatches Method	603
8.5.5	ST_PatchN Method	604
8.5.6	ST_PhSFromText Functions	605
8.5.7	ST_PhSFromWKB Functions.....	606
8.5.8	ST_PhSFromGML Functions	607
8.6	ST_TIN Type and Routines.....	608
8.6.1	ST_TIN Type	608
8.6.2	ST_TIN Methods	613
8.6.3	ST_TINElements Methods	617
8.6.4	ST_MaxSideLength Methods	619
8.6.5	ST_TINTable Methods.....	621
8.6.6	ST_Clip Method	637
8.6.7	ST_Patches Methods	638
8.6.8	ST_TINFromText Functions	639
8.6.9	ST_TINFromWKB Functions	640
8.6.10	ST_TINFromGML Functions	641
8.7	ST_CompoundSurface Type and Routines	642
8.7.1	ST_CompoundSurface Type	642
8.7.2	ST_CompoundSurface Methods	646
8.7.3	ST_Surfaces Methods.....	649
8.7.4	ST_NumSurfaces Method	651
8.7.5	ST_SurfaceN Method	652
8.7.6	ST_CompSurfFromTxt Functions	653
8.7.7	ST_CompSurfFromWKB Functions	654
8.7.8	ST_CompSurfFromGML Functions	655
9	Solid Types	656
9.1	ST_Solid Type and Routines	656
9.1.1	ST_Solid Type	656
9.1.2	ST_3DSurfaceArea Methods	658
9.1.3	ST_3DVolume Methods	660
9.1.4	ST_3DCentroid Method	662
9.1.5	ST_3DPointOnSolid Method	663
9.2	ST_BRepSolid Type and Routines	664
9.2.1	ST_BRepSolid Type	664
9.2.2	ST_BRepSolid Methods.....	668
9.2.3	ST_ExteriorShell Methods.....	672
9.2.4	ST_InteriorShells Methods.....	674
9.2.5	ST_NumIntShells Method.....	677
9.2.6	ST_InteriorShellN Method	678
9.2.7	ST_BRepFromText Functions	679
9.2.8	ST_BRepFromWKB Functions	680
9.2.9	ST_BRepFromGML Functions.....	681
10	Geometry Collection Types	682
10.1	ST_GeomCollection Type and Routines.....	682
10.1.1	ST_GeomCollection Type	682
10.1.2	ST_GeomCollection Methods	686
10.1.3	ST_Geometries Methods.....	689
10.1.4	ST_NumGeometries Method	691
10.1.5	ST_GeometryN Method	692

10.1.6	ST_GeomCollFromTxt Functions	693
10.1.7	ST_GeomCollFromWKB Functions	694
10.1.8	ST_GeomCollFromGML Functions	695
10.2	ST_MultiPoint Type and Routines	696
10.2.1	ST_MultiPoint Type	696
10.2.2	ST_MultiPoint Methods	699
10.2.3	ST_Geometries Methods	701
10.2.4	ST_MPointFromText Functions	703
10.2.5	ST_MPointFromWKB Functions	704
10.2.6	ST_MPointFromGML Functions	705
10.3	ST_MultiCurve Type and Routines	706
10.3.1	ST_MultiCurve Type	706
10.3.2	ST_MultiCurve Methods	710
10.3.3	ST_IsClosed Method	712
10.3.4	ST_3DIsClosed Method	713
10.3.5	ST_Length Methods	714
10.3.6	ST_3DLength Methods	716
10.3.7	ST_PerpPoints Method	718
10.3.8	ST_Geometries Methods	719
10.3.9	ST_MCurveFromText Functions	721
10.3.10	ST_MCurveFromWKB Functions	722
10.3.11	ST_MCurveFromGML Functions	723
10.4	ST_MultiLineString Type and Routines	724
10.4.1	ST_MultiLineString Type	724
10.4.2	ST_MultiLineString Methods	727
10.4.3	ST_Geometries Methods	729
10.4.4	ST_MLineFromText Functions	731
10.4.5	ST_MLineFromWKB Functions	732
10.4.6	ST_MLineFromGML Functions	733
10.5	ST_MultiSurface Type and Routines	734
10.5.1	ST_MultiSurface Type	734
10.5.2	ST_MultiSurface Methods	738
10.5.3	ST_Area Methods	740
10.5.4	ST_3DArea Methods	742
10.5.5	ST_Perimeter Methods	744
10.5.6	ST_3DPerimeter Methods	746
10.5.7	ST_Centroid Method	748
10.5.8	ST_3DCentroid Method	749
10.5.9	ST_PointOnSurface Method	750
10.5.10	ST_3DPointOnSurf Method	751
10.5.11	ST_Geometries Methods	752
10.5.12	ST_MSurfaceFromTxt Functions	754
10.5.13	ST_MSurfaceFromWKB Functions	755
10.5.14	ST_MSurfaceFromGML Functions	756
10.6	ST_MultiPolygon Type and Routines	757
10.6.1	ST_MultiPolygon Type	757
10.6.2	ST_MultiPolygon Methods	760
10.6.3	ST_Geometries Methods	762
10.6.4	ST_MPolyFromText Functions	764
10.6.5	ST_MPolyFromWKB Functions	765
10.6.6	ST_MPolyFromGML Functions	766
10.6.7	ST_BdMPolyFromText Functions	767
10.6.8	ST_BdMPolyFromWKB Functions	769
11	Topology-Geometry	771
11.1	Topo-Geo Topology Schema	771
11.1.1	Introduction	771
11.1.2	ST_NODE view	772
11.1.3	ST_EDGE view	773
11.1.4	ST_FACE view	774
11.2	Topo-Geo Definition Schema	775

11.2.1	Introduction	775
11.2.2	ST_NODE base table.....	776
11.2.3	ST_EDGE base table.....	777
11.2.4	ST_FACE base table	779
11.3	Topo-Geo Routines	780
11.3.1	ST_AddIsoNode Function	780
11.3.2	ST_MoveIsoNode Procedure	782
11.3.3	ST_RemIsoNode Procedure.....	784
11.3.4	ST_AddIsoEdge Function	785
11.3.5	ST_GetFaceEdges Function	787
11.3.6	ST_ChangeEdgeGeom Procedure	788
11.3.7	ST_RemIsoEdge Procedure	790
11.3.8	ST_NewEdgesSplit Function	792
11.3.9	ST_ModEdgeSplit Function	794
11.3.10	ST_NewEdgeHeal Function	796
11.3.11	ST_ModEdgeHeal Procedure.....	799
11.3.12	ST_AddEdgeNewFaces Function.....	802
11.3.13	ST_AddEdgeModFace Function.....	805
11.3.14	ST_RemEdgeNewFace Function	808
11.3.15	ST_RemEdgeModFace Procedure	810
11.3.16	ST_GetFaceGeometry Function	812
11.3.17	ST_InitTopoGeo Procedure	814
11.3.18	ST_CreateTopoGeo Procedure.....	815
11.3.19	ST_ValidateTopoGeo Function.....	818
12	Topology-Network.....	823
12.1	Topo-Net Network Schema	823
12.1.1	Introduction	823
12.1.2	ST_NODE view	824
12.1.3	ST_LINK view	825
12.2	Topo-Net Definition Schema	826
12.2.1	Introduction	826
12.2.2	ST_NODE base table.....	827
12.2.3	ST_LINK base table.....	828
12.3	Topo-Net Routines	829
12.3.1	ST_AddIsoNetNode Function	829
12.3.2	ST_MoveIsoNetNode Procedure	830
12.3.3	ST_RemIsoNetNode Procedure.....	831
12.3.4	ST_AddLink Function	832
12.3.5	ST_ChangeLinkGeom Procedure.....	834
12.3.6	ST_RemoveLink Procedure	836
12.3.7	ST_InitTopoNet Procedure.....	837
12.3.8	ST_NewLogLinkSplit Function	838
12.3.9	ST_ModLogLinkSplit Function	840
12.3.10	ST_NewGeoLinkSplit Function.....	842
12.3.11	ST_ModGeoLinkSplit Function.....	844
12.3.12	ST_NewLinkHeal Function	846
12.3.13	ST_ModLinkHeal Procedure	849
12.3.14	ST_LogiNetFromTGeo Procedure	852
12.3.15	ST_SpatNetFromTGeo Procedure	854
12.3.16	ST_SpatNetFromGeom Procedure.....	856
12.3.17	ST_ValidLogicalNet Function	858
12.3.18	ST_ValidSpatialNet Function	860
13	General Routines	863
13.1	Shortest Path Routines	863
13.1.1	ST_ShortestUndPath Function.....	863
13.1.2	ST_ShortestDirPath Function.....	866
14	Spatial Reference System Type.....	869
14.1	ST_SpatialRefSys Type and Routines	869
14.1.1	ST_SpatialRefSys Type	869

14.1.2	ST_SpatialRefSys Methods	871
14.1.3	ST_AsWKTSRS Method	872
14.1.4	ST_WKTSRSToSQL Method	873
14.1.5	ST_SRID Method	874
14.1.6	ST_Equals Method	875
14.1.7	ST_OrderingEquals Function	876
14.1.8	ST_WellKnownText SQL Transform Group	877
14.1.9	<spatial reference system>	878
15	Linear Referencing Types	882
15.1	ST_LRM Type and Routines	882
15.1.1	ST_LRM Type	882
15.1.2	ST_LRM Methods	888
15.1.3	ST_LRMID Methods	892
15.1.4	ST_LRMName Methods	893
15.1.5	ST_LRMType Methods	894
15.1.6	ST_UnitOfMeasure Methods	895
15.1.7	ST_Constraints Methods	896
15.1.8	ST_OffsetMeasUnit Methods	897
15.1.9	ST_PosLatOffsetDir Methods	899
15.1.10	ST_PosVerOffsetDir Methods	900
15.1.11	ST_LRMFromText Function	901
15.1.12	ST_LRMFromGML Function	902
15.2	ST_LinearElement Type and Routines	903
15.2.1	ST_LinearElement Type	903
15.2.2	ST_LinearElementID Methods	907
15.2.3	ST_DefaultLRM Methods	908
15.2.4	ST_DefaultMeasure Methods	909
15.2.5	ST_LEType Methods	910
15.2.6	ST_StartValue Methods	911
15.2.7	ST_TranslateToInst Method	914
15.2.8	ST_TranslateToType Method	915
15.2.9	ST_LEFromText Function	916
15.2.10	ST_LEFromGML Function	917
15.3	ST_LRFeature Type and Routines	918
15.3.1	ST_LRFeature Type	918
15.3.2	ST_LRFeature Methods	921
15.3.3	ST_FeatureID Methods	924
15.3.4	ST_Referents Methods	925
15.3.5	ST_LRFeatFromText Function	926
15.3.6	ST_LRFeatFromGML Function	927
15.4	ST_LRCurve Type and Routines	928
15.4.1	ST_LRCurve Type	928
15.4.2	ST_LRCurve Methods	930
15.4.3	ST_Curve Methods	932
15.4.4	ST_Point Method	933
15.4.5	ST_LRPosition Method	934
15.4.6	ST_LRCurveFromText Function	935
15.4.7	ST_LRCurveFromGML Function	936
15.5	ST_LRDirectedEdge Type and Routines	937
15.5.1	ST_LRDirectedEdge Type	937
15.5.2	ST_LRDirectedEdge Methods	940
15.5.3	ST_TopologyType Methods	942
15.5.4	ST_TopoOrNetName Methods	943
15.5.5	ST_EdgeOrLinkID Methods	944
15.5.6	ST_LREdgeFromText Function	945
15.5.7	ST_LREdgeFromGML Function	946
15.6	ST_PositionExp Type and Routines	947
15.6.1	ST_PositionExp Type	947
15.6.2	ST_PositionExp Methods	951
15.6.3	ST_LinearElementID Methods	954

15.6.4	ST_LinearElement Methods	955
15.6.5	ST_LRMID Methods	956
15.6.6	ST_LRM Methods	957
15.6.7	ST_DistanceExp Methods	958
15.6.8	ST_Equals Method	959
15.6.9	ST_PosExpFromText Function.....	960
15.6.10	ST_PosExpFromGML Function	961
15.7	ST_LRMeasure Type and Routines	962
15.7.1	ST_LRMeasure Type	962
15.7.2	ST_LRMeasure Methods	964
15.7.3	ST_Measure Methods	966
15.7.4	ST_UnitOfMeasure Methods	967
15.8	ST_StartValue Type and Routines	968
15.8.1	ST_StartValue Type	968
15.8.2	ST_StartValue Method	970
15.8.3	ST_LRM Methods	971
15.8.4	ST_Measure Methods	972
15.9	ST_DistanceExp Type and Routines.....	973
15.9.1	ST_DistanceExp Type.....	973
15.9.2	ST_DistanceExp Methods	983
15.9.3	ST_DistanceAlong Methods	992
15.9.4	ST_FromRefFealD Methods	993
15.9.5	ST_FromRefName Methods	994
15.9.6	ST_TowardsRefFealD Methods	995
15.9.7	ST_TowardsRefName Methods	996
15.9.8	ST_LatOffsetExp Methods	997
15.9.9	ST_VerOffsetExp Methods	998
15.9.10	ST_VectorOffsetExp Methods	999
15.9.11	ST_DisExpFromText Function.....	1000
15.9.12	ST_DisExpFromGML Function	1001
15.10	ST_Referent Type and Routines.....	1002
15.10.1	ST_Referent Type.....	1002
15.10.2	ST_Referent Methods	1006
15.10.3	ST_ReferentName Methods	1008
15.10.4	ST_ReferentType Methods.....	1009
15.10.5	ST_Position Methods.....	1010
15.10.6	ST_Location Methods.....	1011
15.10.7	ST_ChangePosAndLoc Method	1012
15.11	ST_LatOffsetExp Type and Routines	1013
15.11.1	ST_LatOffsetExp Type.....	1013
15.11.2	ST_LatOffsetExp Methods	1016
15.11.3	ST_OffsetLatDist Methods	1018
15.11.4	ST_FeatureGeometry Methods.....	1019
15.11.5	ST_OffsetRefDesc Methods	1020
15.12	ST_VerOffsetExp Type and Routines	1021
15.12.1	ST_VerOffsetExp Type	1021
15.12.2	ST_VerOffsetExp Methods	1024
15.12.3	ST_OffsetVerDist Methods.....	1026
15.12.4	ST_FeatureGeometry Methods.....	1027
15.12.5	ST_OffsetRefDesc Methods	1028
15.13	ST_VectorOffsetExp Type and Routines	1029
15.13.1	ST_VectorOffsetExp Type	1029
15.13.2	ST_VectorOffsetExp Methods	1031
15.13.3	ST_Vectors Methods.....	1032
15.14	Linear Referencing Well-Known Text	1033
15.14.1	<position expression text representation>	1033
15.14.2	<linear element text representation>	1034
15.14.3	<lrn text representation>	1039
15.14.4	<distance expression text representation>	1042
15.14.5	Additional BNF Productions	1047

16	Angle and Direction Types.....	1050
16.1	ST_Angle Type and Routines	1050
16.1.1	ST_Angle Type	1050
16.1.2	ST_Angle Methods.....	1056
16.1.3	ST_Radians Methods.....	1063
16.1.4	ST_Degrees Methods.....	1064
16.1.5	ST_DegreeComponent Method	1065
16.1.6	ST_MinuteComponent Method	1066
16.1.7	ST_SecondComponent Method.....	1067
16.1.8	ST_String Methods	1068
16.1.9	ST_Gradians Methods	1070
16.1.10	ST_Add Method.....	1071
16.1.11	ST_Subtract Method	1072
16.1.12	ST_Multiply Method	1073
16.1.13	ST_Divide Method	1074
16.1.14	ST_AsText Method.....	1075
16.1.15	ST_GMLToSQL Method.....	1076
16.1.16	ST_AsGML Method	1077
16.1.17	ST_AngleFromText Function	1078
16.1.18	ST_AngleFromGML Function	1079
16.1.19	ST_Angle Ordering Definition.....	1080
16.1.20	SQL Transform Functions.....	1081
16.1.21	<angle text representation>	1082
16.2	ST_Direction Type and Routines.....	1083
16.2.1	ST_Direction Type.....	1083
16.2.2	ST_Direction Methods	1088
16.2.3	ST_Radians Method.....	1093
16.2.4	ST_AngleNAzimuth Methods	1094
16.2.5	ST_AsText Method.....	1095
16.2.6	ST_GMLToSQL Method.....	1096
16.2.7	ST_AsGML Method	1097
16.2.8	ST_RadianBearing Method	1098
16.2.9	ST_DegreesBearing Method	1100
16.2.10	ST_DMSBearing Method	1102
16.2.11	ST_RadianNAzimuth Method	1104
16.2.12	ST_DegreesNAzimuth Method.....	1105
16.2.13	ST_DMSNAzimuth Method	1106
16.2.14	ST_RadianSAzimuth Method	1107
16.2.15	ST_DegreesSAzimuth Method.....	1109
16.2.16	ST_DMSSAzimuth Method	1111
16.2.17	ST_AddAngle Method.....	1112
16.2.18	ST_SubtractAngle Method	1113
16.2.19	ST_DirectionFrmTxt Function	1114
16.2.20	ST_DirectionFrmGML Function	1115
16.2.21	ST_Direction Ordering Definition	1116
16.2.22	SQL Transform Functions.....	1117
16.2.23	<direction text representation>	1118
17	Support Types	1119
17.1	ST_TINElement Type and Routines	1119
17.1.1	ST_TINElement Type	1119
17.1.2	ST_TINElement Methods.....	1123
17.1.3	ST_ElementType Methods	1126
17.1.4	ST_ElementID Methods	1127
17.1.5	ST_ElementTag Methods	1128
17.1.6	ST_ElementGeometry Methods.....	1129
17.1.7	ST_IsEmpty Method.....	1132
17.2	ST_Vector Type and Routines	1133
17.2.1	ST_Vector Type	1133
17.2.2	ST_Vector Methods.....	1139
17.2.3	ST_X Methods	1143

17.2.4	ST_Y Methods	1144
17.2.5	ST_Z Methods.....	1145
17.2.6	ST_Coordinates Method.....	1146
17.2.7	ST_Is3D Method	1147
17.2.8	ST_SRID Methods	1148
17.2.9	ST_IsEmpty Method.....	1149
17.2.10	ST_Equals Method	1150
17.2.11	ST_WKTTToSQL Method.....	1151
17.2.12	ST_AsText Method.....	1152
17.2.13	ST_WKBTToSQL Method	1153
17.2.14	ST_AsBinary Method	1154
17.2.15	ST_GMLToSQL Method	1155
17.2.16	ST_AsGML Method	1157
17.2.17	ST_VectorFromText Functions.....	1158
17.2.18	ST_VectorFromWKB Functions	1159
17.2.19	ST_VectorFromGML Functions	1160
17.2.20	ST_Vector Ordering Definition	1162
17.2.21	SQL Transform Functions.....	1163
17.2.22	<well-known text representation>	1164
17.2.23	<well-known binary representation>.....	1166
17.3	ST_AffinePlacement Type and Routines	1168
17.3.1	ST_AffinePlacement Type	1168
17.3.2	ST_AffinePlacement Method	1171
17.3.3	ST_Location Methods.....	1172
17.3.4	ST_RefDirections Methods	1174
17.3.5	ST_InDimension Method	1176
17.3.6	ST_OutDimension Method	1177
17.3.7	ST_Transform Method	1178
17.3.8	ST_IsEmpty Method.....	1179
17.4	ST_NURBSPoint Type and Routines	1180
17.4.1	ST_NURBSPoint Type	1180
17.4.2	ST_NURBSPoint Method	1182
17.4.3	ST_WeightedPoint Methods	1183
17.4.4	ST_Weight Methods.....	1184
17.4.5	ST_IsEmpty Method.....	1185
17.5	ST_Knot Type and Routines	1186
17.5.1	ST_Knot Type	1186
17.5.2	ST_Knot Method.....	1188
17.5.3	ST_Value Methods	1189
17.5.4	ST_Multiplicity Methods	1190
17.5.5	ST_IsEmpty Method.....	1191
18	Support Routines	1192
18.1	ST_Geometry ARRAY Support Routines.....	1192
18.1.1	ST_MaxDimension Function	1192
18.1.2	ST_CheckSRID Function	1194
18.1.3	ST_GetCoordDim Functions	1196
18.1.4	ST_GetIs3D Function	1200
18.1.5	ST_GetIsMeasured Function	1201
18.1.6	ST_CheckNulls Procedure	1202
18.1.7	ST_CheckConsecDups Procedure.....	1204
18.1.8	ST_ToPointAry Cast Function	1205
18.1.9	ST_ToCurveAry Cast Function.....	1207
18.1.10	ST_ToLineStringAry Cast Function	1209
18.1.11	ST_ToCircularAry Cast Function	1211
18.1.12	ST_ToCircleAry Cast Function	1213
18.1.13	ST_ToGeodesicAry Cast Function.....	1215
18.1.14	ST_ToEllipticalAry Cast Function	1217
18.1.15	ST_ToNURBSAry Cast Function	1219
18.1.16	ST_ToClothoidAry Cast Function	1221
18.1.17	ST_ToSpiralAry Cast Function	1223

18.1.18	ST_ToCompoundArc Cast Function	1225
18.1.19	ST_ToSurfaceArc Cast Function	1227
18.1.20	ST_ToCurvePolyArc Cast Function	1229
18.1.21	ST_ToPolygonArc Cast Function	1231
18.1.22	ST_ToTriangleArc Cast Function	1233
18.1.23	ST_ToPolyhedronArc Cast Function	1235
18.1.24	ST_ToTINArc Cast Function	1237
18.1.25	ST_ToCompSurfArc Cast Function	1239
18.1.26	ST_ToBRepSolidArc Cast Function	1241
19	SQL/MM Spatial Information Schema	1243
19.1	Introduction	1243
19.2	ST_GEOMETRY_COLUMNS view	1243
19.3	ST_SPATIAL_REFERENCE_SYSTEMS view	1244
19.4	ST_UNITS_OF_MEASURE view	1244
19.5	ST_SIZINGS view	1244
19.6	Short name views	1244
20	SQL/MM Spatial Definition Schema	1245
20.1	Introduction	1245
20.2	ST_GEOMETRY_COLUMNS base table	1245
20.3	ST_SPATIAL_REFERENCE_SYSTEMS base table	1245
20.4	ST_UNITS_OF_MEASURE base table	1247
20.5	ST_SIZINGS base table	1247
21	SQL/MM Linear Referencing Information and Definition Schemas	1249
21.1	Information Schema	1249
21.1.1	Introduction	1249
21.1.2	ST_LR_COLUMNS view	1249
21.1.3	ST_LRMS view	1249
21.2	Definition Schemata	1250
21.2.1	Introduction	1250
21.2.2	ST_LR_COLUMNS base table	1250
21.2.3	ST_LRMS base table	1251
22	Status Codes	1253
23	Conformance	1256
23.1	Requirements for conformance	1256
23.2	Features of ISO/IEC 9075 required for this part of ISO/IEC 13249	1256
23.3	Claims of conformance	1256
Annex A		1267
A.1	Implementation-defined Meta-variables	1290
Annex B		1293
Annex C		1294
Annex D		1295
Annex E		1296
Bibliography		1298
Index		1299

Figures	Page
Figure E.1 — Geometry Type Hierarchy Diagram	1296

Tables	Page
Table 1 — Symbols	9
Table 2 — Data Type Codes	15
Table 3 — Cast Codes	15
Table 4 — Supported Casts	16
Table 5 — DE-9IM	20
Table 6 — Parameter Types	124
Table 7 — Return Type Sets.....	124
Table 8 — Return Type Matrix for the ST_Intersection Method	126
Table 9 — Return Type Matrix for the ST_Union Method	126
Table 10 — Return Type Matrix for the ST_Difference Method	126
Table 11 — Return Type Matrix for the ST_SymDifference Method	126
Table 12 — DE-9IM Mapping	136
Table 13 — Cell Values	136
Table 14 — Mapping between ST_Geometry values and GML representation	168
Table 15 — <well-known binary representation> <uint32> Values	250
Table 16 — SQLSTATE class and subclass values	1253

Foreword

ISO (the International Organization for Standardization) and IEC (the International Electrotechnical Commission) form the specialized system for worldwide standardization. National bodies that are members of ISO or IEC participate in the development of International Standards through technical committees established by the respective organization to deal with particular fields of technical activity. ISO and IEC technical committees collaborate in fields of mutual interest. Other international organizations, governmental and non-governmental, in liaison with ISO and IEC, also take part in the work. In the field of information technology, ISO and IEC have established a joint technical committee, ISO/IEC JTC 1.

The procedures used to develop this document and those intended for its further maintenance are described in the ISO/IEC Directives, Part 1. In particular the different approval criteria needed for the different types of document should be noted. This document was drafted in accordance with the editorial rules of the ISO/IEC Directives, Part 2 (see www.iso.org/directives).

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. ISO and IEC shall not be held responsible for identifying any or all such patent rights. Details of any patent rights identified during the development of the document will be in the Introduction and/or on the ISO list of patent declarations received (see www.iso.org/patents).

Any trade name used in this document is information given for the convenience of users and does not constitute an endorsement.

For an explanation on the meaning of ISO specific terms and expressions related to conformity assessment, as well as information about ISO's adherence to the WTO principles in the Technical Barriers to Trade (TBT), see the following URL: [Foreword — Supplementary information](#).

ISO/IEC 13249-3 was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, Subcommittee SC 32, *Data management and interchange*.

This fifth edition cancels and replaces the fourth edition (ISO/IEC 13249—3:2011), which has been technically revised.

ISO/IEC 13249 consists of the following parts, under the general title *Information technology — Database languages — SQL multimedia and application packages*:

- *Part 1: Framework*
- *Part 2: Full-Text*
- *Part 3: Spatial*
- *Part 5: Still Image*
- *Part 6: Data Mining*
- *Part 7: History*

Annexes A to E and the Bibliography of this part of ISO/IEC 13249 are for information only.

Introduction

ISO/IEC 13249 defines multimedia and application-specific types and their associated routines using the features for the creation of user-defined types specified in ISO/IEC 9075, "Information technology - Database languages – SQL".

The organization of this part of ISO/IEC 13249 is as follows:

Clause 1, "Scope", specifies the scope of this part of ISO/IEC 13249.

Clause 2, "Normative references", identifies additional standards that, through reference in this part of ISO/IEC 13249, constitute provisions of this part of ISO/IEC 13249.

Clause 3, "Terms and definitions, notations, and conventions", defines the notations and conventions used in this part of ISO/IEC 13249.

Clause 4, "Concepts", presents concepts used in the definition of this part of ISO/IEC 13249.

Clause 5, "Geometry Types", defines the geometry supertype.

Clause 6, "Point Types", defines primitive 0-dimensional geometry types.

Clause 7, "Curve Types", defines primitive 1-dimensional geometry types.

Clause 8, "Surface Types", defines primitive 2-dimensional geometry types.

Clause 9, "Solid Types", defines primitive 3-dimensional geometry types.

Clause 10, "Geometry Collection Types", defines the geometry collection types.

Clause 11, "Topology-Geometry", defines node, edge, and face topology-geometry primitives.

Clause 12, "Topology-Network", defines node and link topology-network primitives.

Clause 13, "General Routines", defines the routines to determine shortest path in directed or undirected graphs.

Clause 14, "Spatial Reference System Type", defines the user-defined type to manage spatial reference systems.

Clause 15, "Linear Referencing Types", defines user-defined types to manage linear referencing.

Clause 16, "Angle and Direction Types", defines the angles and direction types.

Clause 17, "Support Types", defines supporting types and routines used by this part of ISO/IEC 13249.

Clause 18, "Support Routines", defines supporting functions and procedures used by this part of ISO/IEC 13249.

Clause 19, "SQL/MM Spatial Information Schema" defines the SQL/MM Spatial Information Schema.

Clause 20, "SQL/MM Spatial Definition Schema" defines the SQL/MM Spatial Definition Schema.

Clause 21, "SQL/MM Linear Referencing Information and Definition Schemas" defines the SQL/MM Linear Referencing Information and Definition Schemas.

Clause 22, "Status Codes", defines the SQLSTATE codes used in this part of ISO/IEC 13249.

Clause 23, "Conformance", defines the criteria for conformance to this part of ISO/IEC 13249.

Annex A, "Implementation-defined elements", is an informative Annex. It lists those features for which the body of this part of ISO/IEC 13249 states that the syntax or meaning or effect on the database is partly or wholly implementation-defined, and describes the defining information that an implementer shall provide in each case.

Annex B, "Implementation-dependent elements", is an informative Annex. It lists those features for which the body of this part of ISO/IEC 13249 states explicitly that the meaning or effect on the database is implementation-dependent.

Annex C, "Deprecated features", is an informative Annex. It lists features that the responsible Technical Committee intend will not appear in a future revised version of this part of ISO/IEC 13249.

Annex D, "Incompatibilities with ISO/IEC 13249-3:2006", is an informative Annex. It lists incompatibilities with the previous version of this part of ISO/IEC 13249-3.

Annex E, "Geometry Type Hierarchy", is an informative Annex. It visually describes the inheritance relationship between user-defined types in this part of ISO/IEC 13249.

Bibliography is the last informative Annex. It is a list of selective reading relating to this part of ISO/IEC 13249.

In the text of this part of ISO/IEC 13249, in Clause 5, "Geometry Types", through Clause 20, "SQL/MM Spatial Definition Schema", subclauses begin on a new page. Any resulting blank space is not significant.

The spatial user-defined types defined in this part adhere to the following:

- A spatial user-defined type is generic to spatial data handling. It addresses the need to store, manage and retrieve information based on aspects of spatial data such as geometry, location, and topology.
- A spatial user-defined type does not redefine the database language SQL directly or in combination with another spatial data type.

Implementations of this part of ISO/IEC 13249 may exist in environments that also support geographic information, decision support, data mining, and data warehousing systems.

Application areas addressed by implementations of this part of ISO/IEC 13249 include, but are not restricted to, automated mapping, desktop mapping, facilities management, geoengineering, graphics, linear referencing, location based services, multimedia, and resource management applications.

Information technology — Database languages — SQL multimedia and application packages —

Part 3: Spatial

1 Scope

This part of ISO/IEC 13249:

- a) defines concepts specific to this part of ISO/IEC 13249,
- b) defines spatial user-defined types and their associated routines.

2 Normative references

The following documents, in whole or in part, are normatively referenced in this document and are indispensable for its application. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

ISO/IEC 9075-1, Information technology — Database languages — SQL — Part 1: Framework (SQL/Framework).

ISO/IEC 9075-2, Information technology — Database languages — SQL — Part 1: Foundation (SQL/Foundation).

ISO/IEC 13249-1, Information technology — Database languages — SQL multimedia and application packages — Part 1: Framework.

ISO 19107, Geographic information — Spatial schema.

ISO 19111, Geographic information — Spatial referencing by coordinates.

ISO 19136, Geographic information — Geography Markup Language.

ISO 19136-2, Geographic information — Geography Markup Language - Extended schemas and encoding rules.

ISO 19148, Geographic information — Linear referencing

IEC 559:1989, Binary floating-point arithmetic for microprocessor systems.

3 Terms and definitions, notations, and conventions

3.1 Terms and definitions

3.1.1 Terms and definitions provided in Part 1

For the purposes of this part of ISO/IEC 13249, the terms and definitions given in ISO/IEC 13249-1 apply.

3.1.2 Terms and definitions provided in Part 3

For the purposes of this part of ISO/IEC 13249, the following terms and definitions apply.

3.1.2.1

0-dimensional geometry

a geometry with a geometric dimension of 0 (zero)

3.1.2.2

1-dimensional geometry

a geometry with a geometric dimension of 1 (one)

3.1.2.3

2-dimensional geometry

a geometry with a geometric dimension of 2

3.1.2.4

3-dimensional geometry

a geometry with a geometric dimension of 3

3.1.2.5

angle

a measure of the inclination of two surfaces, or of two coplanar lines, with respect to each other

Note 1 to entry: The angle may be measured by the arc of a circle intercepted between the two lines forming the angle, the center of the circle being the point of intersection). The value of an angle can be expressed in degrees, in degrees, minutes, and seconds, in radians, or in gradians.

3.1.2.6

azimuth

the angle, measured clockwise, between a given direction and a reference direction

Note 1 to entry: The reference direction is usually true north (North Azimuth) or true south (South Azimuth), and the angle is positive and less than 360 degrees or the equivalent in other angle measures.

3.1.2.7

bearing

a representation of a geographic heading that is given by a rotation measured from True North or True South towards East or West

Note 1 to entry: A bearing is specified with three parts: the prefix is either 'N' or 'S' for North or South; an angle in the range of 0 (zero) to 90 degrees, 0 to $\pi/2$ radians, or 0 to 100 gradians; and a suffix of 'E' or 'W' for East or West. For example, a direction of Northeast is defined as the bearing N 45 E, where 45 is in degrees. A bearing of S 30 E is 30 degrees measured counterclockwise from due South and is equivalent to a North azimuth of 150 degrees.

3.1.2.8

boundary of a curve

if the curve is closed, then the empty set; otherwise the set consisting of the end points of the curve

3.1.2.9

boundary of a point

the empty set

3.1.2.10

boundary of a solid

the set of surfaces that delineate the edge of the solid, including interior and exterior shells

3.1.2.11**boundary of a surface**

the set of curves that delineate the edge of the surface, including interior and exterior rings

3.1.2.12**break void**

void whose boundary elevations take precedence over existing surface elevations

3.1.2.13**breakline**

discontinuity in the slope of a TIN surface

Note 1 to entry: When inserting a breakline into a surface, the elevations along the breakline take precedence over the original elevations on the surface.

3.1.2.14**closed curve**

a curve such that its start point is equal to its end point

3.1.2.15**closed surface**

a surface that is isomorphic to the surface of a sphere, or some torus

3.1.2.16**closure**

a topological function to cause an open point-set to include its boundary making the point-set topologically closed

3.1.2.17**control contour**

breakline having a constant elevation throughout its length

3.1.2.18**control point**

position used in the construction of a geometry that partially controls its shape but does not necessarily lie on the geometry

Note 1 to entry: A center of an arc is a control point; poles in b-spline curves are control points.

3.1.2.19**data point**

position used in the construction of a geometry that lies on the geometry

Note 1 to entry: The vertices in a line string are data points, the points used to construct a polynomial spline are data points. Data points can be used as control points, but are often derived after the geometry is constructed.

3.1.2.20**degree**

a unit of measurement for angles such that there are 360 degrees in a circle

3.1.2.21**degrees, minutes, and seconds representation**

a system of measurement for angles in which fractions of a degree can be expressed in terms of minutes and seconds

EXAMPLE *180 00 00.0 for an angle of π radians*

Note 1 to entry: For angles less than zero, only the degrees part is negative, however the positive minutes and seconds values are interpreted as increasing the negative-ness of the angle. An angle of $-180\ 30\ 00.0$ shall be interpreted to mean $-(180\ 30\ 00.0)$.

3.1.2.22**dimension**

geometric dimension

3.1.2.23**direction**

the heading from one place to another

Note 1 to entry: The heading is usually given as an azimuth, but may be given as a bearing

3.1.2.24**distance between two geometries**

the minimum of the distances between all pairs of points composed of one point from each of the two geometries

let g_1 and g_2 be two geometries,
 $\text{dist}(g_1, g_2) = \text{Min} (\{ \text{distance between } p \text{ and } q \mid p \in g_1 \wedge q \in g_2 \})$

3.1.2.25**distance between two points**

the minimum of the lengths of all curves connecting two points

3.1.2.26**drape void**

void whose boundary elevations are ignored

Note 1 to entry: When inserting a drape void into a surface, the elevation of the surface takes precedence over any elevations which may have been included in the drape void boundary.

3.1.2.27**geometry**

the shape and geographic location of a feature

3.1.2.28**GML**

Geography Markup Language as defined in ISO 19136

3.1.2.29**GML representation**

an XML element that is valid against an XML element definition in the Geometry Schemas as defined in ISO 19136

3.1.2.30**gradian**

a unit of measurement for angles such that there are 400 gradians in a circle

3.1.2.31**heading**

a geographic direction that is measured as a rotation from some reference direction

3.1.2.32**heal**

topological operation which amalgamates two edges (or links) into one by deleting their common node or which amalgamates two faces into one by deleting their common edge

Note 1 to entry: Heal is the reverse operation of split. When healing two successive edges (or links), the connecting node is deleted and a single edge (or link) replaces the original two. When healing two adjacent faces, their common edge is deleted and a single face replaces the original two.

3.1.2.33**intersection**

(of two geometries) the set of points that are common to both geometries

3.1.2.34**isolated edge**

an edge which is not part of the boundary of any face and is not connected to any other edge

3.1.2.35**isolated link**

a link which is not connected to any other link

3.1.2.36**linear ring**

a linestring that is closed and simple

3.1.2.37**linestring**

a curve with linear interpolation between control points

3.1.2.38**link**

one-dimensional topological primitive that represents a relationship between two nodes

3.1.2.39**logical network**

a network which contains connectivity information but no geometric information

3.1.2.40**minute**

a unit of measurement for angles such that there are 60 minutes in a degree

3.1.2.41**mod 2 union rule**

operation on a multiset producing a resultant set consisting only of those elements occurring an odd number of times in the original multiset

Note 1 to entry: For example, by applying the mod 2 union rule to the multiset: $A = \{ 10, 20, 20, 30, 30, 30, 40, 40, 40, 40 \}$, the result is the set: $R = \{ 10, 30 \}$. In this example, R contains element 10 and 30 because these elements occur an odd number of times in A. Element 20 and 40 are not in R because they occur an even number of times in A.

3.1.2.42**network**

set of nodes and links

3.1.2.43**non-closed curve**

a curve such that its start point is not equal to its end point

3.1.2.44**non-universal face**

any face other than the universal face

3.1.2.45**North azimuth**

an azimuthal heading whose rotation is measured clockwise from True North

3.1.2.46**patch**

planar component of a polyhedral surface

3.1.2.47**planar graph**

graph that can be drawn in a plane without any of its edges crossing

3.1.2.48**point set**

the representation of a geometry as a finite set or infinite set of points

Note 1 to entry: Mathematical set intersection ($\cap$), set union ($\cup$) and set difference ($-$) operations work on point sets.

3.1.2.49**polygon**

a surface that is defined using linear rings to define its boundary

3.1.2.50**radian**

a unit of measurement for angles such that there are 2π radians in a circle

3.1.2.51**random point**

point of known elevation from which a TIN surface can be generated

3.1.2.52**(regular) void**

area within a TIN surface which signifies that the triangles within (or originally within) this area are voided

Note 1 to entry: When inserting a (regular) void into a surface, the vertices of the void boundary are added to the other points in the surface and re-triangulation (perhaps only local to the boundary) occurs.

3.1.2.53

ring

a curve that is closed and simple

3.1.2.54

rotation

an angle with a specific sense, which may be either clockwise or counterclockwise

3.1.2.55

second

a unit of measurement for angles such that there are 60 seconds in a minute

3.1.2.56

slope

<line segment> ratio of (a) the absolute difference in z coordinate values of the two end points to (b) the distance between them, ignoring their z coordinates

3.1.2.57

slope

<triangle> maximal slope of any line segment in a triangle

3.1.2.58

soft break

surface discontinuity which behaves as a breakline except that contour lines may be smoothed where they cross a soft break

3.1.2.59

South azimuth

an azimuthal heading whose rotation is measured clockwise from True South

3.1.2.60

spatial network

a network which contains both connectivity and geometric information

3.1.2.61

spatially 2D equals

a 2D test for equivalence between geometry values

Note 1 to entry: Ignoring z and m coordinate values, if the point sets of two geometry values are equal, then the geometries are spatially 2D equal.

3.1.2.62

spatially 3D equals

a 3D test for equivalence between geometry values

Note 1 to entry: Considering their z (but not m) coordinate values, if the point sets of two geometry values are equal, then the geometries are spatially 3D equal.

3.1.2.63

split

topological operation which creates two edges (or links) from one by inserting a node or which creates two faces from one by inserting an edge.

Note 1 to entry: Split is the reverse operation of heal.

3.1.2.64

stop line

indicator of places where the contour lines of a TIN surface are irregular or not continuous

EXAMPLE *the slope from the top to the bottom of a cliff*

3.1.2.65

TIN

Triangular Irregular Network as defined in ISO 19107

3.1.2.66**topologically closed**

a characteristic of a geometry type that every value includes its own boundary

3.1.2.67**topology-geometry**

a model assuming full planar topology, comprised of nodes, edges, and faces

3.1.2.68**topology-network**

a model for linear applications including node and link topological primitives

3.1.2.69**True North**

a geographic reference direction towards the Earth's geographic North pole

3.1.2.70**True South**

a geographic reference direction towards the Earth's geographic South pole

3.1.2.71**unit of measure**

defined quantity in which dimensioned parameters are expressed

3.1.2.72**universal face**

a face containing everything else in a topology-geometry that is exterior to all other faces in that topology-geometry

3.1.2.73**voided area**

within a TIN surface, a collection of contiguous triangles that remain in the surface but are designated as being void

Note 1 to entry: a voided area is bounded by a break void, drape void, or void.

3.1.2.74**XML element**

an element as defined by ISO 19136

3.1.3 Terms and definitions taken from ISO 19107

For the purposes of this part of ISO/IEC 13249, the following terms defined in ISO 19107 apply.

- a) boundary
- b) buffer
- c) computational topology
- d) connected
- e) connected node
- f) convex hull of a geometric object
- g) coordinate
- h) coordinate dimension
- i) coordinate reference system
- j) coordinate system
- k) curve
- l) cycle
- m) direct position
- n) directed edge

- o) directed face
- p) directed topological object
- q) edge
- r) end node
- s) end point
- t) exterior
- u) face
- v) geometric complex
- w) geometric dimension
- x) geometric object
- y) geometric primitive
- z) homomorphism
- aa) interior
- ab) isolated node
- ac) isomorphism
- ad) node
- ae) point
- af) shell
- ag) solid
- ah) start node
- ai) start point
- aj) surface
- ak) topological complex
- al) topological object
- am) topological primitive

3.1.4 Terms and definitions taken from ISO 19111

For the purposes of this part of ISO/IEC 13249, the following terms defined in ISO 19111 apply.

- a) datum
- b) ellipsoid
- c) flattening
- d) geodetic coordinate system, ellipsoidal coordinate system
- e) height
- f) meridian
- g) prime meridian, zero meridian
- h) projected coordinate system
- i) semi-major axis
- j) semi-minor axis

3.1.5 Terms and definitions taken from ISO 19148

For the purposes of this document, the following terms defined in ISO 19148 apply.

- a) attribute event
- b) attributed feature
- c) feature
- d) feature event
- e) linear element
- f) linear referencing
- g) Linear Referencing Method
- h) Linear Referencing System
- i) linear segment
- j) linearly located
- k) linearly located event
- l) linearly referenced location
- m) located feature
- n) locating feature
- o) spatial position

3.2 Notations

3.2.1 Notations provided in Part 1

For the purposes of this part of ISO/IEC 13249, the notations given in ISO/IEC 13249-1 apply.

3.2.2 Notations provided in Part 3

This part of ISO/IEC 13249 uses the prefix 'ST_' for user-defined type, attribute, SQL-invoked routine table and view names.

This part of ISO/IEC 13249 uses the prefix 'ST_Private' for names of certain attributes. The use of 'ST_Private' indicates that the attribute is not for public use.

The real number mathematical constant that represents the circumference of a circle with unit diameter is notated as " π ". This number is transcendental and cannot be represented exactly in any algebraic form, so the precision is implementation-defined.

This part of ISO/IEC 13249 uses the symbols in Table 1 — Symbols.

Table 1 — Symbols

Symbols	Meaning
$\emptyset$	empty set
$\cap$	Intersection
$\cup$	union
$-$	difference
$\in$	is a member of
$\notin$	is not a member of
$\subset$	is a proper subset of
$\subseteq$	is a subset of
$\Leftrightarrow$	if and only if
$\Rightarrow$	implies
$\forall$	for all
$\{ x \mid \dots \}$	set of all x such that ...
$\wedge$	and
$\vee$	or
$\neg$	not

3.3 Conventions

For the purposes of this part of ISO/IEC 13249, the conventions given in ISO/IEC 13249-1 apply.

3.4 Extended BNF notation for WKT and WKB

For defining Well-Known Text and Well-Known Binary strings, the conventions given in ISO/IEC 9075-1, Subclause 6.2 “Notation provided in this International Standard” shall apply. The following Clauses in this standard define well-known text:

geometry: 5.1.67 “<well-known text representation>”

spatial reference system: 14.1.9 “<spatial reference system>”

linear referencing: 15.14 “Linear Referencing Well-Known Text”

angle: 16.1.21 “<angle text representation>”

direction: 16.2.23 “<direction text representation>”

vector: 17.2.22 “<well-known text representation>”

4 Concepts

4.1 Concepts provided in Part 1

ISO/IEC 13249-1 Information Technology - Database Languages — SQL Multimedia and Application Packages — Part 1: Framework, clause 4 “Concepts” contains concepts that are generally applicable to multiple parts of ISO/IEC 13249, including this Part 3: Spatial. Additional concepts specific to Part 3: Spatial are included in the subsequent subclauses of clause 4 “Concepts” of this part.

4.2 Geometry Types

The following geometry types are defined: ST_Geometry, ST_Point, ST_Curve, ST_LineString, ST_CircularString, ST_Circle, ST_GeodesicString, ST_EllipticalCurve, ST_NURBSCurve, ST_Clothoid, ST_SpiralCurve, ST_CompoundCurve, ST_Surface, ST_CurvePolygon, ST_Polygon, ST_Triangle, ST_PolyhedralSurface, ST_TIN, ST_CompoundSurface, ST_Solid, ST_BRepSolid, ST_GeomCollection, ST_MultiPoint, ST_MultiCurve, ST_MultiLineString, ST_MultiSurface, and ST_MultiPolygon. ST_Geometry and its subtype family constitute the *geometry type hierarchy*, which is visually described in Annex E, “Geometry Type Hierarchy”.

ST_Geometry, ST_Curve, ST_Surface, and ST_Solid are not instantiable types. No constructor functions are defined for these types.

ST_Point, ST_LineString, ST_CircularString, ST_Circle, ST_GeodesicString, ST_EllipticalCurve, ST_NURBSCurve, ST_Clothoid, ST_SpiralCurve, ST_CompoundCurve, ST_CurvePolygon, ST_Polygon, ST_Triangle, ST_PolyhedralSurface, ST_TIN, ST_CompoundSurface, ST_BRepSolid, ST_GeomCollection, ST_MultiPoint, ST_MultiCurve, ST_MultiLineString, ST_MultiSurface, and ST_MultiPolygon are instantiable and have constructor functions.

Any geometry type can be used as the type for a column. Declaring a column to be of a particular type implies that any value of the type or of any of its subtypes can be stored in the column.

ST_TINElement is a support type. It is used by the ST_TIN geometry type to specify information about the TIN surface, other than just its triangles. Each ST_TINElement contains an ST_Geometry.

ST_AffinePlacement and ST_Knot are also support types, used to specify certain ST_Curve types. They are not themselves subtypes of ST_Geometry.

4.2.1 ST_Geometry

The ST_Geometry type is the maximal supertype of the geometry type hierarchy. The ST_Geometry type is not instantiable. The instantiable subtypes of the ST_Geometry type are 0-dimensional geometry, 1-dimensional geometry, 2-dimensional geometry, and 3-dimensional geometry types that exist in two-dimensional coordinate space (R^2), three-dimensional coordinate space (R^3) or four-dimensional coordinate space (R^4). ST_Geometry values in R^2 have points with x and y coordinate values. ST_Geometry values in R^3 have points exclusively with x y and z coordinate values or exclusively with x, y and m coordinate values. ST_Geometry values in R^4 have points with x, y, z and m coordinate values.

The z coordinate is a coordinate of a point typically, but not necessarily, considered to represent altitude. The m coordinate is a coordinate of a point representing arbitrary measurement. ST_Geometry values that have the m coordinate value are key to supporting linear networking applications such as street routing, transportation, pipeline, telecommunications network, and utility management.

All instantiable types are defined so that all values are topologically closed (all ST_Geometry values include their boundary).

All locations in a geometry value are in the same spatial reference system.

In all routines, the geometric calculations are done in the spatial reference system of the first ST_Geometry value in the parameter list. If a routine returns an ST_Geometry value, then that value is in the spatial reference system of the first ST_Geometry value in the parameter list. Similarly, if the routine returns a measurement value such as length or area, then those values are returned in the spatial reference system of the first ST_Geometry value in the parameter list.

An implementation may define additional subtypes in the hierarchy that are outside the scope of this part of ISO/IEC 13249. An implementation shall preserve the subtype relationships between geometry types. Given two types *A* and *B* where *B* is an immediate subtype of *A*, an implementation may introduce another type *T* between types *A* and *B*, as an immediate supertype of *A*, or as an immediate subtype of *B*.

4.2.1.1 Methods on ST_Geometry

- 1) ST_Dimension: returns the dimension of an ST_Geometry value. The dimension of an ST_Geometry value is less than or equal to the coordinate dimension.
- 2) ST_CoordDim: returns the coordinate dimension of an ST_Geometry value. The coordinate dimension shall be the same as the coordinate dimension of the spatial reference system for the ST_Geometry value.
- 3) ST_GeometryType: returns the type of the ST_Geometry value as a CHARACTER VARYING value.
- 4) ST_SRID: observes and mutates the spatial reference system identifier of an ST_Geometry value.
- 5) ST_Transform: returns the ST_Geometry value in the specified spatial reference system, considering *z* and *m* coordinate values in the calculations and including them in the resultant geometry.
- 6) ST_IsEmpty: tests if an ST_Geometry value corresponds to the empty set.
- 7) ST_IsSimple: tests if an ST_Geometry value has no anomalous geometric points, such as self intersection or self tangency, ignoring *z* coordinate values in the determination. Subtypes of ST_Geometry will define the specific conditions that cause a value to be classified as simple.
- 8) ST_3DIsSimple: tests if an ST_Geometry value has no anomalous geometric points, such as self intersection or self tangency, considering *z* coordinate values in the determination. Subtypes of ST_Geometry will define the specific conditions that cause a value to be classified as simple.
- 9) ST_IsValid: tests if an ST_Geometry value is well formed.
- 10) ST_Is3D: tests whether an ST_Geometry value has *z* coordinates.
- 11) ST_IsMeasured: tests whether an ST_Geometry value has *m* coordinate values.
- 12) ST_LocateAlong: returns a derived geometry value that matches the specified *m* coordinate value, ignoring *z* coordinate values in the calculations and not including them in the resultant geometry. See Subclause 4.2.1.7, "Measures on ST_Geometry" for more details.
- 13) ST_3DLocateAlong: returns a derived geometry value that matches the specified *m* coordinate value, considering *z* coordinate values in the calculations and including them in the resultant geometry. See Subclause 4.2.1.7, "Measures on ST_Geometry" for more details.
- 14) ST_LocateBetween: returns a derived geometry value that matches the specified range of *m* coordinate values inclusively, ignoring *z* coordinate values in the calculations and not including them in the resultant geometry. See Subclause 4.2.1.7, "Measures on ST_Geometry" for more details.
- 15) ST_3DLocateBetween: returns a derived geometry value that matches the specified range of *m* coordinate values inclusively, considering *z* coordinate values in the calculations and including them in the resultant geometry. See Subclause 4.2.1.7, "Measures on ST_Geometry" for more details.
- 16) ST_Boundary: returns the boundary of an ST_Geometry value, ignoring *z* and *m* coordinate values in the calculations and not including them in the resultant geometry.
- 17) ST_3DBoundary: returns the boundary of an ST_Geometry value, considering *z* coordinate values in the calculations and including them in the resultant geometry.
- 18) ST_Envelope: returns the bounding rectangular polygon of an ST_Geometry value, ignoring *z* and *m* coordinate values in the calculations and not including them in the resultant geometry.
- 19) ST_EnvelopeAsPts: returns the minimum and maximum coordinate values of an ST_Geometry value as two ST_Point values.
- 20) ST_MinX: returns the minimum *x* coordinate value.
- 21) ST_MaxX: returns the maximum *x* coordinate value.
- 22) ST_MinY: returns the minimum *y* coordinate value.

- 23) ST_MaxY: returns the maximum y coordinate value.
- 24) ST_MinZ: returns the minimum z coordinate value.
- 25) ST_MaxZ: returns the maximum z coordinate value.
- 26) ST_MinM: returns the minimum m coordinate value.
- 27) ST_MaxM: returns the maximum m coordinate value.
- 28) ST_ConvexHull: returns the convex hull of an ST_Geometry value, ignoring z and m coordinate values in the calculations and not including them in the resultant geometry.
- 29) ST_Buffer: returns the ST_Geometry value that represents all points whose distance from any point of an ST_Geometry value is less than or equal to a specified distance, ignoring z and m coordinate values in the calculations and not including them in the resultant geometry.
- 30) ST_Intersection: returns the ST_Geometry value that represents the point set intersection of two ST_Geometry values, ignoring z and m coordinate values in the calculations and not including them in the resultant geometry.

Given two geometries *a* and *b*, ST_Intersection is defined as:

$$a.ST\_Intersection(b) \Leftrightarrow \text{Closure}(a \cap b)$$

- 31) ST_3DIntersection: returns the ST_Geometry value that represents the point set intersection of two ST_Geometry values, considering z coordinate values in the calculations and including them in the resultant geometry.
- 32) ST_Union: returns the ST_Geometry value that represents the point set union of two ST_Geometry values, ignoring z and m coordinate values in the calculations and not including them in the resultant geometry.

Given two geometries *a* and *b*, ST_Union is defined as:

$$a.ST\_Union(b) \Leftrightarrow \text{Closure}(a \cup b)$$

- 33) ST_3DUnion: returns the ST_Geometry value that represents the point set union of two ST_Geometry values, considering z coordinate values in the calculations and including them in the resultant geometry.
- 34) ST_Difference: returns the ST_Geometry value that represents the point set difference of two ST_Geometry values, ignoring z and m coordinate values in the calculations and not including them in the resultant geometry.

Given two geometries *a* and *b*, ST_Difference is defined as:

$$a.ST\_Difference(b) \Leftrightarrow \text{Closure}(a - b)$$

- 35) ST_3DDifference: returns the ST_Geometry value that represents the point set difference of two ST_Geometry values, considering z coordinate values in the calculations and including them in the resultant geometry.
- 36) ST_SymDifference: returns the ST_Geometry value that represents the point set symmetric difference of two ST_Geometry values, ignoring z and m coordinate values in the calculations and not including them in the resultant geometry.

Given two geometries *a* and *b*, ST_SymDifference is defined as:

$$a.ST\_SymDifference(b) \Leftrightarrow \text{Closure}(a - b) \cup \text{Closure}(b - a) \Leftrightarrow a.ST\_Difference(b).ST\_Union(b.ST\_Difference(a))$$

- 37) ST_3DSymDifference: returns the ST_Geometry value that represents the point set symmetric difference of two ST_Geometry values, considering z coordinate values in the calculations and including them in the resultant geometry.
- 38) ST_Distance: returns the distance between two geometries, ignoring z and m coordinate values in the calculations.
- 39) ST_3DDistance: returns the distance between two geometries, considering z coordinate values in the calculations.

- 40) ST_WKTToSQL: returns the ST_Geometry value for the specified well-known text representation.
- 41) ST_AsText: returns the well-known text representation for the specified ST_Geometry value.
- 42) ST_WKBToSQL: returns the ST_Geometry value for the specified well-known binary representation.
- 43) ST_AsBinary: returns the well-known binary representation for the specified ST_Geometry value.
- 44) ST_GMLToSQL: returns the ST_Geometry value for the specified GML representation.
- 45) ST_AsGML: returns the GML representation for the specified ST_Geometry value.

4.2.1.2 Functions on ST_Geometry

- 1) ST_GeomFromText: returns an ST_Geometry value, which is transformed from a CHARACTER LARGE OBJECT value that represents the well-known text representation of an ST_Geometry.
- 2) ST_GeomFromWKB: returns an ST_Geometry value, which is transformed from a BINARY LARGE OBJECT value that represents the well-known binary representation of an ST_Geometry.
- 3) ST_GeomFromGML: returns an ST_Geometry value, which is transformed from a CHARACTER LARGE OBJECT value that represents the GML representation of an ST_Geometry.

4.2.1.3 Ordering on ST_Geometry

- 1) ST_OrderingEquals: is the equals only ordering definition for the ST_Geometry type.

4.2.1.4 SQL Transforms on ST_Geometry

- 1) ST_WellKnownText: is the SQL Transform group that transforms an ST_Geometry value to and from a well-known text representation in a CHARACTER LARGE OBJECT value.
- 2) ST_WellKnownBinary: is the SQL Transform group that transforms an ST_Geometry value to and from a well-known binary representation in a BINARY LARGE OBJECT value.
- 3) ST_GML: is the SQL Transform group that transforms an ST_Geometry value to and from a GML representation in a CHARACTER LARGE OBJECT value.

4.2.1.5 Casts on ST_Geometry

The ST_Geometry type supports data conversion of a value from a source value (SV) of data type (SDT) to a target data type (TDT). SDT and TDT are instantiable subtypes of ST_Geometry. The supported types are described in

Table 4 — Supported Casts. The SDT and TDT codes are data type codes (DT) defined in Table 2 — Data Type Codes. The cast codes used in

Table 4 — Supported Casts are defined in Table 3 — Cast Codes.

NOTE For CX, casts are only supported in kind; casts from one CX Data Type to another CX Data Type are not supported.

Table 2 — Data Type Codes

DT	Data Type
P	ST_Point
C	ST_Curve
LS	ST_LineString
CX	any ST_Curve other than ST_LineString or ST_CompoundCurve
CC	ST_CompoundCurve
S	ST_Surface
CP	ST_CurvePolygon
PY	ST_Polygon
T	ST_Triangle
PS	ST_PolyhedralSurface
TN	ST_TIN
CS	ST_CompoundSurface
BS	ST_BRepSolid
GC	ST_GeomCollection
MP	ST_MultiPoint
MC	ST_MultiCurve
MLS	ST_MultiLineString
MS	ST_MultiSurface
MPY	ST_MultiPolygon

Table 3 — Cast Codes

Cast Code	Description
Y	The cast from the source type to the target type is supported
E	If the value is an empty set, then the cast from the source type to the target type is supported.
1-DT	If the value is an empty set or the value contains 1 element of type <i>DT</i> , then the cast from the source type to the target type is supported. The <i>DT</i> codes are defined in Table 2 — Data Type Codes.
1-DT1 1-DT2	If the value is an empty set or the value contains 1 element of type <i>DT1</i> which has 1 element of type <i>DT2</i> , then the cast from the source type to the target type is supported. The <i>DT1</i> and <i>DT2</i> codes are defined DT codes in Table 2 — Data Type Codes.
1-DT1 4-DT2	If the value is an empty set or the value contains 1 element of type <i>DT1</i> which has 4 elements of type <i>DT2</i> , then the cast from the source type to the target type is supported. The <i>DT1</i> and <i>DT2</i> codes are defined DT codes in Table 2 — Data Type Codes.
n-DT	If the value is an empty set or the value contains 1 or more elements of type <i>DT</i> , then the cast from the source type to the target type is supported. The <i>DT</i> codes are defined in Table 2 — Data Type Codes.

Table 4 — Supported Casts

SDT	TDT																
	P	LS	CX	CC	CP	PY	T	PS	TN	CS	BS	GC	MP	MC	MLS	MS	MP Y
P	Y	E	E	E	E	E	E	E	E	E	E	Y	Y	E	E	E	E
LS	E	Y	E	Y	E	E	E	E	E	E	E	Y	E	Y	Y	E	E
CX	E	Y	Y	Y	E	E	E	E	E	E	E	Y	E	Y	Y	E	E
CC	E	Y	1-CX	Y	E	E	E	E	E	E	E	Y	E	Y	Y	E	E
CP	E	E	E	E	Y	Y	1-LS 4-P	Y	1-LS 4-P	Y	E	Y	E	E	E	Y	Y
PY	E	E	E	E	Y	Y	1-LS 4-P	Y	1-LS 4-P	Y	E	Y	E	E	E	Y	Y
T	E	E	E	E	Y	Y	Y	Y	Y	E	E	Y	E	E	E	Y	Y
PS	E	E	E	E	1-PY	1-PY	1-T	Y	E	Y	E	Y	E	E	E	Y	1-PY
TN	E	E	E	E	1-T	1-T	1-T	Y	Y	E	E	Y	E	E	E	Y	1-T
CS	E	E	E	E	1-PS 1-CP	1-PS 1-CP	1-PS 1-T	1-PS	E	Y	E	Y	E	E	E	Y	1-PS 1-PY
BS	E	E	E	E	E	E	E	E	E	E	Y	Y	E	E	E	E	E
GC	1-P	1-C	1-CX	1-C	1-CP	1-CP	1-T	1-PS	1-TN	1-CS	1-BS	Y	n-P	n-C	n-C	n-S	n-S
MP	1-P	E	E	E	E	E	E	E	E	E	E	Y	Y	E	E	E	E
MC	E	1-C	1-CX	1-C	E	E	E	E	E	E	E	Y	E	Y	Y	E	E
MLS	E	1-C	E	1-C	E	E	E	E	E	E	E	Y	E	Y	Y	E	E
MS	E	E	E	E	1-CP	1-CP	1-T	1-PS	1-TN	1-S	E	Y	E	E	E	Y	Y
MPY	E	E	E	E	1-PY	1-PY	1-T	1-PY	E	1-PY	E	Y	E	E	E	Y	Y

4.2.1.6 Use of Z and M coordinate values

An ST_Point value may include a z coordinate value. The z coordinate value traditionally represents the third dimension (i.e. 3D). In a Geographic Information System (GIS) this may be height above or below sea level. For example: A map might have a point identifying the position of a mountain peak by its location on the earth, with the x and y coordinate values, and the height of the mountain, with the z coordinate value.

An ST_Point value may include an m coordinate value. The m coordinate value allows the application environment to associate some measure with the point values. For example: A stream network may be modeled as a multilinestring value with the m coordinate values measuring the distance from the mouth of the stream. ST_LocateBetween may be used to find all the parts of the stream that are between 10 and 12 kilometers from the mouth. There are no constraints on the m coordinate values in an ST_Geometry (e.g. the m coordinate values do not have to be continually increasing along an ST_LineString value).

As a general rule, unless explicitly specified otherwise, when z or m coordinate values are present:

- 1) Observer methods that return ST_Point values shall include the z and m coordinate values.
- 2) Mutator methods shall maintain the ST_Point value z and m coordinate values.
- 3) Spatial operations shall not consider the z or m coordinate values in calculations (e.g. ST_Equals, ST_Length).
- 4) Routines which generate new geometry values (eg. ST_Intersection) shall not consider the z or m coordinate values.
- 5) Cast and transform routines shall maintain the z and m coordinate values.
- 6) Routines converting to or from WKT, WKB, and GML shall honor the z and m coordinate values when possible.
- 7) Routines whose name begins with "ST_3D" (e.g., ST_3DEquals, ST_3DIntersection) shall consider the z coordinate values and if returning a geometry value, shall include z coordinate values in the returned geometry.

When this general rule may lead to ambiguous behavior, the behavior is made explicit in the Description section of the routine specification.

4.2.1.7 Measures on ST_Geometry

The ST_LocateAlong and ST_LocateBetween methods derive geometry values from the given geometry that match a measure *M* or a specific range of measures from the start measure, *SM*, to the end measure, *EM*. The ST_LocateAlong method is a variation of the ST_LocateBetween method where the *SM* and *EM* are equal to *M*.

4.2.1.7.1 Empty Sets

A null value is returned when ST_LocateAlong or ST_LocateBetween are invoked on an empty set.

4.2.1.7.2 Geometry values without m coordinate values.

An empty set of type ST_Point is returned for geometry values without m coordinate values.

4.2.1.7.3 0-dimensional geometry values

Only points in the 0-dimensional geometry values with m coordinate values between *SM* and *EM* inclusively are returned as a multipoint value. If no matching m coordinate values are found, then an empty set of type ST_Point is returned.

For example:

- a) If ST_LocateAlong is invoked with an *M* value of 4 on an ST_MultiPoint value with well-known text representation:

multipoint m(1 0 4, 1 1 1, 1 2 2, 3 1 4, 5 3 4)

then the result is the following ST_MultiPoint value with well-known text representation:

multipoint m(1 0 4, 3 1 4, 5 3 4)

- b) If `ST_LocateBetween` is invoked with an *SM* value of 2 and an *EM* value of 4 on an `ST_MultiPoint` value with well-known text representation:

`multipoint m(1 0 4, 1 1 1, 1 2 2, 3 1 4, 5 3 5, 9 5 3, 7 6 7)`

then the result is the following `ST_MultiPoint` value with well-known text representation:

`multipoint m(1 0 4, 1 2 2, 3 1 4, 9 5 3)`

- c) If `ST_LocateBetween` is invoked with an *SM* value of 1 and an *EM* value of 4 on an `ST_Point` value with well-known text representation:

`point m(7 6 7)`

then the result is the following `ST_Point` value with well-known text representation:

`point m empty`

- d) If `ST_LocateBetween` is invoked with an *SM* value of 7 and an *EM* value of 7 on an `ST_Point` value with well-known text representation:

`point m(7 6 7)`

then the result is the following `ST_MultiPoint` value with well-known text representation:

`multipoint m(7 6 7)`

4.2.1.7.4 1-dimensional geometry value

Interpolation is used to determine any points on the 1-dimensional geometry with an *m* coordinate value between *SM* and *EM* inclusively. The implementation-defined interpolation algorithm is used to estimate values between measured values, usually using a mathematical function. For example, given a measure of 6 and a 2-point linestring where the *m* coordinate value of the start point is 4 and the *m* coordinate value of the end point is 8, since 6 is halfway between 4 and 8, the interpolation algorithm would be a point on the linestring halfway between the start and end points. The interpolation is within a line segment and not across line segments in an `ST_Curve`. The interpolation is within an `ST_Curve` element and not across `ST_Curve` elements in an `ST_MultiCurve`. Interpolation shall be based on the length of the 1-dimensional geometry.

The results are produced in a geometry collection. If there are consecutive points in the 1-dimensional geometry with an *m* coordinate value between *SM* and *EM* inclusively, then a curve value element is added to the geometry collection to represent the curve elements between these consecutive points. Any disconnected points in the 1-dimensional geometry value with *m* coordinate values between *SM* and *EM* inclusively are also added to the geometry collection. If no matching *m* coordinate values are found, then an empty set of type `ST_Point` is returned.

For example:

- a) If `ST_LocateAlong` is invoked with an *M* value of 4 on an `ST_LineString` value with well-known text representation:

`linestring m(1 0 0, 3 1 4, 5 3 4, 5 5 1, 5 6 4, 7 8 4, 9 9 0)`

then the result is the following `ST_MultiLineString` value with well-known text representation:

`multilinestring m((3 1 4, 5 3 4), (5 6 4, 7 8 4))`

- b) If `ST_LocateBetween` is invoked with an *SM* value of 2 and an *EM* value of 4 on an `ST_LineString` value with well-known text representation:

`linestring m(1 0 0, 1 1 1, 1 2 2, 3 1 3, 5 3 4, 9 5 5, 7 6 6)`

then the result is the following `ST_MultiLineString` value with well-known text representation:

`multilinestring m((1 2 2, 3 1 3, 5 3 4))`

- c) If `ST_LocateBetween` is invoked with an *SM* value of 6 and an *EM* value of 9 on an `ST_LineString` value with well-known text representation:

`linestring m(1 0 0, 1 1 1, 1 2 2, 3 1 3, 5 3 4, 9 5 5, 7 6 6)`

then the result is the following `ST_MultiPoint` value with well-known text representation:

multipoint m(7 6 6)

- d) If ST_LocateBetween is invoked with an SM value of 1 and an EM value of 2 on an ST_LineString value with a well-known text representation:

linestring m(0 0 1, 2 2 3, 4 4 2)

then the result is the following ST_GeomCollection value with well-known text representation:

geometrycollection m(linestring m(0 0 1, 1 1 2), point m(4 4 2))

- e) If ST_LocateBetween is invoked with an SM value of 2 and an EM value of 4 on an ST_MultiLineString value with well-known text representation:

multilinestring m((1 0 0, 1 1 1, 1 2 2, 3 1 3), (4 5 3, 5 3 4, 9 5 5, 7 6 6))

then the result is the following ST_MultiLineString value with well-known text representation:

multilinestring m((1 2 2, 3 1 3),(4 5 3, 5 3 4))

- f) If ST_LocateBetween is invoked with an SM value of 1 and an EM value of 3 on an ST_LineString value with well-known text representation:

linestring m(0 0 0, 2 2 2, 4 4 4)

then the result is the following ST_MultiLineString value with well-known text representation:

multilinestring m((1 1 1, 2 2 2, 3 3 3))

- g) If ST_LocateBetween is invoked with an SM value of 7 and an EM value of 9 on an ST_MultiLineString value with well-known text representation:

multilinestring m((1 0 0, 1 1 1, 1 2 2, 3 1 3), (4 5 3, 5 3 4, 9 5 5, 7 6 6))

then the result is the following ST_Point value with well-known text representation:

point m empty

4.2.1.7.5 2-dimensional geometry value

The computation for 2-dimensional geometries is implementation-defined.

4.2.1.7.6 3-dimensional geometry value

3-dimensional geometries do not have m coordinate values.

4.2.2 Spatial Relationships using ST_Geometry

The spatial relationships are methods that are used to test for the existence of a specified topological spatial relationship between two ST_Geometry values. The basic approach to comparing two ST_Geometry values is to make pair-wise tests of the intersections between the interiors, boundaries and exteriors of the two ST_Geometry values and to classify the relationship between the two ST_Geometry values based on the entries in the resulting intersection matrix.

The concepts of interior, boundary and exterior are well defined in general topology. These concepts can be applied in defining spatial relationships between 2-dimensional geometry values in n -dimensional coordinate space (R^n) where n is greater than 1 (one). In order to apply the concepts of interior, boundary and exterior to 0-dimensional geometry and 1-dimensional geometry values in R^n , a combinatorial topology approach is applied. This approach is based on the accepted definitions of the boundaries, interiors and exteriors for simplicial complexes and yields the following results.

The boundary of an ST_Geometry value is a set of ST_Geometry values of the next lower dimension. The boundary of an ST_Point value or an ST_MultiPoint value is the empty set. The boundary of a non-closed ST_Curve consists of the start and end ST_Point values; the boundary of a closed ST_Curve value is the empty set. The boundary of an ST_MultiCurve consists of those ST_Point values that are in the boundaries of an odd number of its element ST_Curve values. The boundary of an ST_Polygon value consists of its set of linear rings. The boundary of an ST_MultiPolygon value consists of the set of linear rings of its ST_Polygon values. The boundary of an arbitrary collection of geometries whose interiors are disjoint consists of geometries drawn from the boundaries of the element geometries by application of the mod 2 union rule.

The domain of ST_Geometry values considered consists of those values that are topologically closed. The interior of an ST_Geometry value consists of those points that are left when the boundary points are removed. The exterior of an ST_Geometry value consists of points not in the interior or boundary.

4.2.2.1 The Dimensionally Extended 9 Intersection Model

Given an ST_Geometry value g , let Interior(g), Boundary(g) and Exterior(g) represent the interior, boundary and exterior of g , respectively. The intersection of any two of Interior(g), Boundary(g) and Exterior(g) can result in a set of ST_Geometry values, x , of mixed dimension. For example, the intersection of the boundaries of two ST_Polygon values may consist of an ST_Point value and an ST_LineString value. Let $x.ST\_Dimension()$ return the maximum dimension (-1, 0, 1, or 2) of x , with a value of -1 corresponding to the dimension of the empty set ($\emptyset$). A dimensionally extended nine intersection matrix (DE-9IM) is specified in Table 5 — DE-9IM.

Table 5 — DE-9IM

	Interior	Boundary	Exterior
Interior	(Interior(a) $\cap$ Interior(b)). ST_Dimension()	(Interior(a) $\cap$ Boundary(b)). ST_Dimension()	(Interior(a) $\cap$ Exterior(b)). ST_Dimension()
Boundary	(Boundary(a) $\cap$ Interior(b)). ST_Dimension()	(Boundary(a) $\cap$ Boundary(b)). ST_Dimension()	(Boundary(a) $\cap$ Exterior(b)). ST_Dimension()
Exterior	(Exterior(a) $\cap$ Interior(b)). ST_Dimension()	(Exterior(a) $\cap$ Boundary(b)). ST_Dimension()	(Exterior(a) $\cap$ Exterior(b)). ST_Dimension()

For topologically closed input geometries computing the dimension of the intersection of the interior, boundary and exterior sets does not have as a prerequisite the explicit computation and representation of these sets. For example, to compute if the interiors of two ST_Polygon values intersect and to ascertain the dimension of this intersection, it is not necessary to explicitly represent the interior of the two polygons (which are topologically open sets) as separate ST_Geometry values.

In most cases the dimension of the intersection value at a cell is highly constrained given the type of the two ST_Geometry values. For example, in the case of comparing an ST_LineString value with an ST_Polygon: the only possible values for the Interior-Interior cell are drawn from $\{-1, 1\}$. In the case of comparing an ST_Polygon with an ST_Polygon, the only possible values for the Interior-Interior cell are drawn from $\{-1, 2\}$. In such cases, no work beyond detecting the intersection is required.

A spatial relationship predicate can be formulated on ST_Geometry values that takes as input a pattern matrix representing the set of acceptable values for the DE-9IM for the two geometries. If the spatial relationship between the two ST_Geometry values corresponds to one of the acceptable values as represented by the pattern matrix, then the predicate returns 1 (one). Otherwise, 0 (zero).

The pattern matrix consists of a set of 9 pattern-values, one for each cell in the matrix. Let p be a pattern value. The possible values of p are $\{T, F, 0, 1, 2, *\}$ and their meanings for any cell where x is the intersection set for the cell are:

Case:

- if $p = T$, then $x.ST\_Dimension() \in \{0, 1, 2\}$, i.e. $x \neq \emptyset$
- if $p = F$, then $x.ST\_Dimension() = -1$, i.e. $x = \emptyset$
- if $p = 0$, then $x.ST\_Dimension() = 0$ (zero)
- if $p = 1$, then $x.ST\_Dimension() = 1$ (one)
- if $p = 2$, then $x.ST\_Dimension() = 2$
- if $p = *$, then $x.ST\_Dimension() \in \{-1, 0, 1, 2\}$, i.e. any value

The pattern matrix can be represented as a <character string literal> with cardinality 9 representing the DE-9IM in row major order. See Subclause 5.3, "<literal>" in Part 2 of ISO/IEC 9075 for the definition of <character string literal>.

4.2.2.2 Common Spatial Relationships based on the DE-9IM

The ST_Relate method based on the pattern matrix enables a large number of spatial relationships to be tested and fine-tuned to the particular relationship being tested. However it is a low level building block and does not have a corresponding natural language equivalent. For this reason, commonly used spatial relationships have been specified: ST_Disjoint, ST_Intersects, ST_Touches, ST_Crosses, ST_Within, ST_Contains and ST_Overlaps. The theoretical basis for these relationships is limited to geometries having two coordinate dimensions. The z and m coordinate values will therefore be ignored when determining if these relationships hold between two geometry values. These spatial relationships have the following properties:

- 1) They are mutually exclusive.
- 2) They provide a complete covering of all topological cases.
- 3) They apply to spatial relationships between two geometries of either the same or different dimension.
- 4) Each predicate can be expressed in terms of a corresponding set of DE-9IM patterns.
- 5) Any realizable DE-9IM can be expressed as a Boolean expression over the 7 predicates, given the ST_Boundary method on the ST_Geometry type and the ST_StartPoint() and ST_EndPoint() method on the ST_Curve type.

4.2.2.3 Spatial Methods using ST_Geometry

- 1) ST_Equals: tests if an ST_Geometry value is spatially 2D equal to another ST_Geometry value. This test is for equivalence between two, possibly quite different, representations. The test may be limited by the resolution of the spatial reference system or the accuracy of the data. An implementation-defined tolerance may be provided such that two points are considered equal if the distance between the points is less than the tolerance.

Given two geometries *a* and *b*, ST_Equals is defined as:

$$\begin{aligned} a.ST\_Equals(b) &\Leftrightarrow \\ (a - b) \cup (b - a) &= \emptyset \Leftrightarrow \\ a.ST\_SymDifference(b).ST\_IsEmpty() \end{aligned}$$

When determining if two geometries are equal, z and m coordinate values are ignored in the calculations.

- 2) ST_3DEquals: tests if an ST_Geometry value is spatially 3D equal to another ST_Geometry value. When determining if two geometries are 3D equal, z (but not m) coordinate values are considered in the calculations.
- 3) ST_Relate: tests if an ST_Geometry value is spatially related to another ST_Geometry value by testing for intersections between the interior, boundary and exterior of the two ST_Geometry values as specified by the intersection matrix.

When determining if two geometries are spatially related, z and m coordinate values are ignored in the calculations.

- 4) ST_Disjoint: tests if an ST_Geometry value is spatially disjoint from another ST_Geometry value.

Given two geometries *a* and *b*, ST_Disjoint is defined as:

$$\begin{aligned} a.ST\_Disjoint(b) &\Leftrightarrow \\ a \cap b &= \emptyset \end{aligned}$$

Expressed in terms of the DE-9IM:

$$\begin{aligned} a.ST\_Disjoint(b) &\Leftrightarrow \\ (\text{Interior}(a) \cap \text{Interior}(b) = \emptyset) \wedge \\ &(\text{Interior}(a) \cap \text{Boundary}(b) = \emptyset) \wedge \\ &(\text{Boundary}(a) \cap \text{Interior}(b) = \emptyset) \wedge \\ &(\text{Boundary}(a) \cap \text{Boundary}(b) = \emptyset) \Leftrightarrow \\ a.ST\_Relate(b, 'FF*FF****') \end{aligned}$$

When determining if two geometries are disjoint, z and m coordinate values are ignored in the calculations.

5) **ST_3DDisjoint**: tests if an **ST_Geometry** value is spatially 3D disjoint from another **ST_Geometry** value. When determining if two geometries are 3D disjoint, z (but not m) coordinate values are considered in the calculations.

6) **ST_Intersects**: tests if an **ST_Geometry** value spatially intersects another **ST_Geometry** value:

Given two geometries *a* and *b*, **ST_Intersects** is defined as:

$a.ST\_Intersects(b) \Leftrightarrow$
 Case $a.ST\_Disjoint(b)$ when 0 then 1 when 1 then 0 else NULL end

When determining if two geometries intersect, z and m coordinate values are ignored in the calculations.

7) **ST_3DIntersects**: tests if an **ST_Geometry** value spatially 3D intersects another **ST_Geometry** value. When determining if two geometries 3D intersect, z (but not m) coordinate values are considered in the calculations.

8) **ST_Touches**: tests if an **ST_Geometry** value spatially touches another **ST_Geometry** value.

Given two geometries *a* and *b*, **ST_Touches** is defined as:

Case:

- a) If $a.ST\_Dimension() = 0$ (zero) and $b.ST\_Dimension() = 0$ (zero), then the null value
- b) Otherwise, $a.ST\_Touches(b) \Leftrightarrow (Interior(a) \cap Interior(b) = \emptyset) \wedge (a \cap b) \neq \emptyset$

Expressed in terms of the DE-9IM:

Case:

- a) If $a.ST\_Dimension() = 0$ (zero) and $b.ST\_Dimension() = 0$ (zero), then the null value
- b) Otherwise:

$a.ST\_Touches(b) \Leftrightarrow$
 $(Interior(a) \cap Interior(b) = \emptyset) \wedge$
 $((Boundary(a) \cap Interior(b) \neq \emptyset) \vee$
 $(Interior(a) \cap Boundary(b) \neq \emptyset) \vee$
 $(Boundary(a) \cap Boundary(b) \neq \emptyset)) \Leftrightarrow$
 Case $a.ST\_Relate(b, 'FT*****') = 1$ OR
 $a.ST\_Relate(b, 'F**T*****') = 1$ OR
 $a.ST\_Relate(b, 'F***T****') = 1$ when TRUE then 1 when FALSE then 0 else NULL end

When determining if two geometries intersect, z and m coordinate values are ignored in the calculations.

9) **ST_Crosses**: tests if an **ST_Geometry** value spatially crosses another **ST_Geometry** value.

Given two geometries *a* and *b*, **ST_Crosses** is defined as:

Case:

- a) If $a.ST\_Dimension() = 2$ or $b.ST\_Dimension() = 0$ (zero), then the null value
- b) Otherwise:

$a.ST\_Crosses(b) \Leftrightarrow$
 $((Interior(a) \cap Interior(b)).ST\_Dimension() <$
 $\max(Interior(a).ST\_Dimension(), Interior(b).ST\_Dimension())) \wedge (a \cap b \neq a) \wedge (a \cap b \neq b)$

Expressed in terms of the DE-9IM:

Case:

- a) If $a.ST\_Dimension() = 2$ or $b.ST\_Dimension() = 0$ (zero), then the null value
- b) If $(a.ST\_Dimension() = 0$ (zero) and $b.ST\_Dimension() = 1$ (one)) or
 $(a.ST\_Dimension() = 0$ (zero) and $b.ST\_Dimension() = 2$) or
 $(a.ST\_Dimension() = 1$ (one) and $b.ST\_Dimension() = 2$), then:

$$\begin{aligned}
 &a.ST\_Crosses(b) \Leftrightarrow \\
 &(\text{Interior}(a) \cap \text{Interior}(b) \neq \emptyset) \wedge (\text{Interior}(a) \cap \text{Exterior}(b) \neq \emptyset) \Leftrightarrow \\
 &a.ST\_Relate(b, 'T*T*****')
 \end{aligned}$$

c) If $a.ST\_Dimension() = 1$ (one) and $b.ST\_Dimension() = 1$ (one), then:

$$\begin{aligned}
 &a.ST\_Crosses(b) \Leftrightarrow \\
 &(\text{Interior}(a) \cap \text{Interior}(b)).ST\_Dimension() = 0 \text{ (zero)} \Leftrightarrow \\
 &a.ST\_Relate(b, '0*****')
 \end{aligned}$$

When determining if two geometries intersect, z and m coordinate values are ignored in the calculations.

10) **ST_Within**: tests if an ST_Geometry value is spatially within another ST_Geometry value.

Given two geometries a and b , **ST_Within** is defined as:

$$a.ST\_Within(b) \Leftrightarrow (a \cap b = a) \wedge (\text{Interior}(a) \cap \text{Exterior}(b) = \emptyset)$$

Expressed in terms of the DE-9IM:

$$\begin{aligned}
 &a.ST\_Within(b) \Leftrightarrow \\
 &(\text{Interior}(a) \cap \text{Interior}(b) \neq \emptyset) \wedge \\
 &(\text{Interior}(a) \cap \text{Exterior}(b) = \emptyset) \wedge (\text{Boundary}(a) \cap \text{Exterior}(b) = \emptyset) \Leftrightarrow \\
 &a.ST\_Relate(b, 'T**F***')
 \end{aligned}$$

When determining if two geometries intersect, z and m coordinate values are ignored in the calculations.

11) **ST_Contains**: tests if an ST_Geometry value spatially contains another ST_Geometry value.

Given two geometries a and b , **ST_Contains** is defined as:

$$a.ST\_Contains(b) \Leftrightarrow b.ST\_Within(a)$$

When determining if two geometries intersect, z and m coordinate values are ignored in the calculations.

12) **ST_Overlaps**: tests if an ST_Geometry value spatially overlaps another ST_Geometry value.

Given two geometries a and b , **ST_Overlaps** is defined as:

Case:

a) If ($a.ST\_Dimension() = 0$ (zero) and $b.ST\_Dimension() = 0$ (zero)) or ($a.ST\_Dimension() = 1$ (one) and $b.ST\_Dimension() = 1$ (one)) or ($a.ST\_Dimension() = 2$ and $b.ST\_Dimension() = 2$), then:

$$\begin{aligned}
 &a.ST\_Overlaps(b) \Leftrightarrow \\
 &(\text{Interior}(a).ST\_Dimension() = \text{Interior}(b).ST\_Dimension() = \\
 &(\text{Interior}(a) \cap \text{Interior}(b)).ST\_Dimension()) \wedge (a \cap b \neq a) \wedge (a \cap b \neq b)
 \end{aligned}$$

b) Otherwise, the null value

Expressed in terms of the DE-9IM:

Case:

a) If ($a.ST\_Dimension() = 0$ (zero) and $b.ST\_Dimension() = 0$ (zero)) or ($a.ST\_Dimension() = 2$ and $b.ST\_Dimension() = 2$), then:

$$\begin{aligned}
 &a.ST\_Overlaps(b) \Leftrightarrow \\
 &(\text{Interior}(a) \cap \text{Interior}(b) \neq \emptyset) \wedge \\
 &(\text{Interior}(a) \cap \text{Exterior}(b) \neq \emptyset) \wedge \\
 &(\text{Exterior}(a) \cap \text{Interior}(b) \neq \emptyset) \Leftrightarrow \\
 &a.ST\_Relate(b, 'T*T***T**')
 \end{aligned}$$

b) If $a.ST\_Dimension() = 1$ (one) and $b.ST\_Dimension() = 1$ (one), then:

$$\begin{aligned}
 &a.ST\_Overlaps(b) \Leftrightarrow \\
 &((Interior(a) \cap Interior(b)).ST\_Dimension() = 1 \text{ (one)}) \wedge \\
 &\quad (Interior(a) \cap Exterior(b) \neq \emptyset) \wedge (Exterior(a) \cap Interior(b) \neq \emptyset) \Leftrightarrow \\
 &a.ST\_Relate(b, '1*T***T**')
 \end{aligned}$$

c) Otherwise, the null value

When determining if two geometries intersect, z and m coordinate values are ignored in the calculations.

4.2.2.4 GML Support for ST_Geometry

An ST_Geometry value may be represented as an XML element in the GML representation. This support does not include support for an XML document using any or all of the XML schemas defined in GML. The provided routines may be used to create ST_Geometry values from specific XML elements from the Geometry Schemas and to generate XML elements from ST_Geometry values. The mapping is defined in Table 14 — Mapping between ST_Geometry values and GML representation.

4.2.3 ST_Point

The ST_Point type is a subtype of ST_Geometry. The ST_Point type is instantiable. An ST_Point value is a 0-dimensional geometry and represents a single location. An ST_Point has an x coordinate value, a y coordinate value, an optional z coordinate value, and an optional m coordinate value. The boundary of an ST_Point value is the empty set. ST_Point values are simple.

4.2.3.1 Methods on ST_Point

- 1) ST_Point: returns an ST_Point value constructed from either:
 - a) the well-known text representation of an ST_Point value;
 - b) the well-known binary representation of an ST_Point value;
 - c) the GML representation of an ST_Point value;
 - d) the specified coordinate values.
- 2) ST_X: observes and mutates the x coordinate value of an ST_Point value.
- 3) ST_Y: observes and mutates the y coordinate value of an ST_Point value.
- 4) ST_Z: observes and mutates the z coordinate value of an ST_Point value.
- 5) ST_M: observes and mutates the m coordinate value of an ST_Point value.
- 6) ST_ExplicitPoint: returns the coordinate values as a DOUBLE PRECISION ARRAY value.

4.2.3.2 Functions on ST_Point

- 1) ST_PointFromText: returns an ST_Point value, which is transformed from a CHARACTER LARGE OBJECT value that represents the well-known text representation of an ST_Point.
- 2) ST_PointFromWKB: returns an ST_Point value, which is transformed from a BINARY LARGE OBJECT value that represents the well-known binary representation of an ST_Point.
- 3) ST_PointFromGML: returns an ST_Point value, which is transformed from a CHARACTER LARGE OBJECT value that represents the GML representation of an ST_Point value.

4.2.4 ST_Curve

The ST_Curve type is a subtype of ST_Geometry. The ST_Curve type is not instantiable. An ST_Curve value is a 1-dimensional geometry usually stored as a sequence of points. The subtype of ST_Curve specifies the form of the interpolation between points.

Topologically, an ST_Curve value is a 1-dimensional geometry that is the homomorphic image of a real, closed interval. An ST_Curve value is not simple if any interior point has the same location as another interior point or a point in the boundary.

An ST_Curve value is closed if its start point is equal to its end point. The boundary of a closed ST_Curve value is the empty set. An ST_Curve value that is simple and closed is called a *ring*. The boundary of a non-closed ST_Curve value consists of its start point and its end point. An ST_Curve value is defined to be topologically closed.

4.2.4.1 Methods on ST_Curve

- 1) ST_Length: returns the length of an ST_Curve value, ignoring z and m coordinate values in the calculations.
- 2) ST_3DLength: returns the length of an ST_Curve value, considering z coordinate values in the calculations.
- 3) ST_StartPoint: returns the ST_Point value that is the start point of an ST_Curve value including existing z and m coordinate values in the resultant geometry.
- 4) ST_EndPoint: returns the ST_Point value that is the end point of an ST_Curve value including existing z and m coordinate values in the resultant geometry.
- 5) ST_IsClosed: tests if an ST_Curve value is closed, ignoring z and m coordinate values in the calculations.
- 6) ST_3DIsClosed: tests if an ST_Curve value is closed, considering z (but not m) coordinate values in the calculations.
- 7) ST_IsRing: tests if an ST_Curve value is a ring, ignoring z coordinate values in the calculations.
- 8) ST_3DIsRing: tests if an ST_Curve value is a ring, considering z coordinate values in the calculations.
- 9) ST_CurveToLine: returns the ST_LineString value approximating the ST_Curve value, considering z and m coordinate values in the calculations and including z and m coordinate values in the resultant geometry.
- 10) ST_DistanceToPoint: returns the distance from the start of the curve measured along the curve to a point on the curve, ignoring z and m coordinate values in the calculations.
- 11) ST_3DDistanceToPt: returns the distance from the start of the curve measured along the curve to a point on the curve, considering z (but not m) coordinate values in the calculations.
- 12) ST_PointAtDistance: returns the ST_Point value that is the specified distance from the start of the curve measured along the curve, ignoring z coordinate values in the calculations and including an interpolated m (but not z) coordinate in the return value.
- 13) ST_3DPtAtDistance: returns the ST_Point value that is the specified distance from the start of the curve measured along the curve, considering z coordinate values in the calculations and including a z and interpolated m coordinate in the return value.
- 14) ST_PerpPoints: returns the geometry representing the perpendicular projection of the given point onto the curve, ignoring z coordinate values in the calculations and including interpolated m (but not z) coordinates in the resultant geometry.

4.2.5 ST_LineString

The ST_LineString type is a subtype of ST_Curve. The ST_LineString type is instantiable. An ST_LineString value has linear interpolation between ST_Point values. Each consecutive pair of ST_Point values defines a line segment. A line is an ST_LineString value with exactly two points. A linear ring is an ST_LineString value that is both closed and simple.

4.2.5.1 Methods on ST_LineString

- 1) ST_LineString: returns an ST_LineString value constructed from either:
 - a) the well-known text representation of an ST_LineString value;
 - b) the well-known binary representation of an ST_LineString value;
 - c) the GML representation of an ST_LineString value;
 - d) the specified ST_Point values.
- 2) ST_Points: observes and mutates the ST_Point collection in the ST_LineString value.
- 3) ST_NumPoints: returns the cardinality of the ST_Point collection in the ST_LineString value.
- 4) ST_PointN: returns the specified element in the ST_Point collection in the ST_LineString value.

4.2.5.2 Functions on ST_LineString

- 1) ST_LineFromText: returns an ST_LineString value, which is transformed from a CHARACTER LARGE OBJECT value that represents the well-known text representation of an ST_LineString.
- 2) ST_LineFromWKB: returns an ST_LineString value, which is transformed from a BINARY LARGE OBJECT value that represents the well-known binary representation of an ST_LineString.
- 3) ST_LineFromGML: returns an ST_LineString value, which is transformed from a CHARACTER LARGE OBJECT value that represents the GML LineString or LineStringSegment representation of an ST_LineString value.

4.2.6 ST_CircularString

The ST_CircularString type is a subtype of ST_Curve. The ST_CircularString type is instantiable. An ST_CircularString value has circular interpolation between ST_Point values. This subtype of ST_Curve consists of one or more circular arc segments connected end to end. The first three points define the first segment. The first point is the start point of the arc. The second point is any intermediate point on the arc other than the start or end point. The third point is the end point of the arc and shall be distinct from the first point. Subsequent segments are defined by their intermediate and end points only, as the start point is implicitly defined as the previous segment's end point.

Let *CHORD1* be the line connecting the start point of a circular arc segment and the intermediate point on the segment. Let *CHORD2* be the line connecting the intermediate point with the end point of this arc segment. The center of the circular arc segment is located at the intersection of the perpendicular bisectors of *CHORD1* and *CHORD2*.

Let distance *R* be the radius of the circular arc segment, equal to the distance from the center of the circular arc segment to the start, intermediate, or end points.

The circular arc segment is the locus of points a distance *R* from the center of the arc, beginning at the start point and ending at the end point of the circular arc segment.

If the start, intermediate, and end points of an arc segment are collinear, then the resultant arc segment degenerates to a straight line for which center and radius are not defined. In this case, the circular arc segment is the locus of points defined by the straight line connecting the start and end points.

An ST_CircularString value with exactly three points is a circular arc. A circular ring is an ST_CircularString value that is both closed and simple.

An ST_CircularString may be specified by using a bulge factor. Each arc (curve segment) is defined by start and end points, a bulge and a normal. Since the arcs can be connected, start points of all but the first arc are not required, as they are the end points of the previous arc.

A single arc ST_CircularString may also be specified by specifying a single control point at the center point of the arc plus the radius and the start and end angles. This representation can be used only in 2D.

4.2.6.1 Methods on ST_CircularString

- 1) ST_CircularString: returns an ST_CircularString value constructed from either:
 - a) the well-known text representation of an ST_CircularString value;
 - b) the well-known binary representation of an ST_CircularString value;
 - c) the GML representation of an ST_CircularString value;
 - d) the GML ArcByBulge representation of an ST_CircularString value;
 - e) the GML ArcStringByBulge representation of an ST_CircularString value;
 - f) the GML ArcByCenterPoint representation of an ST_CircularString value;
 - g) the specified ST_Point values;
 - h) the specified ST_Point control point, DOUBLE PRECISION bulge and ST_Vector bulge normal ARRAY values;
 - i) the specified ST_Point control (center) point, DOUBLE PRECISION radius and ST_Angle start and end angle values.
- 2) ST_Points: observes and mutates the ST_Point collection in the ST_CircularString value.
- 3) ST_NumPoints: returns the cardinality of the ST_Point collection in the ST_CircularString value.
- 4) ST_PointN: returns the specified element in the ST_Point collection in the ST_CircularString value.
- 5) ST_MidPointRep: returns the array of points which identify an ST_CircularString value including start, mid, and end points for each curve segment.
- 6) ST_NumSegments: returns the INTEGER number of curve segments (arcs) for the ST_CircularString.
- 7) ST_SegmentN: returns the Nth curve segment for the ST_CircularString as an ST_CircularString having a single curve segment.
- 8) ST_Bulge: returns the DOUBLE PRECISION bulge value for an ST_CircularString having a single curve segment.
- 9) ST_BulgeNormal: returns the ST_Vector bulge normal value for an ST_CircularString having a single curve segment.
- 10) ST_Center: returns the ST_Point center point value for an ST_CircularString having a single curve segment.
- 11) ST_Radius: returns the DOUBLE PRECISION radius value for an ST_CircularString having a single curve segment.
- 12) ST_StartAngle: returns the ST_Angle start angle value for an ST_CircularString having a single curve segment.
- 13) ST_EndAngle: returns the ST_Angle end angle value for an ST_CircularString having a single curve segment.

4.2.6.2 Functions on ST_CircularString

- 1) ST_CircularFromTxt: returns an ST_CircularString value, which is transformed from a CHARACTER LARGE OBJECT value that represents the well-known text representation of an ST_CircularString.
- 2) ST_CircularFromWKB: returns an ST_CircularString value, which is transformed from a BINARY LARGE OBJECT value that represents the well-known binary representation of an ST_CircularString.
- 3) ST_CircularFromGML: returns an ST_CircularString value, which is transformed from a CHARACTER LARGE OBJECT value that represents the GML Arc, ArcString, ArcByBulge, ArcStringByBulge or ArcByCenter representation of an ST_CircularString.

4.2.7 ST_Circle

The ST_Circle type is a subtype of ST_Curve. The ST_Circle type is instantiable. An ST_Circle value has circular interpolation between ST_Point values.

An ST_Circle is a single arc which is simple and closed. It consists of a sequence of three unique, non-collinear control points. The arc is simply extended past the third control point until the first control point is encountered.

4.2.7.1 Methods on ST_Circle

- 1) ST_Circle: returns an ST_Circle value constructed from either:
 - a) the well-known text representation of an ST_Circle value;
 - b) the well-known binary representation of an ST_Circle value;
 - c) the GML Circle representation of an ST_Circle value;
 - d) the GML CircleByCenterPoint representation of an ST_Circle value;
 - e) the specified ST_Point values;
 - f) the specified ST_Point center point, DOUBLE PRECISION radius and ST_Vector normal vector values;
- 2) ST_Points: observes and mutates the ST_Point collection in the ST_Circle value.
- 3) ST_PointN: returns the specified element in the ST_Point collection in the ST_Circle value.
- 4) ST_Radius: returns the DOUBLE PRECISION radius of the ST_Circle value.
- 5) ST_Center: returns the ST_Point value representing the center of the ST_Circle value.
- 6) ST_Normal: returns the ST_Vector value representing the normal vector of the ST_Circle value.

4.2.7.2 Functions on ST_Circle

- 1) ST_CircleFromTxt: returns an ST_Circle value, which is transformed from a CHARACTER LARGE OBJECT value that represents the well-known text representation of an ST_Circle.
- 2) ST_CircleFromWKB: returns an ST_Circle value, which is transformed from a BINARY LARGE OBJECT value that represents the well-known binary representation of an ST_Circle.
- 3) ST_CircleFromGML: returns an ST_Circle value, which is transformed from a CHARACTER LARGE OBJECT value that represents the GML Circle or CircleByCenterPoint representation of an ST_Circle value.

4.2.8 ST_GeodesicString

An ST_GeodesicString consists of a sequence of two or more control points joined by geodesic curve segments. The control points are a sequence of positions between which the ST_GeodesicString is interpolated using geodesics from the geoid or ellipsoid of the coordinate reference system being used.

The ST_GeodesicString type is a subtype of ST_Curve. The ST_GeodesicString type is instantiable. An ST_GeodesicString value has geodesic interpolation. It is represented by the following attributes:

control points – an ST_Point ARRAY.

4.2.8.1 Methods on ST_GeodesicString

- 1) ST_GeodesicString: returns an ST_GeodesicString value constructed from either:
 - a) the well-known text representation of an ST_GeodesicString value;
 - b) the well-known binary representation of an ST_GeodesicString value;
 - c) the GML representation of an ST_GeodesicString value;
 - d) the specified ST_Point values.
- 2) ST_Points: observes and mutates the ST_Point collection in the ST_GeodesicString value.
- 3) ST_NumPoints: returns the cardinality of the ST_Point collection in the ST_GeodesicString value.

- 4) ST_PointN: returns the specified element in the ST_Point collection in the ST_GeodesicString value.

4.2.8.2 Functions on ST_GeodesicString

- 1) ST_GeodesicFromTxt: returns an ST_GeodesicString value, which is transformed from a CHARACTER LARGE OBJECT value that represents the well-known text representation of an ST_GeodesicString.
- 2) ST_GeodesicFromWKB: returns an ST_GeodesicString value, which is transformed from a BINARY LARGE OBJECT value that represents the well-known binary representation of an ST_GeodesicString.
- 3) ST_GeodesicFromGML: returns an ST_GeodesicString value, which is transformed from a CHARACTER LARGE OBJECT value that represents the GML Geodesic or GeodesicString representation of an ST_GeodesicString value.

4.2.9 ST_EllipticalCurve

The ST_EllipticalCurve type is a subtype of ST_Curve. The ST_EllipticalCurve type is instantiable. An ST_EllipticalCurve value represents a single curve segment having elliptical interpolation. It is represented by the following attributes:

reference location – an affine mapping of type ST_AffinePlacement that describes the local coordinate system with origin at the elliptical curve center.

uAxisLength – the DOUBLE PRECISION elliptical curve axis length along the u axis of its local 2D coordinate system. This is commonly called the "semimajor axis", but is not required to be the larger axis.

vAxisLength – the DOUBLE PRECISION elliptical curve axis length along the v axis of its local 2D coordinate system. This is commonly called the "semiminor axis", but it is not required to be the smaller axis.

start angle – constructive parameter angle of type ST_Angle

end angle – constructive parameter angle of type ST_Angle

start m – (optional) measure value at the start of the curve

end m – (optional) measure value at the end of the curve

4.2.9.1 Methods on ST_EllipticalCurve

- 1) ST_EllipticalCurve: returns an ST_EllipticalCurve value constructed from either:
 - a) the well-known text representation of an ST_EllipticalCurve value;
 - b) the well-known binary representation of an ST_EllipticalCurve value;
 - c) the GML representation of an ST_EllipticalCurve value;

NOTE There is no normative GML type as of GML version 3.2.1. One has formally been proposed.

 - d) the specified reference location ST_AffinePlacement, DOUBLE PRECISION uAxisLength and vAxisLength values, and the start and end ST_Angle values;
 - e) the specified reference location ST_AffinePlacement, DOUBLE PRECISION uAxisLength and vAxisLength values, the start and end ST_Angle values and the DOUBLE PRECISION start and end measure values.
- 2) ST_RefLocation: observes and mutates the ST_AffinePlacement reference location value in the ST_EllipticalCurve value.
- 3) ST_UAxisLength: observes and mutates the DOUBLE PRECISION u axis length value of an ST_EllipticalCurve value.
- 4) ST_VAxisLength: observes and mutates the DOUBLE PRECISION v axis length value of an ST_EllipticalCurve value.
- 5) ST_StartAngle: observes and mutates the ST_Angle start angle value of an ST_EllipticalCurve value.

- 6) ST_EndAngle: observes and mutates the ST_Angle end angle value of an ST_EllipticalCurve value.
- 7) ST_StartM: observes and mutates the measure value at the start of an ST_EllipticalCurve value.
- 8) ST_EndM: observes and mutates the measure value at the end of an ST_EllipticalCurve value.

4.2.9.2 Functions on ST_EllipticalCurve

- 1) ST_EllipticFromTxt: returns an ST_EllipticalCurve value, which is transformed from a CHARACTER LARGE OBJECT value that represents the well-known text representation of an ST_EllipticalCurve.
- 2) ST_EllipticFromWKB: returns an ST_EllipticalCurve value, which is transformed from a BINARY LARGE OBJECT value that represents the well-known binary representation of an ST_EllipticalCurve.
- 3) ST_EllipticFromGML: returns an ST_EllipticalCurve value, which is transformed from a CHARACTER LARGE OBJECT value that represents the GML Ellipse or EllipticalCurve representation of an ST_EllipticalCurve value

NOTE Once one is defined there.

4.2.10 ST_NURBSCurve

The ST_NURBSCurve type is a subtype of ST_Curve having a single, continuous curve segment. The ST_NURBSCurve type is instantiable. An ST_NURBSCurve value has piecewise polynomial of (x,y,z) position as a function of an interval in the knots space. The ST_NURBSCurve data is a "Non-Uniform Rational BSpline" as described by ISO 19107.

An ST_NURBSCurve value is represented by the following attributes:

- degree – the INTEGER degree of the polynomials in the spline
- control points – an ST_NURBSPoint ARRAY which contains ccontrol points which have been adjusted in consideration of their respective weight values.
- knots – ST_Knot ARRAY which contains knot values and their respective multiplicities
- start m – (optional) measure value at the start of the curve
- end m – (optional) measure value at the end of the curve

4.2.10.1 Methods on ST_NURBSCurve

- 1) ST_NURBSCurve: returns an ST_NURBSCurve value constructed from either:
 - a) the well-known text representation of an ST_NURBSCurve value;
 - b) the well-known binary representation of an ST_NURBSCurve value;
 - c) the GML BSpline representation of an ST_NURBSCurve value;
 - d) the specified INTEGER degree, ST_NURBSPoint ARRAY control points and ST_Knot ARRAY knots;
 - e) the specified INTEGER degree, ST_NURBSPoint ARRAY control points, ST_Knot ARRAY knots and the DOUBLE PRECISION start and end measure values.
- 2) ST_Degree: returns the degree of the ST_NURBSCurve value.
- 3) ST_ControlPoints: observes and mutates the control point collection in the ST_NURBSCurve value.
- 4) ST_Knots: observes and mutates the knot collection in the ST_NURBSCurve value.
- 5) ST_StartM: observes and mutates the measure value at the start of an ST_NURBSCurve value.
- 6) ST_EndM: observes and mutates the measure value at the end of an ST_NURBSCurve value.